## Summary – Linear Classification / Regression

- A linear classifier is given by:

$$\begin{aligned} h(X) &= 1 \text{ if } \sum_{i=1}^{d'} w_i \phi_i(X) + w_0 > 0 \\ &= 0 \text{ Otherwise} \end{aligned}$$

  where $\phi_i$ are fixed functions.
- For simplicity of notation, we use augumented feature vector and omit $\phi$.
- We take $h(X) = \text{sign}(W^T X)$ for simplicity of notation.
- Perceptron is a classical algorithm to learn such a classifier.

## Summary – Linear Classification / Regression

- For linear regression the objective is to learn a model:

$$\hat{y}(X) = \sum_{i=1}^{d} w_i x_i \ + \ w_0 = W^T X$$

  We could use $\phi_i(X)$ in place of $x_i$.

- The least squares criterion is to minimize

$$J(W) = \frac{1}{2} \sum_{i=1}^{n} (W^T X_i - y_i)^2$$

- The minimizer of $J$ is given by

$$W^* = (A^T A)^{-1} A^T Y$$

  Where $A$ is a matrix whose rows are $X_i^T$ and $Y$ is a vector whose components are $y_i$.

- This method can be used for learning linear classifiers also

## Summary – Linear Classification / Regression

- We can also minimize $J$ by iterative gradient descent.
- An incremental version of this gradient descent is the LMS algorithm.
- The LMS algorithm is:

$$W(k+1) = W(k) - \eta\,(X(k)^T W(k) - y(k))X(k)$$

  where $(X(k), y(k))$ is the (random) sample picked and $W(k)$ is the weight vector at iteration $k$.
- This is used in many adaptive signal processing problems.

## Summary – Linear Classification / Regression

- For any random variables $X, y$

$$f^*(X) \overset{\text{def}}{=} \arg\min_f \; E\left[(y - f(X))^2\right] = E[y|X]$$

- If $X, y$ are jointly Gaussian then this is a linear function.
- If $y \in \{0, 1\}$, then

$$E[y|X] = \mathsf{Prob}[y = 1|X] = q_1(X)$$

- In linear least squares method, we are essentially learning a parametric model for the posterior probability (A discriminative model)

## Summary – Linear Classification / Regression

- If we assume the model $y = W^T X + \xi$ where $\xi$ is a zero-mean Gaussian, then the MLE would be same as the linear least squares solution.
- The residual squared error is the MLE of noise variance.
- For linear regression, the Gaussian noise assumption is reasonable.

## Summary – Linear Classification / Regression

- For $y \in \{0,\ 1\}$, we can model conditional distribution of $y$ given $X$ as Bernoulli with parameter $\sigma(W^T X)$.
- The resulting algorithm to maximize log likelihood is logistic regression:

$$W(k+1) = W(k) + \eta \sum_{i=1}^{n} \left(y_i - \sigma(W^T X_i)\right) X_i$$

- Logistic regression is a better way to learn posterior probability for classification problems.
- Logistic regression is a prototypical example of learning a discriminative model.
- Logistic regression can also be foormulated as an IRLS algorithm.

## Summary – Linear Classification / Regression

- In regularized least squares (Ridge regression) we minimize

$$J(W) = \frac{1}{2}\sum_{i=1}^{n}(W^T X_i - y_i)^2 + \frac{\lambda}{2}W^T W$$

- The solution now is

$$W^* = (A^T A + \lambda I)^{-1} A^T Y$$

- In general, in regularization we take the objective to be sum of data error and model complexity.
- Regularization helps mitigate problems of overfitting.

## Summary – Linear Classification / Regression

- Under the model $y = W^T X + \xi$ where $\xi$ is a zero-mean Gaussian, the MLE is the linear least squares solution.
- Under the same model, the MAP estimate with a Gaussian prior is the $L_2$-regularized least squares solution.
- In general, different regularization terms amount to choosing different priors.

## Summary – Linear Classification / Regression

- Least squares soln is sensitive to outliers.
- Minimizing absolute value of error is more robust.
- We can formulate it as an IRLS algorithm.
- Huber loss is combination of squared and absolute value of error and is useful for robust regression.

