CS534 Machine Learning

Spring 2011

Course Information

- Instructor:

  Dr. Xiaoli Fern

  Kec 3073, xfern@eecs.oregonstate.edu

- Office hour

  MWF before class 11-12 or by apppointment

- Class Web Page

  web.engr.orst.edu/~xfern/classes/cs534

Course materials

- No text book required, slides and reading materials will be provided on course webpage
- There are a few recommended books that are good references
  - Pattern recognition and machine learning by Chris Bishop (Bishop) – highly recommended
  - Machine learning by Tom Mitchell (TM)

Course Overview

- Covers a wide range of machine learning techniques – from basic to state-of-the-art

  Perceptron, Logistic regression, LDA, Decision tree, Neural net, Naïve Bayes, KNN, SVM, Hidden Markov Models, EM, K-means, Mixture of Gaussian, Dimension reductions

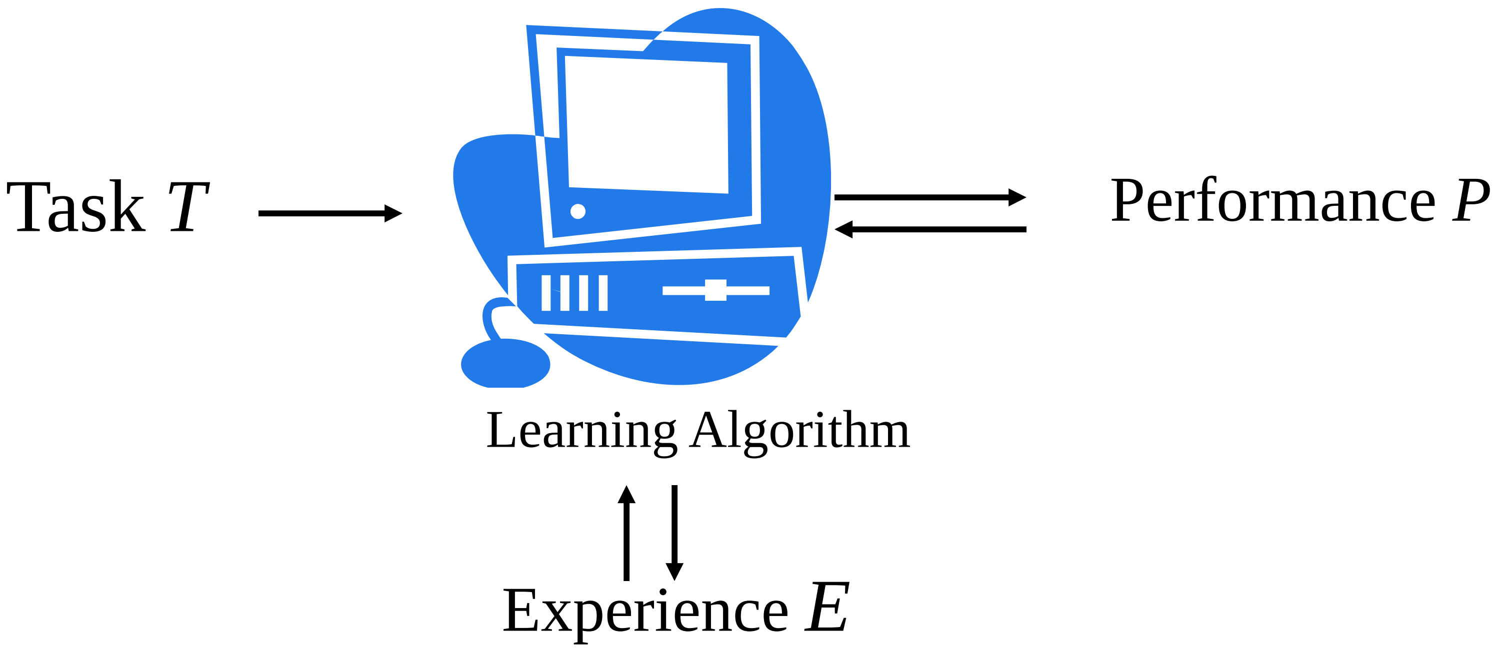
- You will learn about algorithms, related theories and applications

