# Chapter 2: A Gentle Start

Introduction: How successful learning can be achieved in a relatively simplified setting.

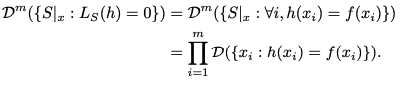
**2.1 A Formal Model**

* The Statistical Learning Framework
* The learner input
  + Domain Set: X. eg. a set of papayas.
    - Represented by a **vector of features**.
    - Domain Points == **instances**
    - X == **instance space**
  + Label Set: Y
    - Restrict label set to be a two-element set {0, 1} or {-1, +1}.
    - eg. 1 represents papaya being tasty and 0 for not-tasty
  + Training Data: S = ((x1, y1), (x2, y2), … , (xm, ym))
    - Finite sequence of X x Y
  + The Learner’s Output: h : X → Y
    - A prediction rule == predictor == hypothesis == classifier
    - Notation: **A(S) ⇒** The hypothesis that a learning algorithm, A, returns upon receiving the training sequence S.
  + A simple data-generation model
    - Assume the instances are generated by some probability distribution.
    - Denote that probability distribution over X by D.
    - **!Important: Learner does not know about D**
    - Aim of Learner: figure out the labeling function f : X → Y
  + Measure of Success: error
    - The probability that the predictor predicts **incorrectly**.
    - P(draw a random instance x, according to the distribution D, s.t. h(x) ≠ f(x))
    - 
* The error of such h is the probability of randomly choosing an example x for which h(x) ≠ f(x).
* The subscript (D, f) indicates that the error is measured w.r.t. the probability distribution D and the correct labeling function f.
* AKA **generalization error, risk,** and **true error of h.**

**2.2 Empirical Risk Minimization**

* Goal of the learning algorithm is to MIN the error r.w.t. the **unknown** D & f.
* True error is **not** directly available to the learner.
* **Training error** is available to the learner.
  + 
  + [m] = {1, … , m}.
  + AKA **empirical error & empirical risk.**
* Overfitting
  + Performance of the algorithm is excellent on the training set but poor on the **true world.**

**2.3 ERM with Inductive Bias**

* Need to guarantee that ERM has good performance w.r.t. the training set as well as over the underlying data distribution.
* What to do → **restricted search space**
  + Before seeing the data, choose a set of predictors == **hypothesis class H**
  + Each h in H is a function mapping from X to Y.
  + Choose h in H with the lowest possible error over S.
    - 
    - argmin ⇒ the set of hypotheses in H that achieve the MIN value of LS(h) over H.
    - By restricting the the learner to choosing a predictor from H, we **bias** it toward a particular set of predictors ⇒ **Inductive Bias.**
    - **Not guaranteed not to overfit.**
* Finite Hypothesis Classes
  + Imposing an **upper bound** on the size of the hypotheses class ⇒ |H| ≤ some value.
  + **The Realizability Assumption (Definition 2.1)**
    - There exists h\* in H s.t. L(D, f)(h\*) = 0.
    - Implies that with probability 1 over random samples, S, where the instances of S are sampled according to D and are labeled by f, we have LS(h) = 0.
  + **The i.i.d. assumption**
    - The examples in the training set are independently and identically distributed according to the distribution D.
    - Every xi in S is freshly sampled according to D and then labeled according to the labeling function, f.
    - Notation: **S ~ Dm**
      * m: the size of the training set S
      * **Dm**: the probability over m-tuples induced by applying D to pick each element of the tuple independently of the other members of the tuple.
  + Randomness in the choice of the predictor hS and L(D, ƒ)(hS).
    - eg. 70% of papayas are tasty. All examples in the training set is **not-tasty**. Then ERMH(S) may be the constant function that labels all papayas as **not-tasty** ⇒ 70% error on the true error.
    - Denote the probability of getting a **nonrepresentative** sample by 𝛿 (delta).
    - 1 - 𝛿 ⇒ **confidence parameter**.
  + Accuracy parameter
    - The quality of the prediction → (epsilon) .
    - If the generalization error is **greater than** epsilon ⇒ failure of the learner.
    - 
    - 
    - 
  + **Union Bound (Lemma 2.2)**
    - For any two sets A, B and a distribution D we have,
    - 
    - Apply the union bound to the equation above,
    - 
    - 
    - **1 - epsilon ⇒ success rate**
    - For each individual sampling of an element of the training set we have,
    - 
    - Combining the previous equation and using the inequality **1 - 𝜖 ≤ e-𝜖,** we obtain that for every h in HBad
    - ****
      * At most (**1 - 𝜖)m** fraction of the training sets would be misleading.
      * **The larger m is, the smaller fraction the misleading set will be.**
    - ****
* **Corollary 2.3**
  + **If .** If m satisfies the condition, then for any labeling function ƒ, and for any distribution D, for which the **realizability assumption** holds, with the probability of at least 1 - 𝛿 over the choice of an i.i.d. sample S of size m, we have that for every ERM hypothesis, it holds that,

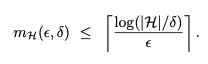
****

# Chapter 3: PAC Learning

Define the main formal learning model - PAC learning model & its extensions.

**3.1 PAC Learning**

* Definition 3.1 (PAC Learnabiilty): A hypothesis class H is PAC learnable if there exists a function mH: (0, 1)2 → N and a learning algorithm with the following property:
  + For every epsilon, delta in (0, 1)
  + For every distribution D over X
  + For every labeling function f: X → {0, 1}
  + if the realizable assumption holds w.r.t. H, D, f, then when running the learning algorithm on m ≥ mH(epsilon, delta) i.i.d. examples generated by D and labeled by f, the algorithm returns a hypothesis h s.t. with probability of at least 1 - delta, L(D, f)(h) ≤ epsilon.
* Epsilon:
  + Determines how far the output classifier can be from the optimal one
* Delta:
  + How likely the classifier is to meet that accuracy requirement
* Sample complexity
  + mH: (0, 1)2
  + How many examples are required to guarantee a probably approximately correct solution
  + Depending on the accuracy (epsilon) and the confidence (delta) parameters.
  + Note that if H is PAC learnable, then there are many functions mH that satisfy the requirements given in the definition. We define the sample complexity to be the **MIN function** in a sense that for any epsilon, delta, mH is the **MIN integer** that satisfies the requirements of PAC learning with accuracy epsilon and confidence delta.
  + Every **finite** hypothesis class is PAC learnable with sample complexity,