## Summary – Linear Classification / Regression

- Fisher Linear Discriminant is another way to learn a linear classifier.
- Seeks to find a direction along which the projected data has best separation between the two classes.
- It can be obtained by solving a generalized eigen value problem

## Summary – Linear Classification / Regression

- Generalizing linear regression to handle vector-valued functions is straight-forward.
- We can formulate a $K$-class linear classifier by having $K$ functions $g_s(X) = W_s^T X + b_s$ and having

$$h(X) = C_j \;\text{ if }\; g_j(X) \geq g_s(X), \forall s$$

- Then learning multi-class claasifier through least squares method is same as learning vector-valued functions.

## Logistic Regression – multi-class case

- We can similarly generalize logistic regression also for multi-class case.
- Let us recall the main idea in logistic regression in the two class case.
- We approximate posterior probability as

$$q_1(X) = h(W^T X + w_0)$$

where $h$ is the logistic function

$$h(a) = \frac{1}{1 + \exp(-a)}$$

- The motivation for using the logistic function is

$$\begin{aligned} q_1(X) &= \frac{f_1(X)\, p_1}{f_0(X)\, p_0 \, + \, f_1(X)\, p_1} \\ &= \frac{1}{1+\exp(-\xi)} \quad \text{where} \end{aligned}$$

$$\xi = - \ln\left(\frac{f_0(X)\, p_0}{f_1(X)\, p_1}\right) \, = \, \ln\left(\frac{f_1(X)\, p_1}{f_0(X)\, p_0}\right)$$

- We now use the same Bayes rule to find a convenient model for posterior probabilities in the multiclass case.

- In the multi-class case, Bayes rule gives

$$q_j(X) = \frac{f_j(X)p_j}{\sum_s f_s(X)p_s} = \frac{\exp(a_j)}{\sum_s \exp(a_s)}$$

  where $a_s = \ln(f_s(X)p_s)$.
- The idea is that we approximate $a_s = W_s^T X + w_{s0}$.
- The above function is a good candidate for modeling posterior probabilities in the multi-class case.

- In the two class case we want to know which of $f_1(X)p_1$ and $f_0(X)p_0$ is greater.
- This can be done by looking at sign of $\ln\left(\frac{f_1(X)\,p_1}{f_0(X)\,p_0}\right)$.
- 'sign' is a discontinuous function and the logistic function is a kind of continuous analog for this.
- In the multiclass case, we need to find the maximum of $f_i(X)p_i$, $i = 1, \cdots, K$.
- So, we need a smooth function to approximate the maximum computation.

- Define a function $g : \Re^K \to \Re^K$, with $g(a) = [g_1(a) \ \cdots \ g_K(a)]^T$ and for $j = 1, \cdots, K$,

$$g_j(a) = \frac{\exp(a_j)}{\sum_s \exp(a_s)}, \ a = (a_1, \cdots, a_K)^T \in \Re^K.$$

- This is known as the softmax function.
- Essentially if $a_j$ is the maximum of the components of $a$ then $g_j(a)$ would be closer to one and all other components of $g$ would be closer to zero.
- We note, for later use, that

$$\frac{\partial g_k}{\partial a_j} = g_k(a)(\delta_{kj} - g_j(a))$$

where $\delta_{kj} = 0$ if $k \neq j$ and $\delta_{kj} = 1$ if $k = j$.

- We now take, for each $s$, $a_s = W_s^T X + w_{s0}$ and learn all $W_s$ and $w_{s0}$.
- Using augumented feature vector, we can write $a_s = W_s^T X$. Let $W$ be a matrix with columns $W_s$.
- After learning, $W$, given a new $X$, we calculate $g(W^T X)$ and then put $X$ in class $C_j$ if the $j^{th}$ component of $g(W^T X)$ is the highest.

- In the 2-class logistic regression, we used the logistic function to model posterior probability.
- In the multi-class case we want to use the softmax function for modeling the conditional distribution of $y$ given $X$.
- Let us now write

$$g_j(W, X) = \frac{\exp(W_j^T X)}{\sum_s \exp(W_s^T X)}$$

- Let us represent the class label as a one-hot vector:

$$y = (y^1, \cdots, y^K)^T; \;\; y^j \in \{0, \; 1\}; \;\; \sum_j y^j = 1$$

- $y$ takes only $K$ different values and let us represent them as $e_1, \cdots, e_K$.
- We take (as our probability model) $P[y = e_j \mid X] = g_j(W, X)$.

