Supervised Learning

Introduction

Some slides were adapted/taken from various sources, including Prof. Andrew Ng's Coursera Lectures, Stanford University, Prof. Kilian Q. Weinberger's lectures on Machine Learning, Cornell University, Prof. Sudeshna Sarkar's Lecture on Machine Learning, IIT Kharagpur, Prof. Bing Liu's lecture, University of Illinois at Chicago (UIC), CS231n: Convolutional Neural Networks for Visual Recognition lectures, Stanford University and many more. We thankfully acknowledge them. Students are requested to use this material for their study only and **NOT** to distribute it.

Supervised Learning Setup

- Training data comes in pairs of inputs (x,y), where $x \in R^d$ is the input instance and y its label.

$$D = \{(\mathbf{x}_1, y_1), \ldots \ldots (\mathbf{x}_n, y_n)\} \subseteq \mathcal{R}^d \times C$$

where:

- R^d is the d-dimensional feature space
- x_i is the input vector of the i^{th} sample
- y_i is the label of the i^{th} sample
- C is the label space

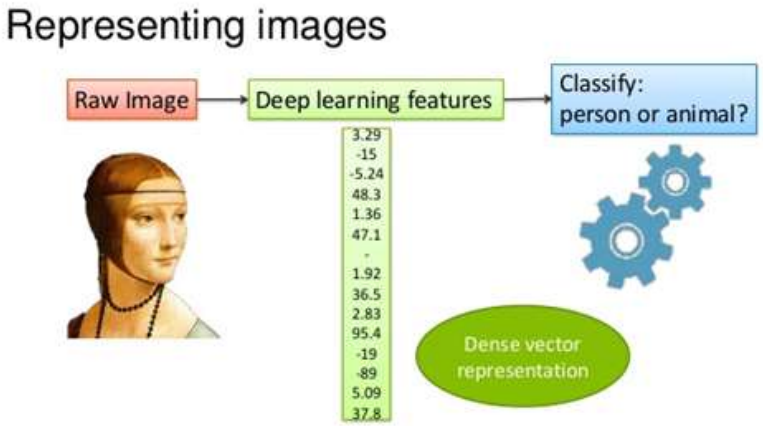
The data points $(\mathbf{x}_1, y_1)$are drawn from some (unknown) distribution $P(x, y)$. Ultimately we would like to learn a function h such that for a new pair,$(x, y) \sim P$, we have h(x)=y with high probability (or h(x)≈y). We will get to this later. For now let us go through some examples of X and Y.

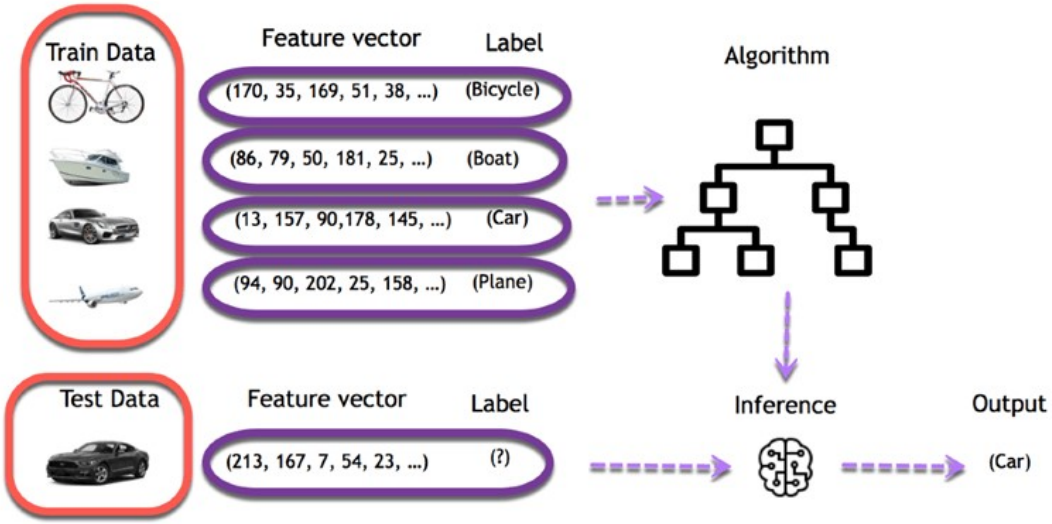
Supervised Learning Setup

There are multiple scenarios for the label space C:

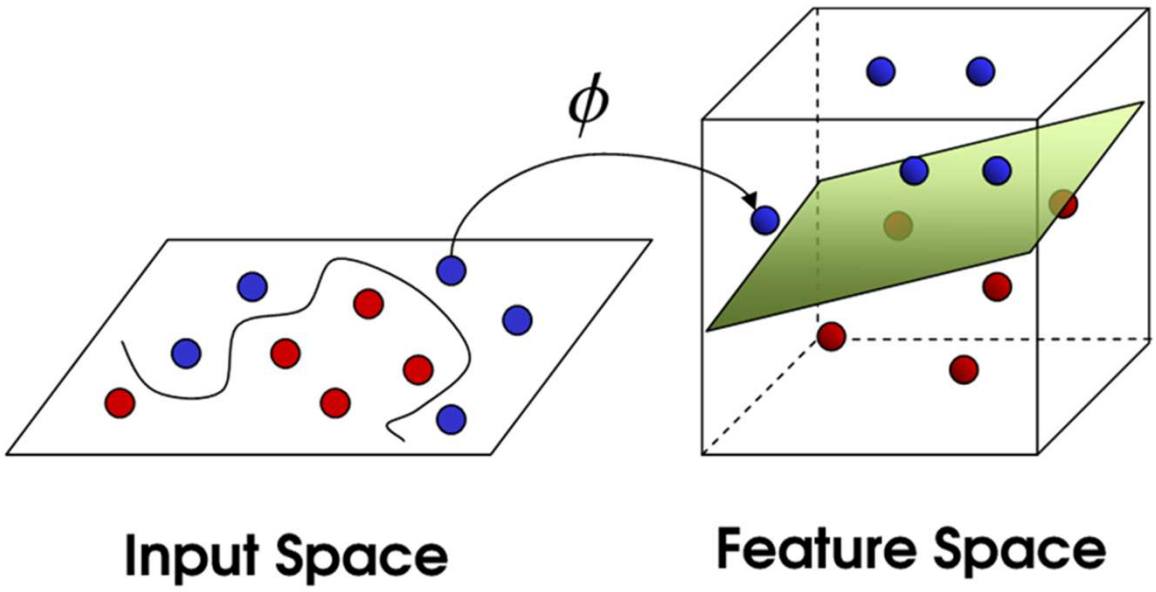
Binary Classification	$C = \{\mathbf{0}, \mathbf{1}\}$ ***or*** $\{+\mathbf{1}, -\mathbf{1}\}$	spam filtering. An email is either spam (+1+1), or not (−1−1).
Multi-class classification	$C = \{1,2, \ldots K)\ K \geq 2$	face classification. A person can be exactly one of K identities
Regression	$C = \mathcal{R}$	predict future temperature or the height of a person.

Feature Space





Feature Space



Hypothesis classes and No Free Lunch

Before we can find a function h, we must specify what type of function it is that we are looking for. It could be an artificial neural network, a decision tree or many other types of classifiers. We call the set of possible functions the hypothesis class.

By specifying the hypothesis class, we are encoding important assumptions about the type of problem we are trying to learn.

The No Free Lunch Theorem states that every successful ML algorithm must make assumptions. This also means that there is no single ML algorithm that works for every setting.

Supervised (Inductive) Learning

- Simplest form: learn a function from examples
  -
  - *f* is the target function
  - An example is a pair $(x, f(x))$