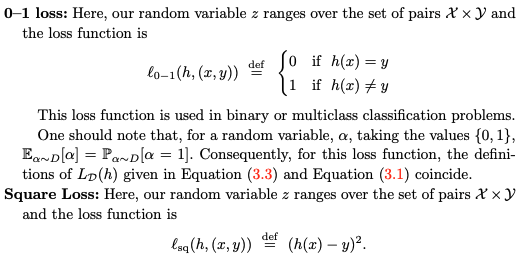
**3.2 A More General Learning Model**

* Generalizations in two aspects
  + Removing the realizability assumption
    - Not realistic for practical learning tasks
  + Learning problems beyond binary classification
    - Many learning tasks take a different form
* 3.2.1 Releasing the realizability assumption - **Agnostic PAC Learning**
  + Realizability Assumption:
    - There exists h\* in H s.t. P[h\*(x) = f(x)] = 1
    - Does not hold in practical examples.
  + The empirical and the True Error Revised (Equation 3.1)
    - True error will be
    - 
    - Empirical Error stays the same,
    - 
  + We wish to find some hypothesis, h: X → Y, that MINs the true error LD(h)
  + Definition 3.3 (Agnostic PAC Learnability) A hypothesis H is agnostic PAC learnable if there exist a function mH: (0, 1)2 → N and a learning algorithm with the following properties,
    - For every epsilon, delta in (0,1)
    - For every distribution D over X, Y
    - When running the learning algorithm on m ≥ mH i.i.d. examples generated by D, the algorithm returns a hypothesis h s.t. with probability of at least 1 - delta,



* 3.2.2 The Scope of Learning Problems Modeled
  + Multiclass classification
  + Regression (Equation 3.4 will be discussed in chapter 9)



* Generalized Loss Function
  + 

# Chapter 4: Uniform Convergence

Develop a general tool, **uniform convergence,** and apply it to show that any finite class is learnable in the agnostic PAC model with general loss functions, **as long as the range loss function is bounded.**

**4.1 Uniform Convergence is Sufficient for Learnability**

* We need that uniformly over all hypotheses in the hypothesis class, the empirical risk will be close to the true risk, as formalized in the following,
* **Definition 4.1 (epsilon-representative example)** A training set S is called epsilon-representative if, for all h in H, |LS(h) - LD(h)| ≤ epsilon
* ****
* **Lemma 4.2:** Assume that a training set S is (epsilon/2)-representative. Then, any output of ERMH(S), namely, any hS in argminh in H LD(h) satisfies the condition above.

**4.2 Finite Classes Are Agnostic PAC Learnable**

* First, apply the union bound (lemma 2.2).
  + Fix some epsilon, delta. Need to find a sample size m s.t. for any distribution D, with probability of at least 1 - delta of the choice of S sampled i.i.d. from D, we have that for all h in H, |LS(h) - LD(h)| ≤ epsilon. That is,



* + In other words, we need to show that **if there exists some h in H, and |LS(h) - LD(h)| > epsilon, then the probability should be less than delta. Namely,**

****

* Apply Union Bound, then we have, (equation 4.1)



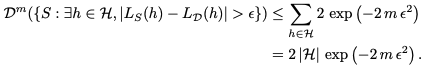
* Second, employ a measure of concentration inequality.
  + Lemma 4.3 (Hoeffding’s Inequality) Let theta\_1, …, theta\_m be a sequence of i.i.d. random variables and assume that for all i, E[theta\_i] = µ and P[a ≤ theta\_i ≤ b] = 1. Then for any epsilon > 0,



* Let theta\_i be the random variable l(h, zi). Since h is fixed and z1, …, zm are sampled i.i.d., it follows that theta1, …, thetam are also i.i.d. random variables.
* Since, LS(h) = 1/m ∑ thetai, from i = 1 to i = m, LD(h) = µ, we then obtain, (equation 4.2)



* Plug equation 4.2 into equation 4.1, then we have,



* Finally, if we choose



* Then,



* Corollary 4.6: Let H be a finite hypothesis class, let Z be a domain, and let *l : H* times *Z → [0, 1]* be a loss function. Then, H enjoys the uniform convergence property if its sample complexity satisfies the following condition,



* Furthermore, the class is agnostically PAC learnable using the ERM algorithm with sample complexity



# Chapter 5: The Bias-Complexity Tradeoff

Note that unless one is careful, the training data can mislead the learner and result in overfitting. To resolve this problem, we can **restrict** the search space to some hypothesis class H.

**5.1 The No-Free-Lunch Theorem**

* There is no **universal learner**: no learner can succeed on all learning tasks
* Theorem 5.1 (No-Free-Lunch): Let A be any learning algorithm for the task of binary classification w.r.t. the 0 - 1 loss over a domain X. Let m be any number smaller than |X| / 2 (training set size). Then, there exists a distribution D over X s.t.
  + There exists a function f: X → {0, 1} with LD(f) = 0
  + **With probability of at least 1/7 over the choice of S ~ Dm we have that LD(A(S)) ≥ ⅛**
  + For every learner, there exists a task on which it fails.
  + Proof (page 61 - 63)
* Corollary 5.2: Let X be an infinite domain set and let H be the set of all functions from X to {0, 1}. Then, **H is NOT PAC learnable.**
  + Assume the class is PAC learnable
  + Choose some epsilon < ⅛, abd delta < 1/7
  + By the definition of PAC learnability, there must be some learning algorithm A and an integer m = mH(epsilon, delta) s.t. for any distribution over X to {0, 1}
    - If for some function f: X → {0, 1}, LD(f) = 0
    - The probability of it > 1 - delta when A is applied to samples S where |S| = m
    - LD(A(S)) ≤ epsilon
  + By No-Free-Lunch Theorem, since |X| > 2m, for every learning algorithm (A included)
    - There exists some distribution D s.t. with probability greater than 1/7 > delta
    - LD(A(S)) > ⅛ > epsilon
  + **Contradiction ⇒** the class is NOT PAC learnable.