- The probability model for the conditional distribution now is

$$f(y \mid X, W) = \prod_{i=1}^{K} (g_i(W, X))^{y^i}$$

- Let the data be $\mathcal{D} = \{(X_1, y_1), \cdots, (X_n, y_n)\}$.
- The likelihood now is

$$L(W_1, \cdots, W_K | \mathcal{D}) = \prod_{i=1}^{n} f(y_i | X_i, W) = \prod_{i=1}^{n} \prod_{j=1}^{K} (g_j(W, X_i))^{y_i^j}$$

- The log likelihood is

$$l(W_1, \cdots, W_K \mid \mathcal{D}) = \sum_{i=1}^{n} \sum_{j=1}^{K} y_i^j \ln (g_j(W, X_i))$$

- We need to maximize this to learn the $W$.

- The log likelihood is

$$l(W_1, \cdots, W_K \mid \mathcal{D}) = \sum_{i=1}^{n} \sum_{j=1}^{K} y_i^j \ln\left(g_j(W, X_i)\right)$$

- By differentiating this and using earlier formula for derivative of $g_j$, we can show that

$$\nabla_{W_j} l(W_1, \cdots, W_K \mid \mathcal{D}) = \sum_{i=1}^{n} (y_i^j - g_j(W, X_i)) X_i$$

- Hence an iterative algorithm for ML estimate of $W_j$, $j = 1, \cdots, K$, is

$$W_j(k+1) = W_j(k) + \eta \sum_{i=1}^{n} (y_i^j - g_j(W, X_i)) X_i$$

- This is the multi-class logistic regression.

- We can generalize Fisher linear discriminant also to multi-class case.
- In the 2-class case we are interested in finding a direction or a one-dimensional subspace onto which we project the data.
- In the K-class case, we want to find a $(K-1)$-dimensional subspace onto which we project the data.
- The idea is to find a subspace where in the projected data, the means of the two classes have maximum separation relative to the variances.

## Learning and generalization

- The problem of designing a classifier is essentially one of learning from examples.
- Given training data, we want to find an appropriate classifier.
- It amounts to searching over a family of classifiers to find one that minimizes 'error' over training set.
- For example, in least squares approach we are searching over the family of linear classifiers for minimizing square of error.

- As we discussed earlier, performance on training set is not the ultimate objective.
- We would like the learnt classifier to perform well on new data.
- This is the issue of generalization. Does the learnt classifier generalize well?

- In practice one assesses the generalization of the learnt classifier by looking at the error on a separate set of labelled data called test set.
- Since the test set would not be used in training, error on that data could be a good measure of the performance of the learnt classifier.
- But here we are more interested in formalizing the notion of generalization error.
- We look at the specific issues of practice later on. Currently our focus would be on theoretical analysis of how to say whether a learning algorithm would generalize well.

- Any learning algorithm takes training data as the input and outputs a specific classifier/function.
- For this, it searches over some chosen family of functions to find one that optimizes a chosen criterion function.

$$\{(X_i, y_i)\} \rightarrow \boxed{\begin{array}{l}\text{Learning Algorithm} \\ \text{(searching over } \mathcal{F}\text{)}\end{array}} \rightarrow f \in \mathcal{F}$$

- The question is: how can we formalize correctness of learning?
- There are many ways of addressing this issue (MDL, VC-theory etc).

- We discuss (at an elementary level) a specific statistical approach to address the issue of correctness of learning.
- We begin with a simple formalism where there is no ‘noise’ and the goal of learning is well-defined.

A Learning problem is defined by giving:

(i) $\mathcal{X}$ – input space;   often $\Re^d$ (*feature space*)

(ii) $\mathcal{Y} = \{0,\ 1\}$ – output space (*set of class labels*)

(iii) $\mathcal{C} \subset 2^{\mathcal{X}}$ – concept space (*family of classifiers*)
Each $C \in \mathcal{C}$ is a subset of $\mathcal{X}$.
It can also be viewed as a function $C : \mathcal{X} \rightarrow \{0,\ 1\}$, with $C(X) = 1$ iff $X \in C$.

(iv) $S = \{(X_i, y_i),\ i = 1, \cdots, n\}$ – the set of examples,
$X_i$ are drawn *iid* according to some distribution $P_x$ on $\mathcal{X}$
$y_i = C^*(X_i)$ for some $C^* \in \mathcal{C}$.
$C^*$ is called target concept.