Prerequisites

- Multivariable Calculus and linear algebra
  - Some basic review slides on class webpage
  - Useful video lectures

  ocw.mit.edu/OcwWeb/Mathematics/18-06Spring-2005/VideoLectures/index.htm
  ocw.mit.edu/OcwWeb/Mathematics/18-02Fall 2007/VideoLectures/index.htm
- Basic probability theory and statistics concepts: Distributions, Densities, Expectation, Variance, parameter estimation …
- Knowledge of basic CS concepts such as data structure, search strategies, complexity

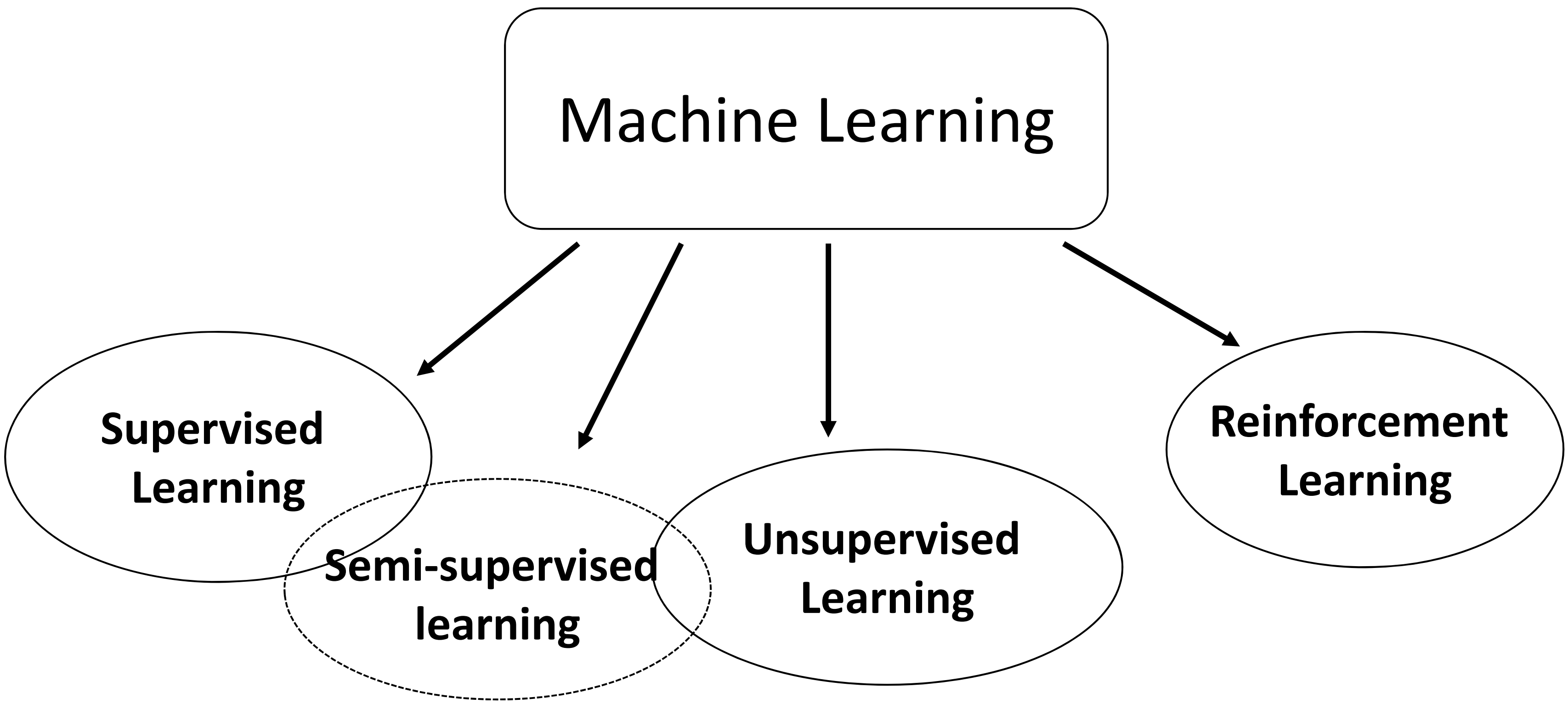
Machine learning



Machine learning studies algorithms that

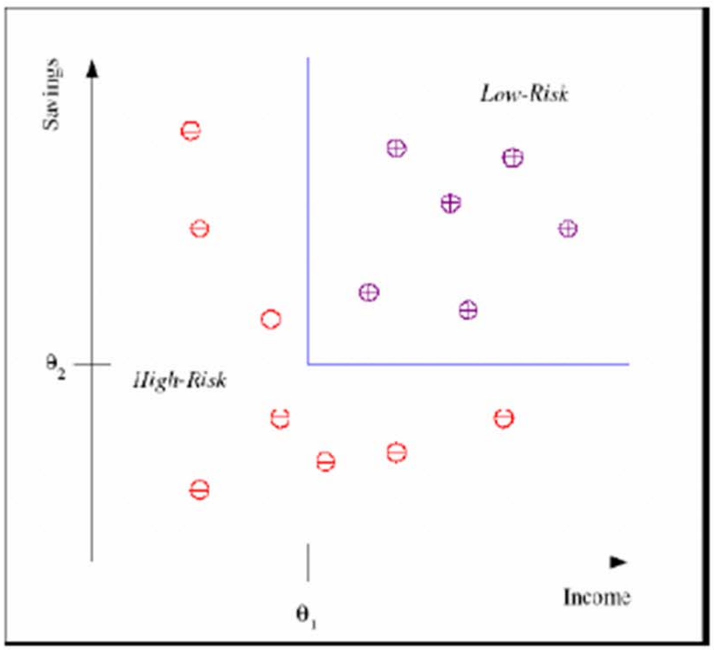
- Improve ***performance*** P
- at some ***task*** T
- based on ***experience*** *E*

Fields of Study



Supervised Learning

- Learn to predict output from input
  - e.g. classify a loan applicant as low-risk or high risk based on income and savings



Unsupervised learning

- Discover groups of similar examples within the data – *clustering*
- Learn the underlying distribution that generates the data – *density estimation*
- Project a high-dimensional data to a low-dimensional space for the purpose of compression or visualization - *dimension reduction*

Reinforcement Learning

- Learn to act
- An agent
  - observes the environment
  - Takes actions
  - Receives awards
  - Goal: maximize rewards
- No examples of optimal outputs are given
- Not covered in this class! Take 533 if you want to learn about this!

When do we need computer to learn?

Do we need learning to do tax return?