**5.2 Error Decomposition**

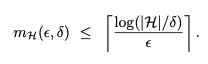
* To prevent the failures mentioned above, we can try to avoid the distributions that will cause us to fail when learning that task.
* Restrictions on hypothesis class
  + The class includes hypothesis that has 0 or small errors
  + Cannot choose the richest class - the class of all functions over the given domain
* Decompose the error of ERM predictor into two components (equation 5.7)
  + 
  + **Approximation Error:** the MIN risk achievable by a predictor in the hypothesis class.
    - How much **inductive bias** we have
    - Does not depend on the sample size
    - Depends on the hypothesis class chosen
    - **Enlarging the hypothesis class can decrease the approximation error**
    - Under the realizability assumption, this error = 0
    - Under agnostic case, this error can be large
  + **Estimation Error:** the difference between the approximation error and the error achieved by the ERM predictor.
    - The quality of this estimation depends on the size of training set, hypothesis class
    - For finite hypothesis class, estimation error increases (**logarithmically**) with the size of hypothesis class and decreases with m.
* To MIN the total risk, we face a tradeoff - **bias-complexity tradeoff**
  + Choosing H to be very rich **decreases** the approximation error but **increases** the estimation error. (rich H → overfitting)
  + Choosing H to be a very small set **increases** the approximation error but **decreases** the estimation error. (small H → underfitting)

# Chapter 6: The VC-Dimension

Infinite class can be also learnable. Finiteness of the hypothesis class is not a necessary condition for learnability.

**6.1 Infinite-Size Classes Can Be Learnable**

* The sample complexity of a hypothesis class is the **upper bound** by the log of its size.



* Simple example of an **infinite-size** hypothesis class that is learnable.
  + Example 6.1: Let H be the set of threshold functions over the real line
    - H = {ha : a in R}, where ha : R → {0, 1} is a function s.t. ha(x) = 1 if x < a
    - H is infinite
  + Lemma 6.1: Let H be the class of thresholds as defined earlier. Then, H is PAC learnable, using the ERM rule, with sample complexity of mH(epsilon, delta) ≤ [log(2/delta) / epsilon] Proof (page 67-68)

**6.2 The VC-Dimension**

* VC-dimension of a hypothesis class gives the correct characterization of its learnability.
* **Definition 6.2 (Restriction of H to C)** A hypothesis class H shatters a finite set C, a subset of domain X, if the restriction of H to C is the set of all functions from C to {0, 1}. That is, |HC| = 2|C|
  + C = {c1, …, cm}
  + HC = {h(c1), …, h(cm)}
* **Definition 6.3 (Shattering)** A hypothesis class H shatters a finite set C, a subset of domain X, if the restriction of H to C is the set of all functions from C to {0, 1}. That is, |HC| = 2|C|
  + Example: Let H be a class of threshold functions over R
    - C = {c1}
    - If we take a = c1 + 1, then ha(c1) = 1 as c1 is in the threshold (c1 < a)
    - If we take a = c1 - 1, then ha(c1) = 0 as c1 is not in the threshold (c1 > a)
    - Therefore, HC is the set of all functions from C to {0, 1}. HC shatters C.
* C = {c1, c2}, c1 ≤ c2
* Then label (0, 1) cannot be output by any h in H
* If c1 is labeled 0, then c2 must also be labeled 0
* Therefore, HC does not shatter C.
* **Corollary 6.4:** Let H be a hypothesis class of functions from X to {0, 1}. Let m be a training set size. Assume that there exists a set C, a subset of domain X, |C| = 2m that is shattered by H. Then for any learning algorithm, A, there exists a distribution D over X x {0, 1} and a predictor h in H s.t. LD(h) = 0, but **with probability of at least 1/7 over the choice of S ~ Dm we have that LD(A(S)) ≥ ⅛** 
  + If H shatters some set C of size 2m, then we cannot learn H using m examples
  + If H shatters C, and we receive a sample containing only **half** instances of C, then the labels of these labels give us no information about the labels of the rest of the instances in C.
  + **If someone can explain every phenomenon, his explanations are worthless.**
* **Definition 6.5 (VC-dimension)** The VC-dimension of a hypothesis class H, denote by **VCdim(H)**, is the max size of a set C, a subset of domain X that can be shattered by H.
  + If H can shatter sets of arbitrarily large size we say that H has infinite VC-dimension
  + **Theorem 6.6:** Let H be a class of infinite VC dimension. Then H is not PAC learnable.
    - Since H has an infiniteVC dimension, for any training set size m, there exists a shattered set of size 2m, and the claim follows by **corollary 6.4.**
  + A finite VC dimension **guarantees** learnability.

**6.3 Examples**

* To show that VCdim(h) = d, need to show that,
  + 1. There exists a set C of size d that is shattered by H.
  + 2. Every set C of size d + 1 is not shattered by H.
* **Threshold Function**
  + Note that for an arbitrary set C = {c1}, H shatters C; therefore VCdim(H) ≥ 1
  + Note that for an arbitrary set C = {c1, c2} where c1 ≤ c2, H does not shatter C
  + Therefore, VCdim(H) = 1 → **finite** → learnable
* **Intervals**
  + Let H be the class of intervals over R. H = {ha, b : a b in R, a < b}, ha, b : R → {0, 1} is a function s.t. h(x) = 1 if a ≤ x ≤ b otherwise 0.
  + Consider when |C| = 3, C = {c1, c2, c3}, ascending orders. Then, (1, 0, 1) cannot be obtained by any h in H (if c1 and c3 are in range [a, b], then c2 must also be in that range)
  + Therefore, VCdim(H) = 2 → **finite** → learnable
* **Axis Aligned Rectangles**
  + For set C, a subset of R2 of 5 points. Then (1, 1, 1, 1, 0) cannot be obtained in the following case → VCdim(H) = 4 → **finite** → learnable

# 

* **Finite** **Classes**
  + Let H be a finite class.
  + For any set C we have |HC| ≤ |H|, and thus C cannot be shattered if |H| < 2|C|
  + This implies that VCdim(H) ≤ log2(|H|) ⇐ [|HC| ≤ |H| → 2|C| ≤ |H| → |C| ≤ log2(|H|)]
  + Note that VC dimension of a finite class H can be **significantly smaller than** log2(|H|).
* VC-dimension and the number of parameters
  + VC dimension does not necessarily equal the number of parameters defining the hypothesis class.

**6.4 The Fundamental Theorem of PAC Learning**

* **Theorem 6.7 (The Fundamental Theorem of Statistical Learning):** Let H be a hypothesis class of functions from a domain X to {0, 1}, and let the loss function be the 0 - 1 loss. Then the following are equivalent:
  + H has the uniform convergence property
  + Any ERM rule is successful agnostic PAC learner for H
  + H is agnostic PAC learnable
  + Any ERM rule is a successful PAC learner for H
  + H has a finite VC-dimension
* **Theorem 6.8 (The Fundamental THeorem of Statistical Learning - Quantitative Version):** Let H be a hypothesis class of functions from a domain X to {0, 1}, and let the loss function be the 0 - 1 loss. Assume that VCdim(H) = d < ∞. Then, there are absolute constants C1, C2 s.t.