- We are considering a 2-class case.
- Hence any classifier is a function $C : \mathcal{X} \rightarrow \{0,\ 1\}$.
- Thus, $\mathcal{C}$ is a family of classifiers.
- We call this concept space because we can say the system is learning a ‘concept’ from examples.
- The learning algorithm knows $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{C}$; but does not know $C^*$.
- It needs to learn the target concept from examples.

## Some Comments

- We do not know the distribution $P_x$.
- We are trying to teach a concept through examples that come from an arbitrary distribution.
- However, taking that the examples are *iid* ensures we get 'representative' examples.
- Since we have taken $y_i = C^*(X_i), \ \forall i$, there is no 'noise'.
- Also assuming that $C^* \in \mathcal{C}$ means that ideally we can learn the target concept. (Realizability assumption)

- We could take $\mathcal{C} = 2^{\mathcal{X}}$.
- This means we are searching over the family of all possible (2-class) classifiers.
- This may not be viable.
- We can choose a particular $\mathcal{C}$ based on either some knowledge we have about the problem or because of the kind of learning algorithm we have.
- For example we can take $\mathcal{C}$ to be all half-spaces – the family of all linear classifiers.

## Probably Approximately Correct Learning

- Let us now try to define the goal of learning.
- Note that each $C \in \mathcal{C}$ can be viewed either as a subset of $\mathcal{X}$ or a binary valued function on $\mathcal{X}$.
- Let $C_n$ denote the concept or classifier output by the learning algorithm after it processes $n$ *iid* examples.
- For correctness of the learning algorithm we want $C_n$ to be 'close' to $C^*$ as $n$ becomes large.
- The closeness of $C_n$ to $C^*$ is in terms of classifying samples drawn from $\mathcal{X}$ according to $P_x$.

- We define **error** of $C_n$ by

$$\begin{aligned} \text{err}(C_n) &= P_x(C_n \Delta C^*) = P_x\left((\bar{C}_n \cap C^*) \cap (C_n \cap \bar{C}^*)\right) \\ &= \text{Prob}[\{X \in \mathcal{X} : C_n(X) \neq C^*(X)\}] \end{aligned}$$

- The $\text{err}(C_n)$ is the probability that on a random sample, drawn according to $P_x$, the classification of $C_n$ and $C^*$ differ.

- Essentially, we want err$(C_n)$ to become zero as $n \to \infty$.
- However, err$(C_n)$ is a random variable because $C_n$ is a function of the random samples $X_1, \cdots, X_n$.
- We take the above convergence to be in probability.

# PAC learning

- We say a learning algorithm **Probably Approximately Correctly** (PAC) learns a concept class $\mathcal{C}$ if given any $\epsilon, \delta > 0$, $\exists N < \infty$ such that

$$\text{Prob}[\text{err}(C_n) > \epsilon] < \delta$$

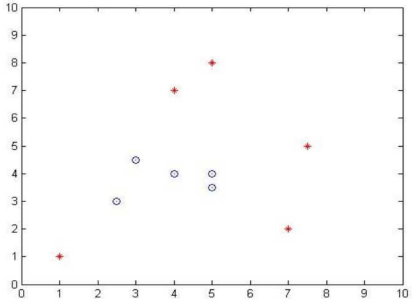
  for all $n > N$ and for any distribution $P_x$ and any $C^*$.
- The probability above is with respect to the distribution of $n$-tuples of *iid* samples drawn according to $P_x$ on $\mathcal{X}$.
- The $P_x$ is arbitrary. But, for testing and training the distribution is same – 'fair' to the algorithm.

- An algorithm PAC learns $\mathcal{C}$ if
$$\text{Prob}[\text{err}(C_n) > \epsilon] < \delta$$
for sufficiently large $n$ and any $P_x$.
- If err$(C_n) \leq \epsilon$, then $C_n$ is 'approximately correct'.
- So, what the above says is that the classifier output by the algorithm after seeing $n$ random examples, $C_n$, is **approximately correct with a high probability**.
- The $\epsilon$ and $\delta$ are called the accuracy and confidence parameters respectively.

## A Simple Example

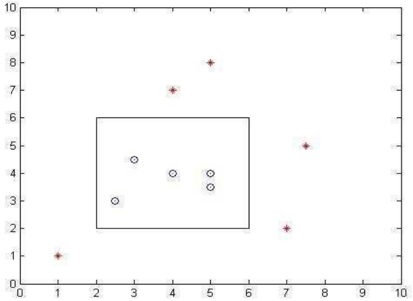
- Suppose we want to learn the concept of medium-build persons based on features of height and weight.
- Here $\mathcal{X} = \Re^2$.
- We would be given examples (with no errors!) drawn from some arbitrary distribution.