Appropriate Applications for Supervised Learning

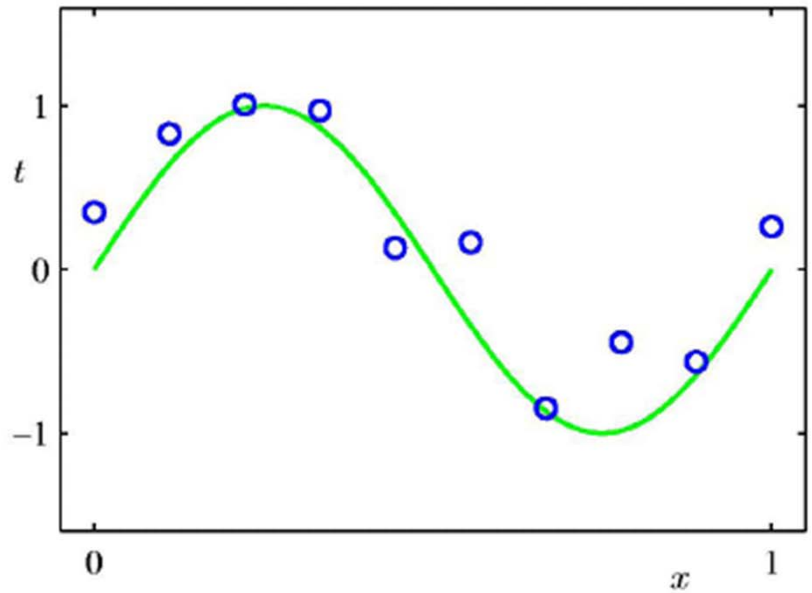
- Situations where there is no human expert
  - x: bond graph of a new molecule
  - f(x): predicted binding strength to AIDS protease molecule
- Situations where humans can perform the task but can’t describe how they do it
  - x: picture of a hand-written character
  - f(x): ascii code of the character
- Situations where the desired function is changing frequently
  - x: description of stock prices and trades for last 10 days
  - f(x): recommended stock transactions
- Situations where each user needs a customized function f
  - x: incoming email message
  - f(x): importance score for presenting to the user (or deleting without presenting)

Supervised Learning

- **Given:** training examples <**x**, *f*(**x**)> for some unknown function *f*
  - x is the input and f(x) is desired output
  - *f*(**x**) can be categorical (*classification*) or numerical (*regression*)

- **Goal:** find a good approximation to *f* so that accurate prediction of output can be made for unseen inputs

Example: regression



The underline function:

$$t = \sin(2\pi x) + \varepsilon$$

where ε is Gaussian noise

Given training examples shown as blue circles

Examples are generated based on the green line (the true underlying function)

Learning goal: make accurate predictions of the *t* values for some ***new*** values of *x* (values that are not included in training)

Polynomial curve fitting

- There are infinite functions that will fit the training data perfectly. In order to learn, we have to focus on a limited set of possible functions
  - We call this our ***hypothesis space***
  - E.g., all M-th order polynomial functions

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M$$

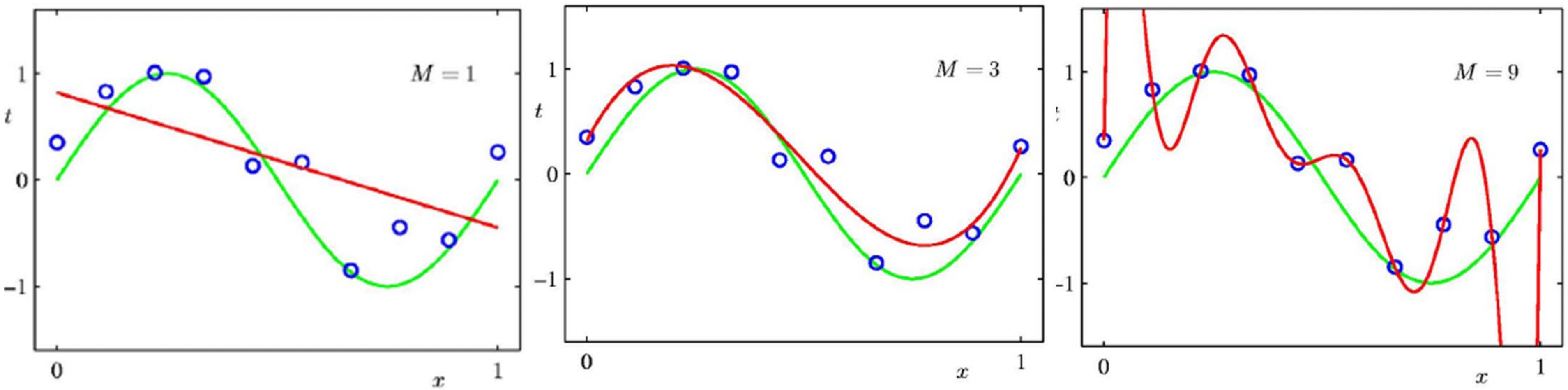
  - **w =** ($w_0, w_1, \ldots, w_M$) represents the unknown parameters that we wish to learn
- Learning here means to find a good set of parameter s ***w*** to minimize some loss function

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} (y(x_n, \mathbf{w}) - t_n)^2$$

Sum-of-squares error

This can be easily solved using standard optimization techniques.
We will not focus on how to solve it here.