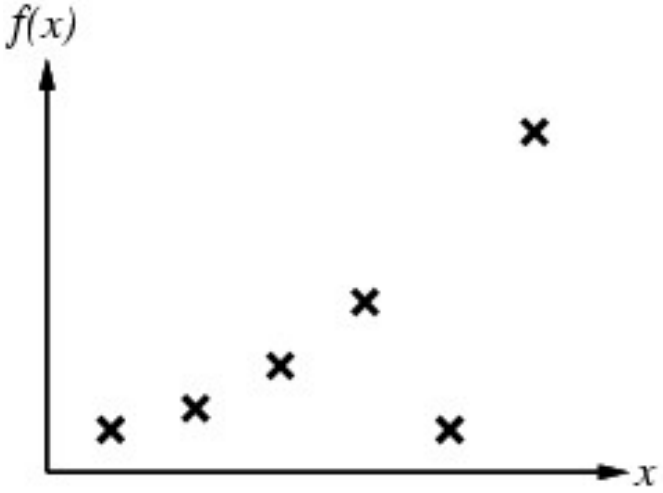
- Pure induction task:
  - **Given a collection of examples of *f*, return a function *h* that approximates *f*.**
  - find a hypothesis *h,* such that $h \approx f$, given a training set of examples

- This is a highly simplified model of real learning:
  - Ignores prior knowledge
  - Assumes examples are given

Supervised (Inductive) Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

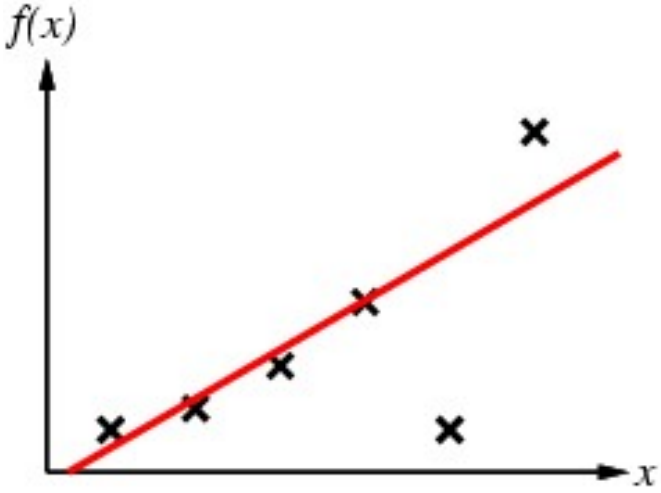
  e.g., curve fitting:



Supervised (Inductive) Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

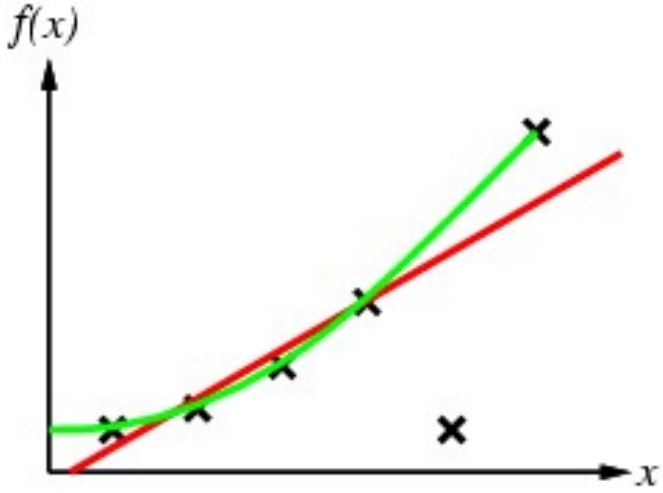
  e.g., curve fitting:



Supervised (Inductive) Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

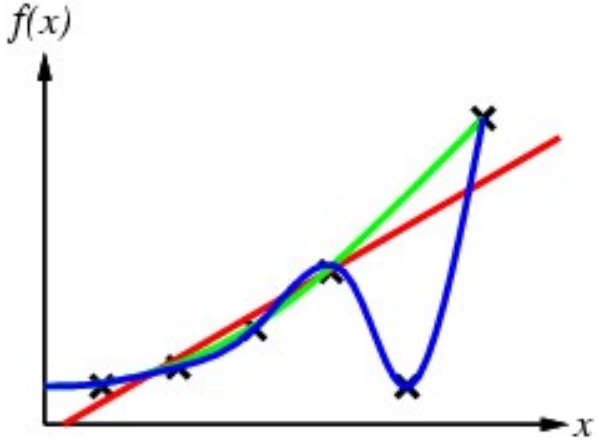
  e.g., curve fitting:



Supervised (Inductive) Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

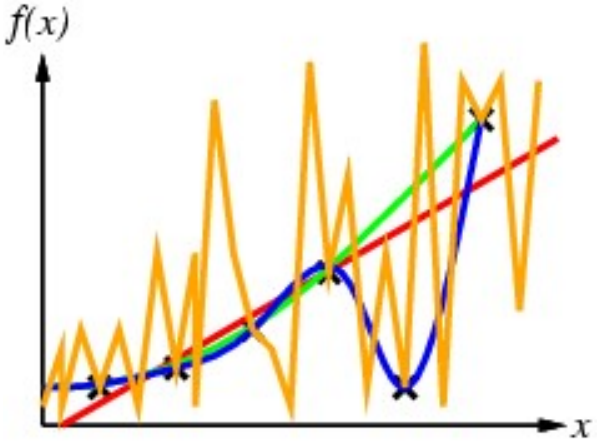
  e.g., curve fitting:



Supervised (Inductive) Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

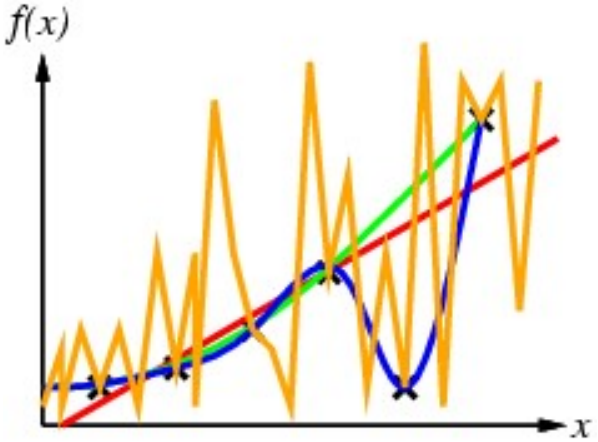
e.g., curve fitting:



Supervised (Inductive) Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

e.g., curve fitting:



- Ockham's razor: prefer the simplest hypothesis consistent with data

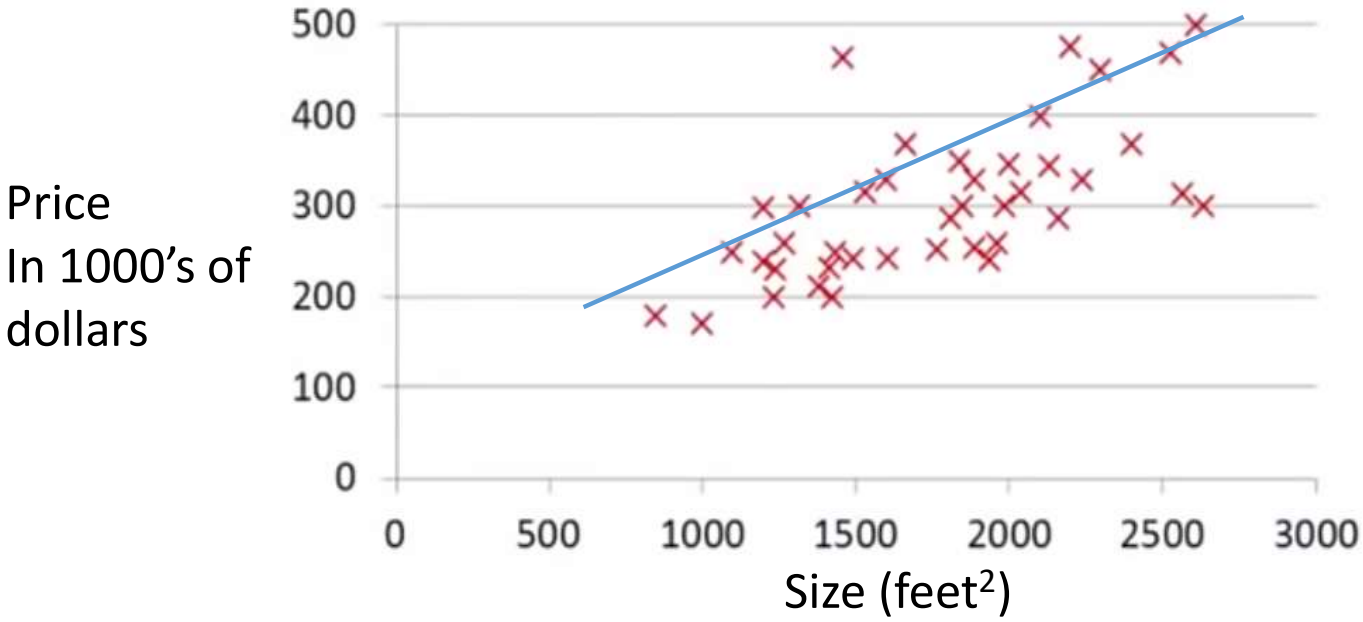
Supervised Learning

- Learning a continuous function: **Regression**

- Learning a discrete function: **Classification**
  - Boolean classification:
    - Each example is classified as true(positive) or false(negative).

Supervised Learning: Linear Regression

Assumption: Hypothesis function is linear



- Given the right answer for each example of the data
  - Regression: Predict real valued data

Supervised Learning: Linear Regression

Training set of housing prices

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

Notation:

m = Number of training examples

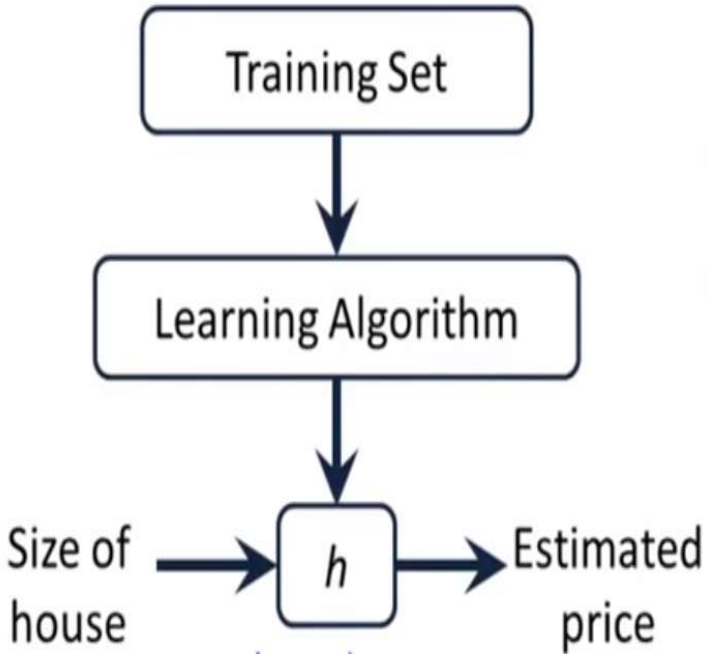
x's = "input" variable / features

y's = "output" variable / "target" variable

(x,y) → one training example $\quad x^{(i)} = 2104$

$(x^{(i)},y^{(i)})$ → i^{th} training example $\quad y^{(i)} = 460$

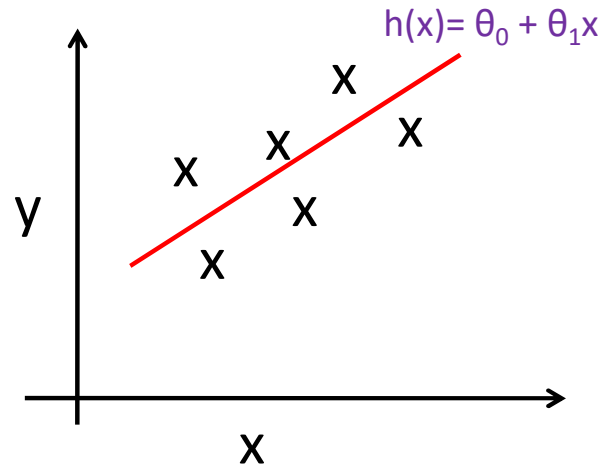
Supervised Learning: Linear Regression



x hypothesis Estimated value of y

How do we represent h

$h_\theta(x) = h(x) = \theta_0 + \theta_1 x$



Univariate linear regression: linear regression with one variable

Linear Regression: Cost Function

Training Set

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

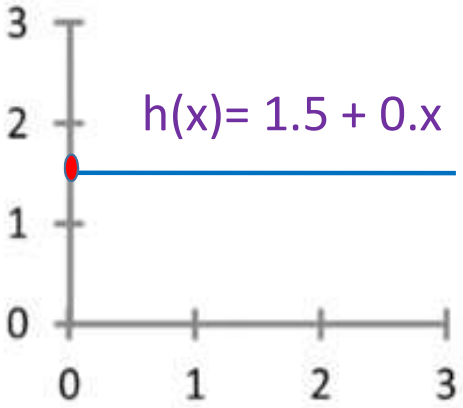
θ_1's → Parameters

How to choose θ_1's

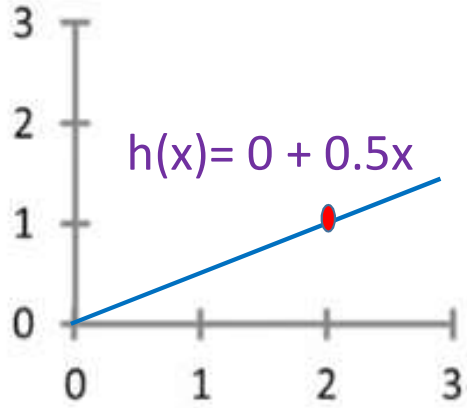
Cost Function

Hypothesis Function:

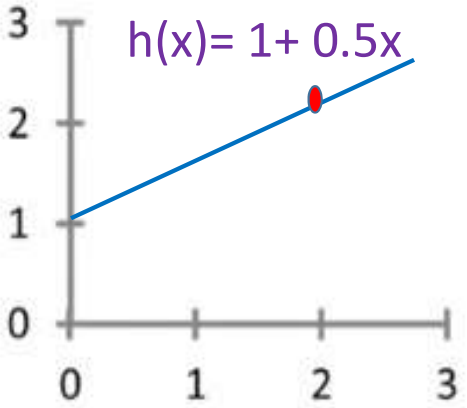
$$h_\theta(x) = \theta_0 + \theta_1 x$$



$\theta_0 = 1.5$
$\theta_1 = 0$

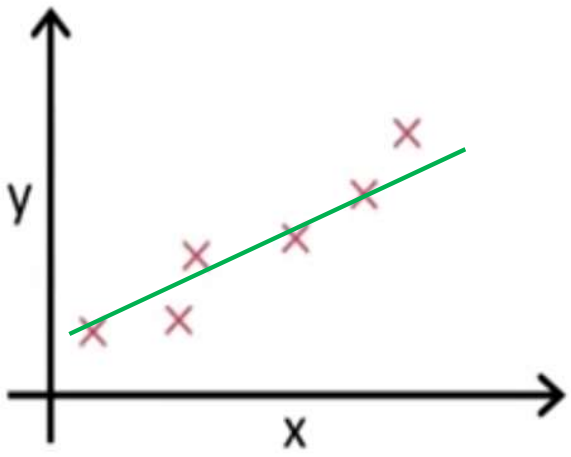


$\theta_0 = 0$
$\theta_1 = 0.5$



$\theta_0 = 1$
$\theta_1 = 0.5$

Cost Function



Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

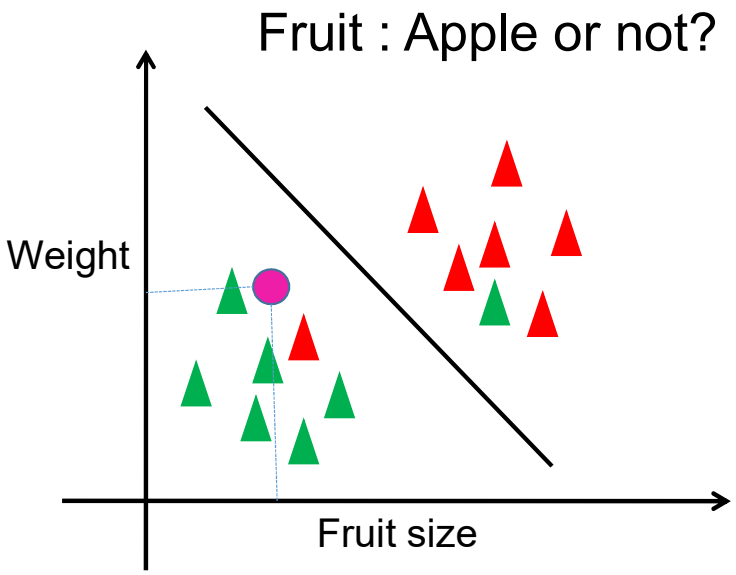
Squared error function

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

m = No. of training samples

Supervised Learning: Classification



Two features
examples

- Features:
  - color
  - texture
  - cost
  - …..