- The examples for learning our concept could be the following.



- What could be $\mathcal{C}$ in this example?
- We can take $\mathcal{C}$ to be all subsets of $\Re^2$.
- Or we can use some problem-based intuition and choose $\mathcal{C}$ to be all axis-parallel rectangles.
- In this case, assuming $C^* \in \mathcal{C}$ means that the 'god-given' classifier is also an axis-parallel rectangle.

- The examples along with $C^*$ now could be



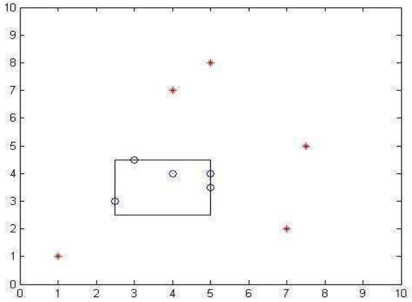
- Now consider a very general class of algorithms.
- The strategy of the learning algorithm is as follows.
- The algorithm outputs a classifier which correctly classifies all examples.
- This is called a consistent classifier. Since $C^* \in \mathcal{C}$, we know there is always at least one consistent classifier.
- If there is more than one $C \in \mathcal{C}$ that is **consistent** with all examples, we output the 'smallest' such $C$.

- For finite sets, smallest is in terms of number of poins; for other sets it is in terms of the ‘areas’ of the sets.
- We take $\mathcal{C}_1$ to be the set of all axis-parallel rectangles.
- We take $\mathcal{C}_2$ to be $2^{\mathcal{X}}$; that is, set of all possible classifiers.
- We will look at what we can say about the output of the algorithm in the two cases.

- We assume that $C^*$ is an axis-parallel rectangle.
- Then $C^*$ belongs to both $\mathcal{C}_1$ and $\mathcal{C}_2$.
- All our examples are classified according to $C^*$.
- We assume that the boundary of the rectangle is part of $C^*$.

- First consider $\mathcal{C}_1$.
- The smallest $C \in \mathcal{C}$ consistent with all examples would be the smallest axis-parallel rectangle enclosing all the positive examples seen so far.

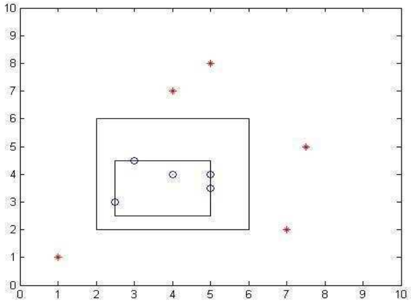
- The concept output by the algorithm (when using $\mathcal{C}_1$) would be the following.



- Thus, under the strategy of our learning algorithm, for all $n$, the $C_n$ would always be inside the $C^*$.
- Now let us show that this is a PAC learning algorithm.

- Whenever any example is classified as positive by $C_n$ it would also be classified positive by $C^*$.
- Hence the points of $\mathcal{X}$ where $C_n$ makes errors is the annular region.

- Since $C_n$ is also an axis-parallel rectangle which is inside $C^*$, the $C_n \Delta C^*$ would be the annular region between the two rectangles.



- Hence, $\text{err}(C_n)$ is the $P_x$-probability of this annular region.
- Note that we are not really bothered about the area of this annular region; we are only interested in the probability mass of this region under $P_x$.

- Now, given an $\epsilon > 0$, we have to bound the probability that err$(C_n) > \epsilon$.
- The error is greater than $\epsilon$ only if the probability mass (under $P_x$) of the annular region is greater than $\epsilon$.
- When does this event $[$err$(C_n) > \epsilon]$ occur?
- Only when none of the examples seen happen to be in the annular region.
- Why? – Otherwise, the rectangle learnt by our algorithm would have been closer to $C^*$.

- Hence the probability of the event $[\text{err}(C_n) > \epsilon]$ is same as the probability that when $n$ *iid* examples are drawn accoding to $P_x$ none of them came from a subset of $\mathcal{X}$ that has $P_x$-probability at least $\epsilon$.
- That is, all examples came from a subset of probability at most $(1 - \epsilon)$.
- The probability of this happenning is at most $(1 - \epsilon)^n$.