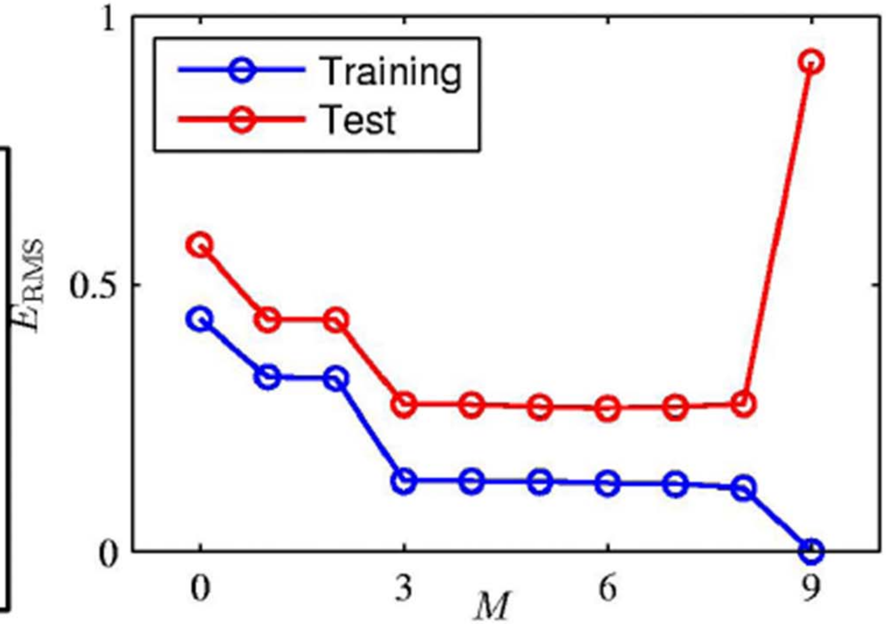
Important Issue: Model Selection



- The red line shows the function learned with different M values
- Which M should we choose – a model selection problem
- Can we use E(**w**) as the criterion to choose M?