# 

* **Sauer’s Lemma:** If VCdim(H) = d, then even though H might be infinite, when restricting it to a finite set C, a subset of domain X, its “effective size” |HC| is only O(|C|d). In other words, the size of HC grows polynomially rather than exponentially with |C|.
* **Definition 6.9 (Growth Function):**  Let H be a hypothesis class. Then the growth function of H, denoted by



* **Lemma 6.10 (Sauer-Shelah-Perles):** Let H be a hypothesis class with VC-dim(H) ≤ 
* **Theorem 6.11:** Let H be a class and tH be its growth function. Then, for every D and every delta in (0, 1), with probability of at least 1 - delta over the choice of S ~ Dm we have that,



# 

# 

# Chapter 11: Model Selection and Validation

Model Selection: Structural Risk Minimization

# 

# Chapter 7: Non-uniform Learnability

Previously, PAC learnability → finite VC dimension, which is limited. Here we consider a **more relaxed, weaker notion of learnability.** We will discuss how it is useful.

**7.1 Nonuniform Learnability**

* NL allows the sample size to be nonuniform w.r.t. the different hypotheses with which the learner is competing. We allow the sample size to be of the form mH(epsilon, delta, h); namely, it also depends on the h
* **Definition 7.1:** A hypothesis class H is nonuniformly learnable if there exists a learning algorithm, A and a function mHNULL: (0, 1)2 x H → N s.t. for every epsilon, delta in (0, 1) and for every h in H,
  + **If m ≥ mHNULL(epsilon, delta, h),**
  + **Then** for every distribution D, with probability of at least 1 - delta over the choice of S ~ Dm, it holds that



* Note that if a class is agnostic PAC learnable, then it is also nonuniformly learnable
* Characterizing Nonuniform Learnability
  + **Theorem 7.2:** A hypothesis class H of binary classifier is nonuniformly learnable **iff** it is a countable union of agnostic PAC learnable hypothesis classes.
  + **Theorem 7.3:** Let H be a hypothesis class that can be written as a countable union of hypothesis classes, H = Un in NHn, where each Hn enjoys the uniform convergence property. Then, H is nonuniformly learnable.
  + Example:
    - Consider a binary classification problem with the instance domain being X = R.
    - For every n in N, let Hn be the class of polynomial classifiers of degree n
    - Namely, Hn is the set of **all** classifiers of the form h(x) = sign(p(x)), where p: R → R is a polynomial of degree n.
    - Let H be the union of all Hn
    - Therefore, H is the class of all polynomial classifiers over R.
    - VCdim(H) = infinity
    - VCdim(Hn) = n + 1
    - Hence, H is not PAC learnable but is nonuniformly learnble

**7.2 Structural Risk Minimization**

* Specifying preferences over hypotheses within H.
* Assume that H is the union of Hn, where n in N
* Specify a weight function w: N → [0, 1], which assigns weight to each Hn
* **Higher weight reflects a stronger preference for the hypothesis class**
* The SRM rule follows a **bound minimization** approach
  + The goal of the paradigm is to find a hypothesis that MINs a certain upper bound on the true risk
  + **Theorem 7.4:** Let w: N → [0, 1] be a function s.t. ∑w(n) ≤ 1, for n in N. Let H be a hypothesis class that can be written as H = Un in NHn,
    - Where for each n, Hn satisfies the uniform convergence property with a sample complexity function mHnUC.
    - Let epsilon be defined as(7.1)
    - Then for every delta in (0, 1) and distribution D, with probability of at least 1 - delta over the choice of S ~ Dm, the following bound holds (simultaneously) for every n in N and h in Hn

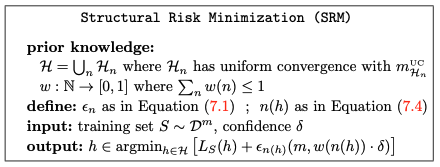


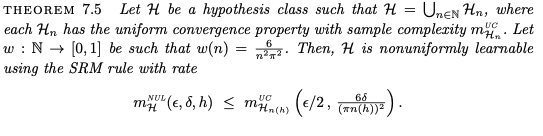
 (equation 7.4)

* Therefore, for every delta in (0, 1) and distribution D, with probability of at least 1 - delta, it holds that, (equation 7.3)

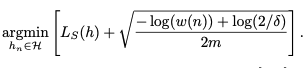


* SRM paradigm searches for h that MINs this bound, pseudocode:

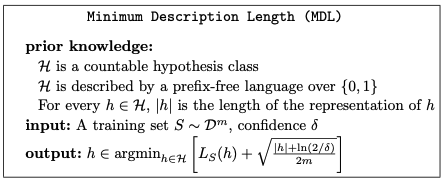


* 
* Theorem 7.5

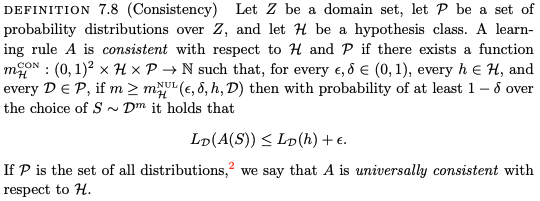
**7.3 Minimum Description Length and Occam’s Razor**

* Let H be a countable hypothesis class
* Write H as a countable union of **singleton classes**, H = Un in N{hn}.
* By Hoeffding;s inequality (Lemma 4.5), each singleton class has the uniform convergence property with rate mUC(epsilon, delta) = log(2 / delta) / (2 \* epsilon2).
* Therefore, the function epsilon given in **equation 7.1** becomes
  + - * 
* SRM rule becomes
  + - 
* A convenient way to define a weight function over H, which is derived from the length of descriptions given to hypotheses
  + Having a hypothesis class, one can wonder how we describe each hypothesis
  + We fix some description language (English, some programming languages)
  + **In any of these languages, a description consists of finite strings of symbols**
  + We shall formalize these notions
    - Let H be the hypothesis class we want to describe
    - Fix some finite set ∑ of symbols (characters) → AKA alphabet
    - For concreteness, let ∑ = {0, 1}
    - A string is a finite sequence of symbols from ∑, e.g. ∂ = (0, 1, 1, 1, 0)
    - |∂| = 5, length
    - A description language for H is a function d: H → ∑\*, mapping each member in H to a string d(h), where d(h) is called **the description of h, and its length is denoted by |h|.**
    - Suppose that the description language is **prefix-free**. Namely, for every h ≠ h’, d(h) is not a prefix of d(h’)
    - **Lemma 7.6 (Kraft Inequality):** If S ⊆ {0, 1}\* is a prefix-free set of strings, then