- Hence we have

$$\text{Prob}[\text{err}(C_n) > \epsilon] \leq (1-\epsilon)^n$$

- Let $N$ be such that $(1-\epsilon)^n < \delta$, for all $n > N$.
- The required $N$ is $N \geq \frac{\ln(\delta)}{\ln(1-\epsilon)}$ (bound on number of examples).
- For this $N$, we have

$$\text{Prob}[\text{err}(C_n) > \epsilon] \leq \delta, \ \forall n \geq N$$

showing the algorithm PAC learns the concept class.

- Now let us consider the same algorithm with concept class $\mathcal{C}_2 = 2^{\mathcal{X}}$.
- Here we are searching over all possible 2-class classifiers.
- So, intuitively, we do not expect the algorithm to be able to learn anything.
- There is too much 'flexibility' in the bag of classifiers over which we are searching.
- Let us show this formally.

- What would be $C_n$ now?
- After seeing $n$ examples, the smallest set in $\mathcal{C}_2$ that is consistent with all examples is the set consisting of all the positive examples seen so far!!
- Now the algorithm simply remembers all the positive examples seen.
- This happened because every possible finite subset of $\mathcal{X}$ is in our concept class.

- So, now, $C_n \Delta C^*$ would be the axis parallel rectangle $C^*$ minus some finite number of points from it.
- So, under any continuous $P_x$, err$(C_n) = P_x(C_n \Delta C^*) = P_x(C^*)$.
- Hence, for $\epsilon < P_x(C^*)$, Prob[err$(C_n) > \epsilon] = 1$ for all $n$.
- Thus, the algorithm can not PAC learn with $\mathcal{C}_2$.

- This example clearly illustrates the difficulty of learning from examples if the bag of classifiers being considered is too ‘large’.
- The largeness is not interms of number of elements in our concept class.
- Both $\mathcal{C}_1$ and $\mathcal{C}_2$ contain uncountably infinite number of classifiers.
- We would later define an appropriate quantity to quantify the sense in which one concept class can be said to be bigger than (or more complex than) another.

- At this point we can still see how $\mathcal{C}_1$ is smaller than $\mathcal{C}_2$.
- Since every axis parallel rectangle can be specified by four quantities, this class can be parameterized by four parameters.
- However, there is no such finite parameterization for $\mathcal{C}_2 = 2^{\Re^2}$.
- Also, the strategy of our algorithm can be coded efficiently in case of $\mathcal{C}_1$.

- The concept of PAC learnability is interesting.
- It allows one to properly define what is correctness of learning and allows us to ask questions like whether a given algorithm learns correctly.
- As we have seen in our example, we can also bound the number of examples needed to learn to a given level of accuracy and confidence.
- Thus we can appreciate relative complexities of different learning problems.

- However, PAC learnability deals with ideal learning situations.
- We assume there is a ('god-given') $C^*$ and that it is in our $\mathcal{C}$.
- Also, we assume that examples are noise free and are perfectly classified.
- Next we consider an extension of this framework that is relevant for realistic learning scenarios.

In our new framework we are given

- $\mathcal{X}$ – input space; ( as earlier, *Feature space*)
- $\mathcal{Y}$ – Output space (as earlier, *Set of class labels*)
- $\mathcal{H}$ – hypothesis space (*family of classifiers*)

  Each $h \in \mathcal{H}$ is a function: $h : \mathcal{X} \to \mathcal{A}$
  where $\mathcal{A}$ is called *action space*.
- Training data: $\{(X_i, y_i),\ i = 1, \cdots, n\}$
  drawn *iid* according to some distribution $P_{xy}$ on $\mathcal{X} \times \mathcal{Y}$.

## Some Comments

- We have replaced $\mathcal{C}$ with $\mathcal{H}$.
- If we take $\mathcal{A} = \mathcal{Y}$ then it is same as earlier.
- But the freedom in choosing $\mathcal{A}$ allows for taking care of many situations.
- For example, when $\mathcal{Y} = \{0,\ 1\}$, we can take $\mathcal{A} = [0,\ 1]$ (e.g., logistic regression).