- Given the right answer for each example of the data
  - Classification: discrete no. of outputs

Classification—A Two-Step Process

- Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label
  - The set of tuples used for model construction is training set
  - The model is represented as **classification rules**, **decision trees**, or **mathematical formulae**

- Model usage: for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - **Test set is independent of training set**, otherwise over-fitting will occur
  - If the accuracy is acceptable, use the model to classify data tuples whose class labels are not known

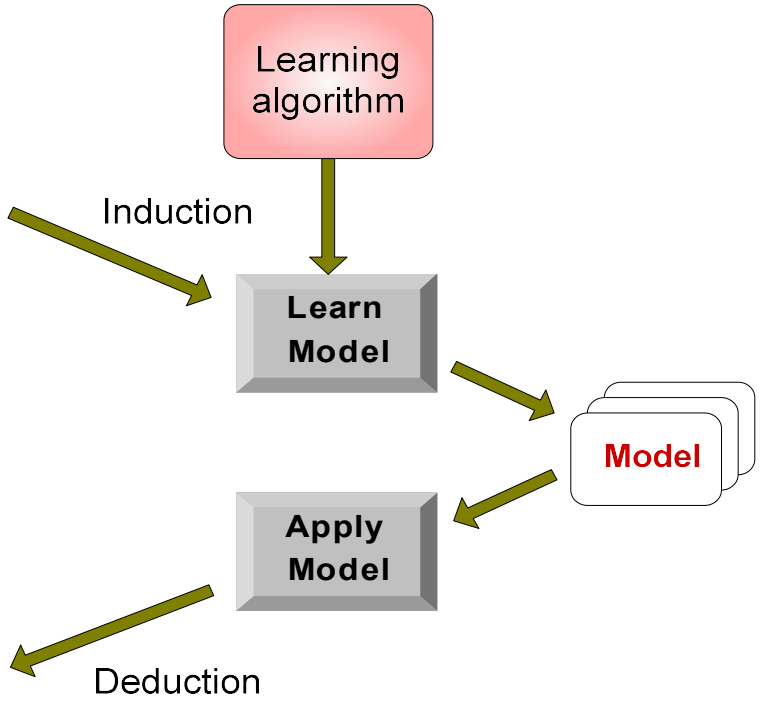
Illustrating Classification Task

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set



Loss Function

- There are typically two steps involved in learning a hypothesis function h().
- First, we select the type of machine learning algorithm that we think is appropriate for this particular learning problem. This defines the hypothesis class $\mathcal{H}$, i.e. the set of functions we can possibly learn.
- The second step is to find the best function within this class, $h \in \mathcal{H}$. This second step is the actual learning process and often, but not always, involves an optimization problem. Essentially, we try to find a function h within the hypothesis class that makes the fewest mistakes within our training data.
- (If there is not a single function we typically try to choose the "simplest" by some notion of simplicity - but we will cover this in more detail in a later class.)
- How can we find the best function? For this we need some way to evaluate what it means for one function to be better than another. This is where the loss function (aka risk function) comes in.
- A loss function evaluates a hypothesis $h \in \mathcal{H}$ on our training data and tells us how bad it is. The higher the loss, the worse it is - a loss of zero means it makes perfect predictions. It is common practice to normalize the loss by the total number of training samples, n, so that the output can be interpreted as the average loss per sample (and is independent of n).

Loss Function: Example

- Zero-one loss: The simplest loss function is the zero-one loss.

- It literally counts how many mistakes an hypothesis function h makes on the training set. For every single example it suffers a loss of 1 if it is mispredicted, and 0 otherwise.

- The normalized zero-one loss returns the fraction of misclassified training samples, also often referred to as the training error.