* In light of Kraft Inequality, any prefix-free description language of a hypothesis class, H, gives rise to a weighting function w over that hypothesis class - we will simply set w(h) = 1/(2|h|).
* **Theorem 7.7:** Let H be a hypothesis class and let d : H → {0, 1}\* be a prefix-free description language for H. Then
  + For every sample size m
  + For every confidence parameter delta > 0
  + For every probability distribution D,
  + With probability greater than 1 - delta over the choice of S ~ Dm we have
    - 
* Minimum Description Length (MDL)
  + 
* Let H be the class of all predictors that can be implemented using some programming language, say, C++. Let us represent each program using the binary string obtained by running the gzip command on the program (this yields a prefix-free description language over the alphabet {0, 1}). Then, |h| is simply the length (in bits) of the output of gzip when running on the C++ program corresponding to h.
* Occam’s Razor
  + Theorem 7.7 suggests that
    - If two hypotheses share the same empirical risk
    - The true risk of the one that **has shorter description** can be bounded by a lower value.
    - A short explanation (a hypothesis that has a short length) tends to be more valid than a long explanation.

**7.4 Other Notions of learnability - Consistency**

* The notion of learnability can be further relaxed by allowing the needed sample size to depend not only on epsilon, delta, and h, but also on the **underlying data-generating probability distribution D.**
* **Definition 7.8 (Consistency):**

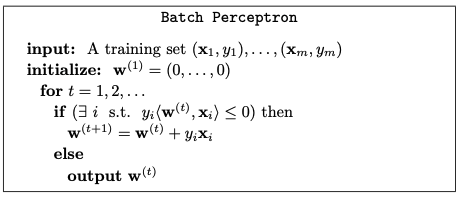
# Chapter 9: Linear Predictors

One of the most useful families of hypothesis class - linear predictors.

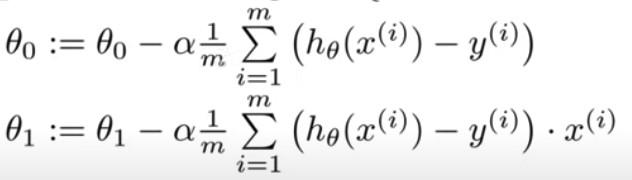
**9.0 Introduction**

* Linear predictors includes: halfspace, linear regression predictors, and logistic regression predictors, Perceptron algorithm, the least squares algorithm for linear regression.
* Define the class of affine functions as
  + 
  + 
  + Ld is a set of functions
    - Each function is parameterized by some weight **w** and some bias **b**
    - Takes a vector **x** as input
    - Returns a scalar <**w, x**> **+ b**
  + Each function in Rd can be rewritten as a **homogeneous linear function** in Rd+1 applied over the transformation that appends the constant 1 to each input vector ⇒ **omit the bias**

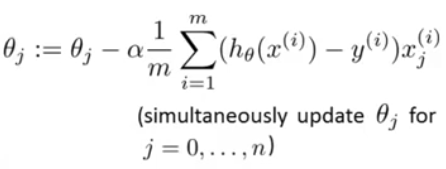
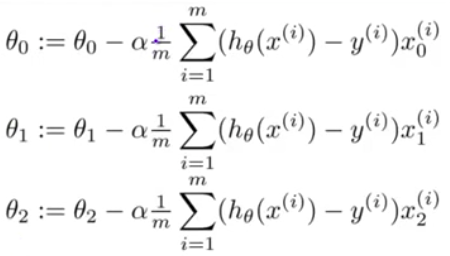
**9.1 Halfspaces**

* Designed for binary classification
* X = Rd, Y = {-1, +1}
* Definition
  + 
  + Consider when d = 2:
    - Each hypothesis forms a **hyperplane**
    - Hyperplane is perpendicular to the vector **w**
    - Label positive to all instances above that hyperplane
    - Label negative to all instances below that hyperplane
* VCdim(HSd) = d + 1
  + We can learn halfspaces using ERM
  + As long as the sample size is in Ω( (d + log(1/∂)) / epsilon )
* “Separable” VS “Non-separable”
  + Whether or not able to separate all the positive and negative examples with a hyperplane
  + For non-separable data, use **surrogate loss function**
* 9.1.1 Linear Programming for the class of halfspaces
  + MAX a linear function subject to / based on linear inequalities.
    - 
    - **w** is some vector in Rd, in which has variables we wish to determine
    - A is an m x d matrix
    - **v** is in Rm
    - **u** is in Rd
  + Show that the ERM problem for halfspaces in the realizable case can be expressed as a linear program.
    - Assume the homogenous case ⇒ 0 error on the training set.
    - Let S be the training set of size m
    - sign(<**w, xi**>) = yi ⇐⇒ yi<**w, xi**> > 0
* 9.1.2 Perceptron for Halfspaces
  + Iterative algorithm that constructs a sequence of vectors **w1, w2, w3, …**
  + Initially, set **w** to be all zeros
  + At iteration t, finds an example that is mislabeled by **w** ⇒ sign(<**wt, xi**>) ≠ yi
  + **Update wt, by adding to the instance xi scaled by the label yi**
  + **wt + 1 = wt + yixi**
  + ****
  + **Theorem 9.1:** Assume that (x1, y1), …, (xm, ym) is **separable**, let B = min{||**w**|| : for all i in [m], yi<**w, xi**> ≥ 1}, and let R = maxi||**xi||.** Then, the Perceptron stops after at most (RB)2 iterations.
    - Let **w\*** be the MIN norm s.t. yi<**w\*, xi**> ≥ 1 for all i
    - Need to show that after T iterations, the cosine of the angle between **w & w\*** is at least sqrt(T) / RB.
    - 
    - By Cauchy-Schwartz inequality, the left-hand-side of the equation above is at most 1
* **Note that**
  + **Perceptron is simple to implement**
  + **Is guaranteed to converge**
  + **The converge rate depends on the parameter B, which in some cases might be exponentially large in d**
* 9.1.3 The VC-dimension of Halfspace
  + To compute the VCdim of halfspace, we start with the homogenous case
  + **Theorem 9.2:** The VCdim of the class of homogenous halfspaces in Rd is d.
  + **Theorem 9.3:** The VCdim of the class of non-homogenous halfspaces in Rd is d + 1