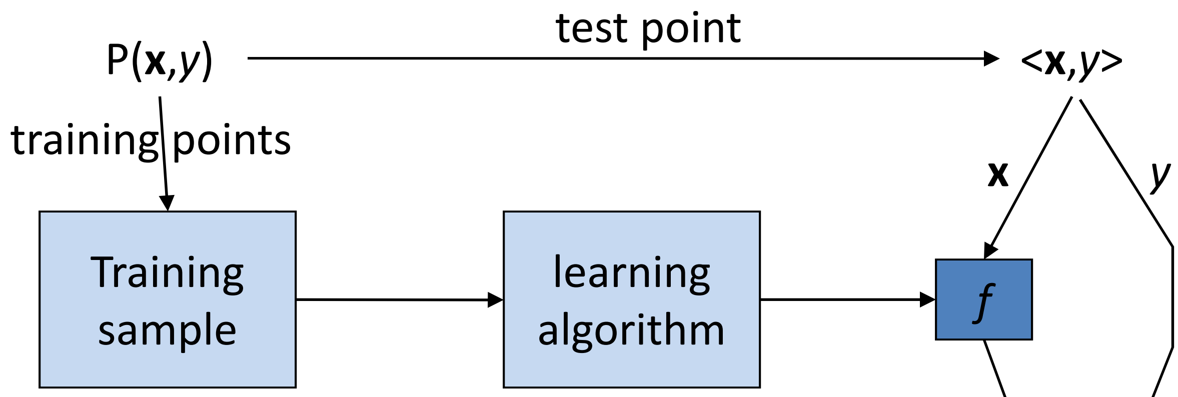
Over-fitting

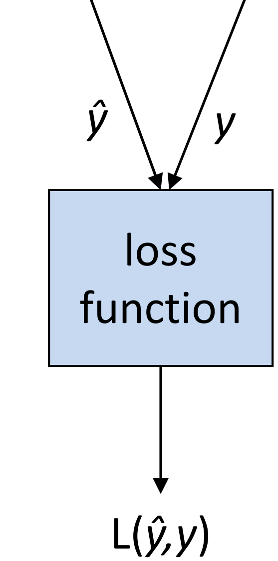
- As M increases, training error decreases monotonically
- The test error, however, starts to increase after a while



- Intuitively Over-fitting happens when
  - There are not enough data to estimate the parameters (e.g., fitting a line to a point)
  - Learning algorithm fits to chance characteristics

Supervised learning: Formal Setting





- Training examples: drawn independently at random according to unknown distribution P(**x**,*y*)
- The learning algorithm analyzes the training examples and produces a classifier *f*
- Given a new point <**x**,*y*> drawn from P, the classifier is given **x** and predicts $\hat{y} = f(\mathbf{x})$
- The loss $L(\hat{y}, y)$ is then measured
- Goal of the learning algorithm: Find the *f* that minimizes the *expected loss* $E_{P(x,y)}[L(f(x),y)]$

Example: Spam Detection

- P(**x**,*y*): distribution of email messages **x** and labels *y* (“spam” or “not spam”)
- Training sample: a set of email messages that have been labeled by the user
- Learning algorithm: what we study in this course!
- *f*: the classifier output by the learning algorithm
- Test point: A new email message **x** (with its true, but hidden, label *y*)
- loss function $L(\hat{y},y)$:

predicted label ŷ	true label *y*: spam	true label *y*: None-spam
spam	0	10
None-spam	1	0

Terminology

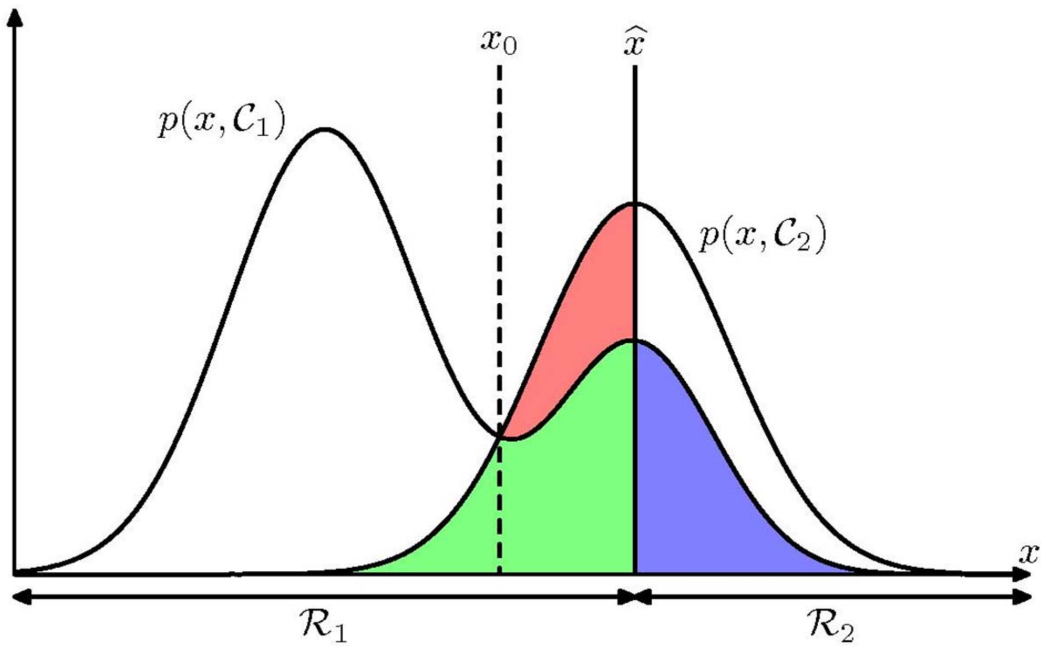
- **<u>Training example</u>** an example of the form <**x**,*y*>
  - ***x*:** feature vector
  - y
    - continuous value for regression problems
    - class label, in [1, 2, …, K] , for classification problems
- **<u>Training Set</u>** a set of training examples drawn randomly from P(**x**,*y*)
- **<u>Target function</u>** the true mapping from **x** to *y*
- **<u>Hypothesis:</u>** a proposed function *h* considered by the learning algorithm to be similar to the target function.
- **<u>Test Set</u>** a set of training examples used to evaluate a proposed hypothesis *h*.
- **<u>Hypothesis space</u>** The space of all hypotheses that can, in principle, be output by a particular learning algorithm