- Now we draw examples from $\mathcal{X} \times \mathcal{Y}$ according to $P_{xy}$. This allows for 'noise' in the training data.
- For example, when class conditional densities overlap, same $X$ can come from different classses with different probabilities.
- We can always factorize $P_{xy} = P_x P_{y|x}$. If $P_{y|x}$ is a degenerate distribution then it will be same as earlier – we draw iid samples from $\mathcal{X}$ and each point is essentially classified by the target classifier.
- However, having examples drawn from $\mathcal{X} \times \mathcal{Y}$ using a distribution, allows for many more scenarios.

- As before, the learning machine outputs a hypothesis, $h_n \in \mathcal{H}$, given the training data consisting of $n$ examples.
- However, now there is no notion of a target concept/hypothesis.
- There may be no $h \in \mathcal{H}$ which is consistent with all examples.
- Hence we use the idea of loss functions to define the goal of learning.

## Loss function

- Loss function: $L : \mathcal{Y} \times \mathcal{A} \to \Re^+$.
- The idea is that $L(y, h(X))$ is the 'loss' suffered by $h \in \mathcal{H}$ on a (random) sample $(X, y) \in \mathcal{X} \times \mathcal{Y}$.
- More generally we can let loss depend on $X$ also explicitly and can write $L(X, y, h(X))$ for loss function.
- By convention we assume that the loss function is non-negative.
- Now we can look for hypotheses that have low average loss over samples drawn accordding to $P_{xy}$.

## Risk Function

- Define the **risk** function, $R : \mathcal{H} \to \Re^+$, by

$$R(h) = E[L(y, h(X))] = \int L(y, h(X))\, dP_{xy}$$

- Risk is expectation of loss where expectation is with respect to $P_{xy}$.
- We want to find $h$ with low risk.

## Risk Minimization

- Let
$$h^* = \arg \min_{h \in \mathcal{H}} R(h)$$
- We define the goal of learning as finding $h^*$, the global minimizer of risk.
- Risk minimization is a very general strategy adopted by most machine learning algorithms.
- Sometimes called 'Agnostic Learning'
- Note that we may not have any knowledge of $P_{xy}$.
- Minimization of $R(\cdot)$ directly is not feasible.

## Empirical Risk function

- Define the **empirical risk function**, $\hat{R}_n : \mathcal{H} \to \Re^+$, by

$$\hat{R}_n(h) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, h(X_i))$$

  This is the sample mean estimator of risk obtained from $n$ *iid* samples.
- Let $\hat{h}_n^*$ be the global minimizer of empirical risk, $\hat{R}_n$.

$$\hat{h}_n^* = \arg \min_{h \in \mathcal{H}} \hat{R}_n(h)$$

## Empirical Risk Minimization

- Given any $h$ we can calculate $\hat{R}_n(h)$.
- Hence, we can (in principle) find $\hat{h}_n^*$ by optimization methods.
- Approximating $h^*$ by $\hat{h}_n^*$ is the basic idea of empirical risk minimization strategy.
- Used in most ML algorithms.

- Is $\hat{h}_n^*$ a good approximator of $h^*$, the minimizer of true risk (for large $n$)?
- This is the question of **consistency of empirical risk minimization**.
- Thus, we can say a learning problem has two parts.
    - The optimization part: find $\hat{h}_n^*$, the minimizer of $\hat{R}_n$.
    - The statistical part: Is $\hat{h}_n^*$ a good approximator of $h^*$.

- The optimization part depends on the loss function.
- Note that the loss function is chosen by us; it is part of the specification of the learning problem.
- The loss function is intended to capture how we would like to evaluate performance of the classifier.
- We look at a few loss functions in the 2-class case.

## The 0–1 loss function

- Let $\mathcal{Y} = \{0,\ 1\}$ and $\mathcal{A} = \mathcal{Y}$.
- Now, the 0–1 loss function is defined by

$$L(y, h(X)) = I_{[y \neq h(X)]}$$

where $I_{[A]}$ denotes indicator of event $A$.

- The 0-1 loss function is

$$L(y, h(X)) = I_{[y \neq h(X)]}$$

- Risk is expectation of loss.
- Hence, $R(h) = \text{Prob}[y \neq h(X)]$;
  the risk is probability of misclassification.
- So, $h^*$ minimizes probability of misclassification.
  (Bayes classifier)

- Here we assumed that the learning algorithm searches over a class of binary-valued functions on $\mathcal{X}$.
- We can extend this to, e.g., discriminant function learning.
- We take $\mathcal{A} = \Re$ (now $h(X)$ is a discriminant function).
- We can define the 0-1 loss now as

$$L(y, h(X)) = I_{[y \neq sgn(h(X))]}$$

- Having any fixed misclassification costs is essentially same as 0–1 loss.
- Even if we take $\mathcal{A} = \Re$, the 0–1 loss compares only sign of $h(x)$ with $y$. The magnitude of $h(x)$ has no effect on the loss.
- Here, we can not trade 'good' performance on some data with 'bad' performance on others.
- This makes 0–1 loss function more robust to noise in classification labels.