**9.2 Linear Regression - One variable**

* Need to MIN the squared error
  + h(xi) = theta\_0 + theta\_1 \* xi, hypothesis
  + J(thetas) 1/2m ∑i = 1(h(xi) - yi)2, ASA cost function, square error function
    - From i = 0, to i = m, where m is the size of training sample
    - J()
* Simplified example
  + h(x) = theta1 \* x
    - A function of x
  + J(theta1) = 1/2m \* ∑(h(x) - y)2
    - A function of parameter theta1
* Algorithm
  + Start with some theta\_0, theta\_1, initialize to all 0’s
  + Keep changing theta\_0, theta\_1 to reduce J(theta\_0, theta\_1) until we hopefully end up at a minimum
  + Gradient descent algorithm
    - 
    - theta\_j := theta\_j - å \* derivative of theta\_j (J(theta\_0, …, theta\_i))
    - **Simultaneously update all thetas**
    - å, the learning rate, need to determine **its sign**
      * If å is too small, then gradient descent can be slow
      * If å is too large, then gradient descent can overshoot the MIN
    - Edge case: if theta is already at a local optima
      * theta will not change b/c the derivative is 0
    - Each step of gradient descent uses **all** the training examples

**9.2. Linear Regression - Multiple variable**

* + Multiple features
  + h(x) = theta\_0 \* x\_0 + theta\_1 \* x\_1 + … + theta\_n \* x\_n, where x\_0 = 1 (bias)
  + h(x) =ThetaT \* x
  + J(Theta) = 1/2m ∑ (h(x) - y)2, from i = 0 to i = m
  + Algorithm
    - Gradient descent, similar to above
    - 
    - 
    - **Feature Scaling**
      * Make sure features are on a similar scale
      * Example
        + x\_1 = size of a house (0 - 2,000 ft2)
        + x\_2 = number of bedrooms (1 - 5)
        + If we define x\_1, x\_2 as follows

x\_1 = size / 2000

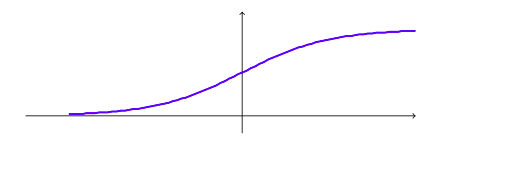
x\_2 = number of bedrooms / 5

Let x\_1 & x\_2 be in range of [-1, 1] (not necessary)

* + - * Mean normalization
        + Replace xi with xi - mu to make features have approximately zero mean, where mu is the average value of one feature
    - Making sure gradient descent is working correctly
      * Declare convergence if J decreases by less than 10-3 in one iteration
      * If gradient descent is not working, try smaller å
        + 0.001, 0.01, 0.1, 1, …

**9.3 Logistic Regression**

* Used for classification tasks.
* h(x) can be interpreted as the **probability that the label of x is 1**.
* Associate with logistic regression si the composition of a **sigmoid function** sig : R → [0, 1]
  + - * 

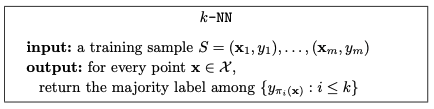


* The above is the logistic function (equation 9.9)
* The hypothesis class is therefore,
  + - * 
  + Note that when <w, x> is very large, sig(<w, x>) is close to 1
  + So when <w, x> is large, halfspace hypothesis and logistic hypothesis are very **similar**
  + Note that when <w, x> is very small, sig(<w, x>) is close to ½
  + Need to specify a loss function
    - 
    - Used to penalize hw based on the log of **1 + exp(-y<w, x>)**
    - 
  + Therefore, given a training set S = (x1, y1), …, (xm, ym), the ERM problem associated with logistic regression is, (equation 9.10)
    - 
* **Pros:** It is a **convex** function w.r.t weight w → can be solved efficiently using standard methods

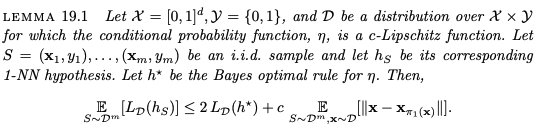
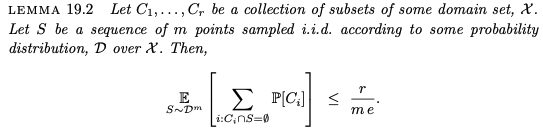
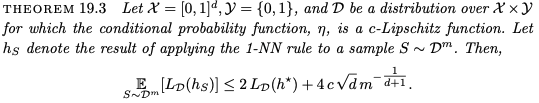
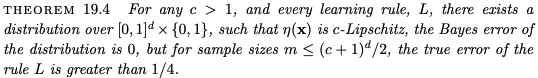
# Chapter 19: Nearest Neighbors

To memorize the training set and then to predict the label of any new instance on the basis of its closest neighbors in the training set.

**19.1 k Nearest Neighbors**

* Assume that we have a metric function p: X x X → R, returns the distance between any two elements of X (**Euclidean distance: || x - x’ ||**)
* Given a sequence of training examples S, reorder them by the distance to x
* k-NN rule for binary classification is defined as follows:
  + 
  + For regression problems, one can define the prediction to be the average target of the **k nearest neighbors**.
  + More generally, for some function : (X x Y)k → Y, the k-NN rule w.r.t. this function is
    -  (Equation 19.1)

**19.2 Analysis**

* How error decreases as a function of m and how it depends on the distribution.
* A Generalization Bound for 1-NN Rule
  + Analyze the true error of the 1-NN rule for binary classification with the 0-1 loss
  + If two vectors are close to each other, then their labels are likely to be the same
  + Lemma 19.1
  + 
  + The next step is to bound the expected distance between a random x and its closest element in S. Need the general probability lemma (19.2). **It bounds the probability weight of subsets that are not hit by a random sample, as a function of the size of that sample.**
  + 
  + **Theorem 19.3**
  + ****
* Curse of Dimensionality
  + The upper bound in Theorem 19.3 grows with c (the Lipschitz coefficient) and with d (the Euclidean dimension of the domain set X)
  + Necessary condition for the last term in 19.3, m ≥ (4c √(d / epsilon))d + 1
    - The training set size should grow **exponentially** with the dimension
  + **Theorem 19.4**
  + ****
    - For some distributions, this amount of examples is indeed necessary for learning with the NN rule.
    - The exponential dependence on the dimension is known as the curse of dimensionality
      * 1-NN might fail if the number of examples is smaller than Ω((c + 1)d)