Three Main Approaches

- Learn the joint probability distribution: p(**x**,*y*)
  - This joint distribution captures all the uncertainty about **x** and *y*
- Learn a conditional distribution p(*y* | **x**)
  - Note that p(**x**, y) = p(y|**x**) p(**x**) – so this avoids modeling the distribution of **x**, just tries to capture the probabilistic relationship that maps from **x** to y
- Directly learn a mapping y=f(**x**)
  - In this case, probabilities play no role
- Lets consider how one can make predictions given that we learn p(**x,** y) – decision theory

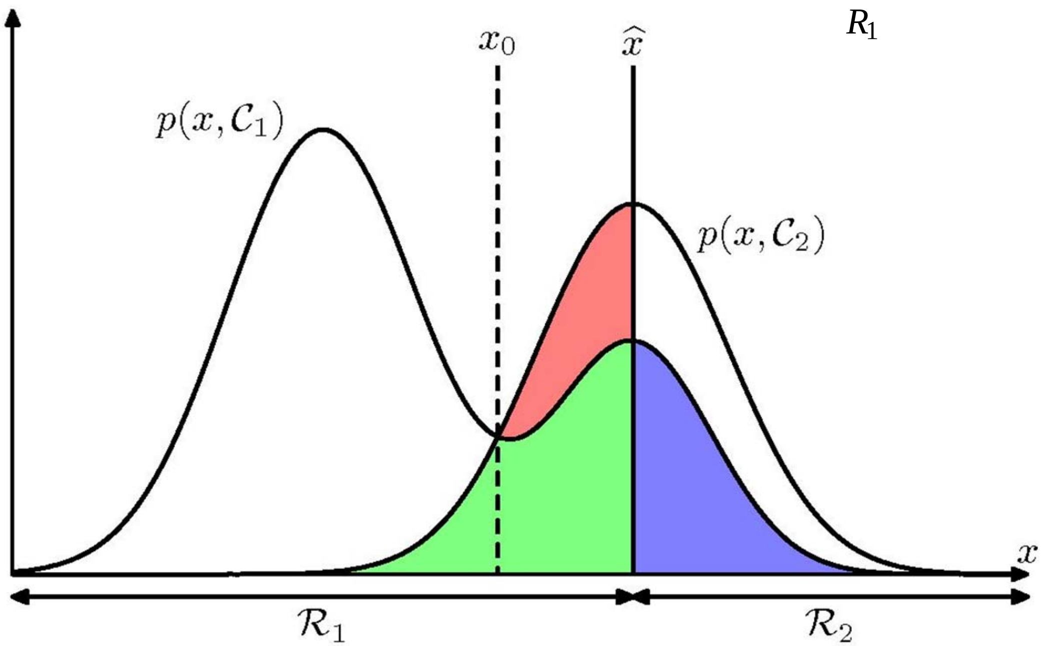
Classification goal 1: Minimizing misclassification rate