- The zero-one loss is often used to evaluate classifiers in multi-class/binary classification settings but rarely useful to guide optimization procedures because the function is non-differentiable and non-continuous. Formally, the zero-one loss can be stated has:

$$\mathcal{L}_{0/1}(h) = \frac{1}{n}\sum_{i=1}^{n} \delta_{h(\mathbf{x}_i)\neq y_i}, \text{ where } \delta_{h(\mathbf{x}_i)\neq y_i} = \begin{cases} 1, & \text{if } h(\mathbf{x}_i) \neq y_i \\ 0, & \text{o.w.} \end{cases}$$

- This loss function returns the error rate on this data set D. For every example that the classifier misclassifies (i.e. gets wrong) a loss of 1 is suffered, whereas correctly classified samples lead to 0 loss.

Squared Loss

- The squared loss function is typically used in regression settings. It iterates over all training samples and suffers the loss $(h(\mathbf{x}_i) - y_i)^2$

- The squaring has two effects:
  - the loss suffered is always nonnegative;
  - the loss suffered grows quadratically with the absolute mis-predicted amount.

- The latter property encourages no predictions to be really far off (or the penalty would be so large that a different hypothesis function is likely better suited).

Squared Loss

- On the flipside, if a prediction is very close to be correct, the square will be tiny and little attention will be given to that example to obtain zero error.
- For example, if $|h(\mathbf{x}_i) - y_i| = 0.001$, the squared loss will be even smaller, 0.000001, and will likely never be fully corrected.
- If, given an input x, the label y is probabilistic according to some distribution $P(y|\mathbf{x})$ then the optimal prediction to minimize the squared loss is to predict the expected value, i.e. $h(\mathbf{x}) = \mathbf{E}_{P(y|\mathbf{x})}[y]$
- Formally the squared loss is:

$$\mathcal{L}_{sq}(h) = \frac{1}{n}\sum_{i=1}^{n}(h(\mathbf{x}_i) - y_i)^2.$$

Absolute loss

- Similar to the squared loss, the absolute loss function is also typically used in regression settings.
- It suffers the penalties $|h(\mathbf{x}_i) - y_i|$. Because the suffered loss grows linearly with the mis-predictions it is more suitable for noisy data (when some mis-predictions are unavoidable and shouldn't dominate the loss).
- If, given an input x, the label y is probabilistic according to some distribution $P(y|\mathbf{x})$ then the optimal prediction to minimize the absolute loss is to predict the median value, i.e. $h(\mathbf{x}) = \text{MEDIAN}_{P(y|\mathbf{x})}[y]$
- Formally, the absolute loss can be stated as:

$$\mathcal{L}_{abs}(h) = \frac{1}{n}\sum_{i=1}^{n} |h(\mathbf{x}_i) - y_i|.$$

Generalization

Given a loss function, we can then attempt to find the function h that minimizes the loss:

$$h = \text{argmin}_{h \in \mathcal{H}} \mathcal{L}(h)$$

A big part of machine learning focuses on the question, how to do this minimization efficiently.

If you find a function h(·) with low loss on your data D, how do you know whether it will still get examples right that are not in D?

This is a very important issue of ML and we will get back on it later.

Summary

- We train our classifier by minimizing the training loss

$$\text{Learning: } h^*(\cdot) = \text{argmin}_{h(\cdot)\in\mathcal{H}} \frac{1}{|D_{\text{TR}}|} \sum_{(\mathbf{x},y)\in D_{\text{TR}}} \ell(\mathbf{x}, y|h(\cdot)),$$

  Where $\mathcal{H}$ is the hypothetical class (i.e., the set of all possible classifiers h(·)).
  In other words, we are trying to find a hypothesis h which would have performed well on the past/known data.

- We evaluate our classifier on the testing loss:

$$\text{Evaluation: } \epsilon_{\text{TE}} = \frac{1}{|D_{TE}|} \sum_{(\mathbf{x},y)\in D_{\text{TE}}} \ell(\mathbf{x}, y|h^*(\cdot)).$$

- If the samples are drawn i.i.d. from the same distribution P, then the testing loss is an unbiased estimator of the true **generalization loss**

$$\text{Generalization: } \epsilon = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{P}}[\ell(\mathbf{x}, y|h^*(\cdot))].$$

… to continue