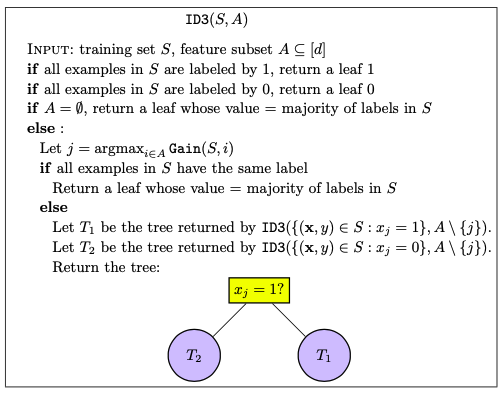
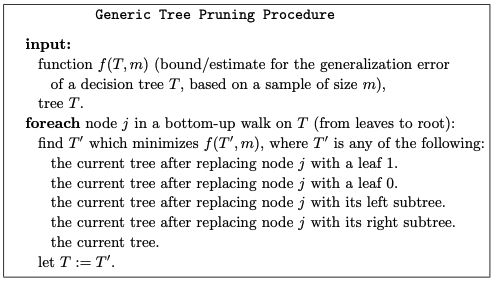
# Chapter 18: Decision Trees

A decision tree is a predictor, h : X → Y, that predicts the label associated with an instance x by traveling from a root node of a tree to a leaf.

**18.1 Sample Complexity**

* Popular splitting - based on thresholding **the value of a single feature**
* Move left or right child of a node on the basis of whether xi < some threshold theta
* If we allow decision trees of arbitrary size, we obtain a hypothesis class of infinite VCdim ⇒ overfitting ⇒ to avoid, use MDL mentioned in chapter 7.
* For simplicity
  + Assume X = {0, 1}d → each instance is a vector of d bits
  + Threshold → xi = 1 for some i in [d]
  + With such simplification, the hypothesis class is finite but still very large
    - Any classifier from {0, 1}d to {0, 1} can be represented as a decision tree of 2d leaf nodes and depth of d + 1 ⇒ VCdim = 2d
* Need to use MDL to make the tree smaller
  + A tree with n nodes will be described in n + 1 blocks, each of size log2(d + 3) bits
  + The first n blocks encode the nodes of the tree, in **preorder**
  + The last block marks the **end of the code**
  + Each block indicates whether the current node is
    - An internal node of the form **1**[xi = 1] for some i in [d]
    - A leaf whose value is 1
    - A leaf whose value is 0
    - End of the code
  + Therefore, there are d + 3 options ⇒ need log2(d + 3) bits to describe each block
  + The description length of a tree of n nodes will be (n + 1)(log2(d + 3))
  + By Theorem 7.7, we have that with probability of at least 1 - delta over a sample of size m, for every n and every decision tree h in H with n nodes it holds that, (equation 18.1)
    - 
    - Tradeoff:
      * Large tree → large n, small LS(h)
      * Small tree → small n, large LS(h)

**18.2 Decision Tree Algorithm**

* Equation 18.1 → we need to find a tree that MINs the right-hand side of the equation.
* A general framework for growing a decision tree
  + Start with the root node, assign it a label according to the majority vote among all labels over the training set
  + Perform a series of iterations
  + For each iteration
    - Examine the effect of splitting a single leaf
    - Define some “gain” that quantifies the improvement due to this split
    - For all possible splits, choose the one wi/ MAX gains and perform OR **not split**
* ID3 - Iterative Dichotomizer 3
  + 
  + Gain: given training set S and and index i, evaluates the gain of a split of the tree according to the i-th feature
* Implementations of Gain Measure
  + PS[F]: probability that an event holds w.r.t. the uniform distribution over S
  + **Train Error:** the decrease in training error
  + **Information Gain:** difference between the entropy of the label before and after the split
    - 
  + **Gini Index:** used by CART algorithm, C(a) = 2a(1 - a)
* Pruning
  + Problem of ID3: the returned tree is very large
  + Such tree may have low empirical risk but true risk tends to be high
  + Solutions
    - Limit the number of iterations
    - Prune the tree after it is build
  + Pruning usually is performed by a **bottom-up** walk on the tree
  + Each node might be replaced with one of its subtrees or with a leaf, based on some bound
    - 
* Threshold-based splitting rules for real-valued features (skip - not important)

**18.3 Random Forest**

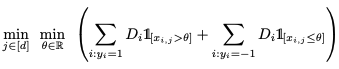
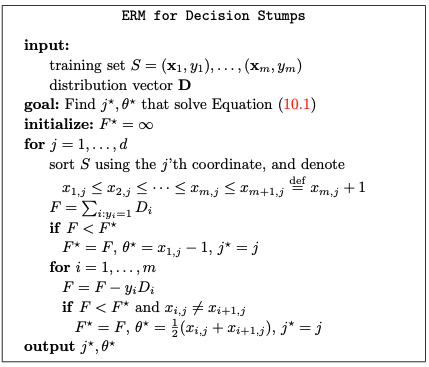
* To reduce the danger of overfitting, use random forest, a classifier consisting of a collection of decision trees, where each tree is constructed by applying an algorithm A on the training set S and an additional random vector, theta, where theta is sampled i.i.id from some distribution
* To specify a particular random forest, we need to define the algorithm A and the distribution over theta.
* To build one particular tree in the random forest
  + First, take a random subsample from S with replacements
  + Construct a sequence I1 , I2, ..., where each It is a subset of [ d ] of size k , which is generated by sampling uniformly at random elements from [ d ] . All these random variables form the vector theta
  + The algorithm A grows a decision tree (e.g., using ID3) based on the sample S0 , where at each splitting stage of the algorithm, the algorithm is restricted to choosing a feature that maximizes Gain from the set It

# Chapter 10: Boosting

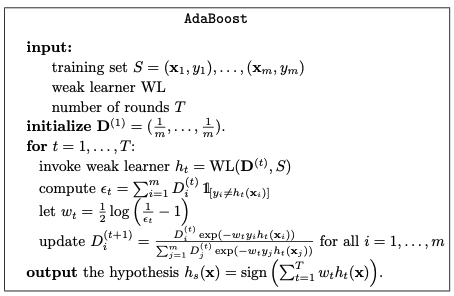
Boosting is an algorithmic paradigm (model). It uses a generalization of linear predictors to address two major issues:

* Bias-complexity tradeoff
* Computational complexity of learning

**10.1 Weak Learnability**

* Trade computational hardness with the requirement for accuracy
* **Definition 10.1 (r-weak-learnability):** 
  + A learning algorithm A, is a r-weak-learner for a class H if there exists a function mH: (0, 1) → N s.t. for every delta in (0, 1), for every distribution D over X, and for every labeling function ƒ: X → {-1, +1}. if the realizable assumption holds w.r.t. H, D, ƒ, then when running the learning algorithm on m ≥ mH(delta) i.i.d. examples generated by D and labeled by ƒ, the algorithm returns a hypothesis h s.t. with probability of at least 1 - delta, L(D, ƒ)(h) ≤ ½ - r.
  + A hypothesis class H is r-weak-learnable if there exists a r-weak-learner for that class
  + The PAC learning mentioned previously is AKA **strong learning**
    - Strong learnability implies the ability to find an arbitrarily good classifier (error rate at most € for an arbitrarily small € > 0)
    - Weak learnability only need to output a hypothesis whose error rate is at most ½ - r (slightly better than the random labeling)
  + **Weak learnability is also characterized by VCdim**
* Take a simple hypothesis class B, and apply ERM w.r.t. B as the weak learning algorithm. B needs to satisfy two requirements
  + 1. ERMB is efficiently implementable
  + 2. For every sample that is labeled by some hypothesis from H, any ERMB hypothesis will have an error of at most ½ - r.
  + Example 10.1 (Weak learning of 3-Piece classifiers using decision stumps)
    - Let X = R
    - Let H be the class of 3-Piece classifiers
      * 
      * An example of b = 1
      * 
    - Let B be the class of Decision Stumps; B = {x → sign(x - theta) • b : theta in R, b in {-1, +1}.}
* 10.1.1 Efficient Implementation of ERM for Decision Stump
  + How to find a decision stump that MINs LS(h)
  + Let D be a probability vector in Rm (∑ Di = 1, al non-negative)
  + The weak learner receives D and S and outputs a decision stump h: X → Y that MINs the risk w.r.t D.
    - *  if D = (1/m, …, 1/m) ⇒ LD(h) = LS(h)
  + Each decision stump is parameterized by an index j in [d], and a threshold theta. MIN LD(h) is, (equation 10.1)
    - 
    - Procedure:
      * Choose j in [d], and theta in Thetaj that MINs equation 10.1
      * For every j and theta, calculate a sum over m examples
      * So the runtime → O(dm2)
      * 
      * Runtime fasten to O(dm)

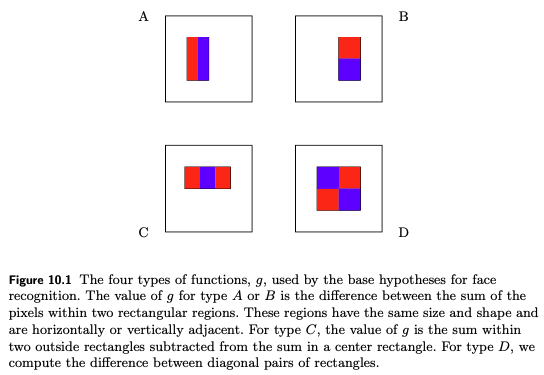
**10.2 AdaBoost**

* Adaptive Boosting - has access to a weak learner and finds a hypothesis with a low empirical risk
* Input: training set S = (x1, y1), …, (xm, ym), for each i, yi = ƒ(xi) for some labeling function ƒ
* Proceeds in a sequence of consecutive rounds
  + At round t, first define a distribution over the examples in S, denoted **D(t), ∑ Di(t) = 1**
  + Passes D(t) and the sample S to the weak learner (learner can construct i.i.d. examples)
  + The weak learner is assumed to return a weak hypothesis whose error,
    - * 
  + is at most ½ - r. (there is a probability of at most delta that the weak learner fails)
  + Then, assign a weight for ht where 
    - Weight is **inversely proportional** to the error of ht
  + At the end of the round, updates the distribution s.t. examples on which ht errors will get a higher probability mass while examples on which ht will get a lower probability mass.
    - → forcing learner to focus on **the problematic examples** in the next round
    - 
* **Theorem 10.2:** Let S be a training set and assume that at each iteration of AdaBoost, the weak learner returns a hypothesis for which €t ≤ ½ - r. Then, the training error of the output hypothesis of AdaBoost is at most,
  + - 
  + We assume that at each iteration of AdaBoost, the weak learner returns a hypothesis with weighted sample error of at most ½ - r.
  + According to the definition of a weak learner, it can fail with probability delta.
  + By union bound, the probability that the weak learner will not fail at all of the iteration is **at least** 1 - delta \* T.
  + This theorem tells us that as T grows, the empirical risk of hypothesis constructed by AdaBoost goes to 0

**10.3 Linear Combinations of Base Hypotheses**

* Given a base hypothesis class B (e.g. decision stump), the output of AdaBoost will be a member of the following class, (equation 10.4)
  + 
  + Lemma 10.3: Let B be a base class and let L(B, T) be as defined in equation 10.4. Assume that both T and VCdim(B) are at least 3, then,
    - 
    - Proof (page 139).

**10.4 AdaBoost for Face Recognition**

* The instance space is images, represented as matrices of gray level values of pixels (24 X 24 pi)
* Goal: learn a classifier, h: X → {-1, +1}, that given an image as input, should output whether the image is of a human face or not
* Each hypothesis in the base class is of the form h(x) = ƒ(g(x)), where ƒ is a decision stump hypothesis and g : R24, 24 → R is a function that maps an image to a **scalar**.
* Each g is parameterized by
  + 1. An axis aligned rectangle R. At most 244 axis aligned rectangles
  + 2. A type t, t in {A, B, C, D}. Each type corresponds to a mask
* 
* To calculate g
  + Stretch the mask t to fit the rectangle R and then calculate the sum of the pixels (sum of their grey values) that lie within the **red rectangle and subtract it from the sum of pixels in the blue rectangles**
* Implement the weak learner
  + Calculate all the possible outputs of g on each image
  + Apply the weak learner of decision stumps

# 

# Chapter X: TODO