- Given a joint distribution p(**x**, y), we need a rule to assign an input **x** to either class 1 or class 2
  - This rule will divide the input space into decision regions R_1 and R_2



$$
\begin{aligned}
p(\text{mistake}) &= p(\mathbf{x} \in R_1, y = c_2) + p(\mathbf{x} \in R_2, y = c_1) \\
&= \int_{R_1} p(\mathbf{x}, y = c_2) d\mathbf{x} + \int_{R_2} p(\mathbf{x}, y = c_1) d\mathbf{x}
\end{aligned}
$$

$$p(\text{mistake}) = p(\mathbf{x} \in R_1, y = c_2) + p(\mathbf{x} \in R_2, y = c_1)$$
$$= \int_{R_1} p(\mathbf{x}, y = c_2) d\mathbf{x} + \int_{R_2} p(\mathbf{x}, y = c_1) d\mathbf{x}$$



Decision rule for minimizing p(mistake):

$$\hat{y}(\mathbf{x}) = \arg\max_{c_i} p(\mathbf{x}, c_i)$$

Note that p(**x**, c) = p(c|**x**) p(**x**), it is equivalent to:

$$\hat{y}(\mathbf{x}) = \arg\max_{c_i} p(c_i \mid \mathbf{x})$$

Classification Goal 2: Minimizing expected loss

- Expected loss: Given x, and p(*y*|x)

The loss of misclassifying an example of class y as class ci

$$\hat{y}(\mathbf{x}) = \arg\min_{c_i} E_{y|\mathbf{x}}[L(c_i, y)]$$

$$= \arg\min_{c_i} \sum_{c_k} L(c_i, c_k)\, p(c_k \mid \mathbf{x})$$

predicted label ŷ	true label *y*	
	spam	None-spam
spam	0	10
none-spam	1	0
P(*y*|**x**)	0.6	0.4

$$E_{y|\mathbf{x}}[L(spam, y)] = ?$$

$$E_{y|\mathbf{x}}[L(nonespam, y)] = ?$$

Key Issues in Machine Learning

- What are good hypothesis spaces?
  - Linear functions? Polynomials?
  - which spaces have been useful in practical applications?
- How to select among different hypothesis spaces?
  - The Model selection problem
  - Trade-off between over-fitting and under-fitting
- How can we optimize accuracy on future data points?
  - This is often called the **Generalization Error** – error on unseen data pts
  - Related to the issue of "overfitting", i.e., the model fitting to the peculiarities rather than the generalities of the data
- What level of confidence should we have in the results? (A statistical question)
  - How much training data is required to find an accurate hypotheses with high probability? This is the topic of learning theory
- Are some learning problems computationally intractable? (A computational question)
  - Some learning problems are provably hard
  - Heuristic / greedy approaches are often used when this is the case
- How can we formulate application problems as machine learning problems? (the engineering question)

Homework Policies

- HW 1 will be posted later today
- Homework is due at the beginning of the class on the due day
- Each student has one allowance of handing in late homework (no more than 48 hours late)
- Collaboration policy
  - Discussions are allowed, but copying of solution or code is not
  - See the **Student Conduct page** on OSU website for information regarding academic dishonesty (http://oregonstate.edu/studentconduct/code/index.php#acdis)

Grading policy

- Grading policy:

  Written homework will not be graded based on correctness. I will record the number of problems that were "completed" (either correctly or incorrectly).

  Completing a problems requires a non-trivial attempt at solving the problem. The judgment of whether a problem was "completed" is left to the instructor.

- Final grades breakdown:
  - Midterm 25%; Final 25%; Final project 25%; Implementation assignments 25%.
  - The resulting letter grade will be decreased by one if a student fails to complete at least 90% of the written homework problems.