- While 0–1 loss is an intuitively appealing performance measure, minimizing empirical risk here is hard.
- The 0–1 loss function is non-differentiable which makes the empirical risk function also non-differentiable.
- Hence many other loss functions are often used in Machine Learning.

## Squared error loss

- The squared error loss function is defined by

$$L(y, h(X)) = (y - h(X))^2$$

- As is easy to see, the linear least squares method is empirical risk minimization with squared error loss function.
- Here we can take $\mathcal{Y}$ as $\{0,\ 1\}$ or $\{+1,\ -1\}$. We take $\mathcal{A} = \Re$ so that each $h$ is a discriminant function.
- As we know, we can use this for regression problems also and then we take $\mathcal{Y} = \Re$.

- Another interesting scenario here is to take $\mathcal{Y} = \{0,\ 1\}$ and $\mathcal{A} = [0,\ 1]$.
- Then each $h$ can be interpreted as a posterior probability (of class-1) function.
- As we know, the minimizer of expectation of squared error loss (the risk here) is the posterior probability function.
- So, risk minimization would now look for a function in $\mathcal{H}$ that is a good approximation for the posterior probability function.

- The empirical risk minimization under squared error loss is a convex optimization problem for linear models (when $h$ is linear in its parameters).
- The squared error loss is extensively used in many learning algorithms.

## soft margin loss or hinge loss

- Take $\mathcal{Y} = \{+1, \ -1\}$ and $\mathcal{A} = \Re$. The loss function is given by

$$L(y, h(X)) = \max(0, \ 1 - yh(X))$$

- Here, if $yh(X) > 0$ then classification is correct and if $yh(X) \geq 1$, loss is zero.
- This also results in convex optimization for empirical risk minimization.

## Margin Losses

- All three losses we mentioned can be written as function of $yh(X)$ by taking $\mathcal{Y} = \{-1,\ +1\}$.
- The 0–1 loss :

$$L(y, h(X)) = \mathsf{sign}(-yh(X))$$

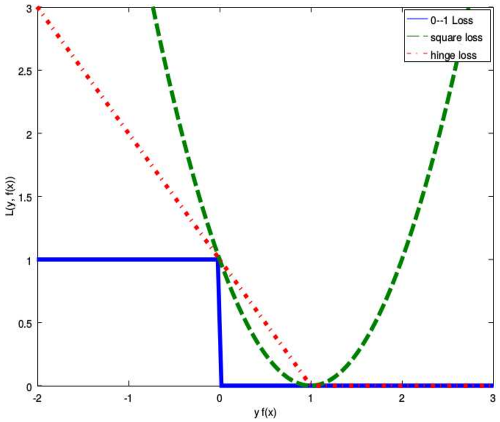
- The squared error loss:

$$L(y,\ h(X)) = (y - h(X))^2 = (1 - yh(X))^2$$

- The hinge loss (used in SVM):

$$L(y, h(X)) = \max(0,\ 1 - yh(X))$$

## Plot of 2-class loss functions



- We can think of the other losses as convex approximations of 0–1 loss.

- As we saw, there are many different loss functions one can think of.
- Many of them also make the empirical risk minimization problem efficiently solvable.
- We consider many such algorithms in this course.
- Now, let us get back to the statistical question that we started with.

## Consistency of Empirical Risk Minimization

- Our objective is to find $h^*$, minimizer of risk $R(\cdot)$.
- We minimize the empirical risk, $\hat{R}_n$, and thus find $\hat{h}_n^*$.
- We want $h^*$ and $\hat{h}_n^*$ to be 'close'.
- More precisely we are interested in the question: Does

$$\forall \delta > 0,\ \text{Prob}[|R(\hat{h}_n^*) - R(h^*)| > \delta] \to 0,\ \text{as } n \to \infty?$$

- Same as asking whether $R(\hat{h}_n^*)$ converges in probability to $R(h^*)$
- This is the question we will address in the next class.