# Supervised learning

## UNIT-IV

- Task of Supervised learning

Given a training set of N example i/p-o/p pairs

$(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)$.

where each $y_j$ was generated by an unknown f^n.

$y = f(x)$,

discover a f^n h that approximates true f^n 'f'.

→ x & y can be any value, need not be numbers.

→ f^n h is a hypothesis

→ Learning is a Search through the Space of possible hypotheses for one that will perform well, even on new examples beyond training set.

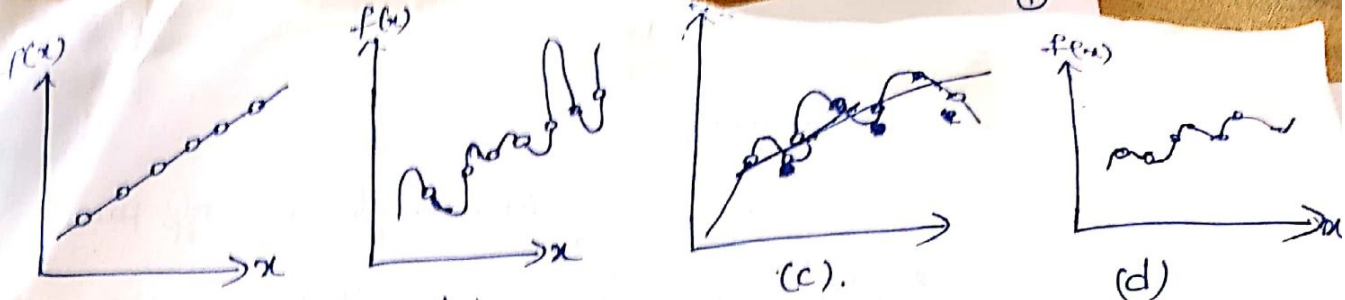
→ To measure accuracy of hypothesis we give it a test set of examples that are distinct from training set.

→ hypothesis generalizes well if it correctly predicts the value of y for examples.

→ Sometimes function f is stochastic - it is not strictly a f^n of x. ~~& what we~~

→ When o/p y is one of a finite set of values (such as Sunny, cloudy or rainy) the learning problem is called classification.

→ when y is a no. (like tommorows temperature) the learning problem is called regression.



(a) (b) (c). (d)

$(x, f(x))$ pairs of a Consistent, linear hypothesis

A Consistent degree-7 polynomial hypothesis for Same data set.

A diff data set which admits an exact degree-6 polynomial fit or app linear fit.

A Simple exact Sinusoidal fit to Same data set.

fig shows a: fitting a $f^n$ of a Single variable to some data points.

points in $(x, y)$ plane where $y = f(x)$.

we don't f., but we will approximate it with $f^n$ h.

Selected from hypothesis Space H, here we take set of polynomials $x^5 + 3x^2 + 2$.

(a) Shows some data with an exact fit by a Straight line $(0.4x + 3)$.

called a Consistent hypothesis bcoz it agrees with all data.

(b) Shows a high degree polynomial that is also Consistent with same data.

illustrates a fundamental problem in inductive learning: how do we choose from among multiple Consistent hypotheses?

to prefer Simplest hypothesis consistent with data.

Called Ockham's razor principle.

(c) Shows a 2nd data set. There is no consistent straight line for this data set, it requires a degree-6 polynomial for an exact fit.

polynomial with 7 parameters doesn't seem to be finding any pattern in data & we can't expect it to generalize well.

A straight line (i.e., not consistent with any of data points, but might generalize fairly well for unseen values 'x'.

- There is a tradeoff b/w complex hypotheses that fit the training data well & simpler hypotheses that may generalize better.

(d) expand the hypotheses space H to allow polynomials over both x & Sin(x). find that data in (c) can be fitted exactly by a simple $f^n$ $ax+b+cSin(x)$. Shows the importance of choice of hypotheses space.

learning problem is realizable if hypotheses space contains true $f^n$.

In some cases, an analyst looking at a problem is willing to make more fine-grained distinctions about hypothesis space, to say - even before seeing any data - not just that a hypothesis is possible or impossible, but rather how probable it is.

Supervised learning can be done by choosing hypotheses $h^*$ i.e. most probable given data:

$$h^* = \underset{h \in H}{\operatorname{argmax}} P(h/\text{data})$$

By Bayes' rule this is equivalent to

$$h^* = \underset{h \in H}{\operatorname{argmax}} P(\text{data}/h) P(h)$$

- we can say prior probability $P(h)$ is high for degree-10 polynomial, lower for degree-7 polynomial
- we allow unusual-looking $f^n$ when data say we really need them, but we discourage them by giving a low prior probability.

Why not let H be class of all Java programs or Turing m/c? (TM/c)

- Every computable $f^n$ can be represented by some TM/c. & i.e., best we can do.
- problem is that it doesn't take into account computational complexity of learning.

There is a trade off b/w expressiveness of a hypothesis space & complexity of finding a good hypothesis within that space.

Ex:- fitting a straight line to data is an easy computation
fitting a high degree polynomials is harder
" TM/cs is undecidable.

Why we prefer simple hypotheses spaces is that we want to use h after we have learned it & computing $h(x)$ & h is linear $f^n$ is guaranteed to be fast, while TM/c is not.

... to wait for a table.

# LEARNING DECISION TREES



Decision tree induction is Simplest & most Successful forms of M/c learning.

We describe representation - the hypothesis Space & then show how to learn a good hypothesis.

## The decision tree representation

- A decision tree represents a function that takes as i/p a vector of attribute values and returns a "decision" - a single o/p value.
- i/p & o/p values can be discrete or continuous.
- ~~Problems~~ with i/ps have discrete values & o/p has exactly two possible values, i.e., Boolean classification. where each example i/p will be classified as true or false.
- Decision tree reaches its decision by performing a Sequence of tests.
- Each internal node in tree corresponds to a test of value of one of i/p attributes, $A_i$ and the branches from node are labeled with possible values of attribute, $A_i = v_{ik}$.

for a table

Aim here is to learn a definition for goal predicate Will Wait.

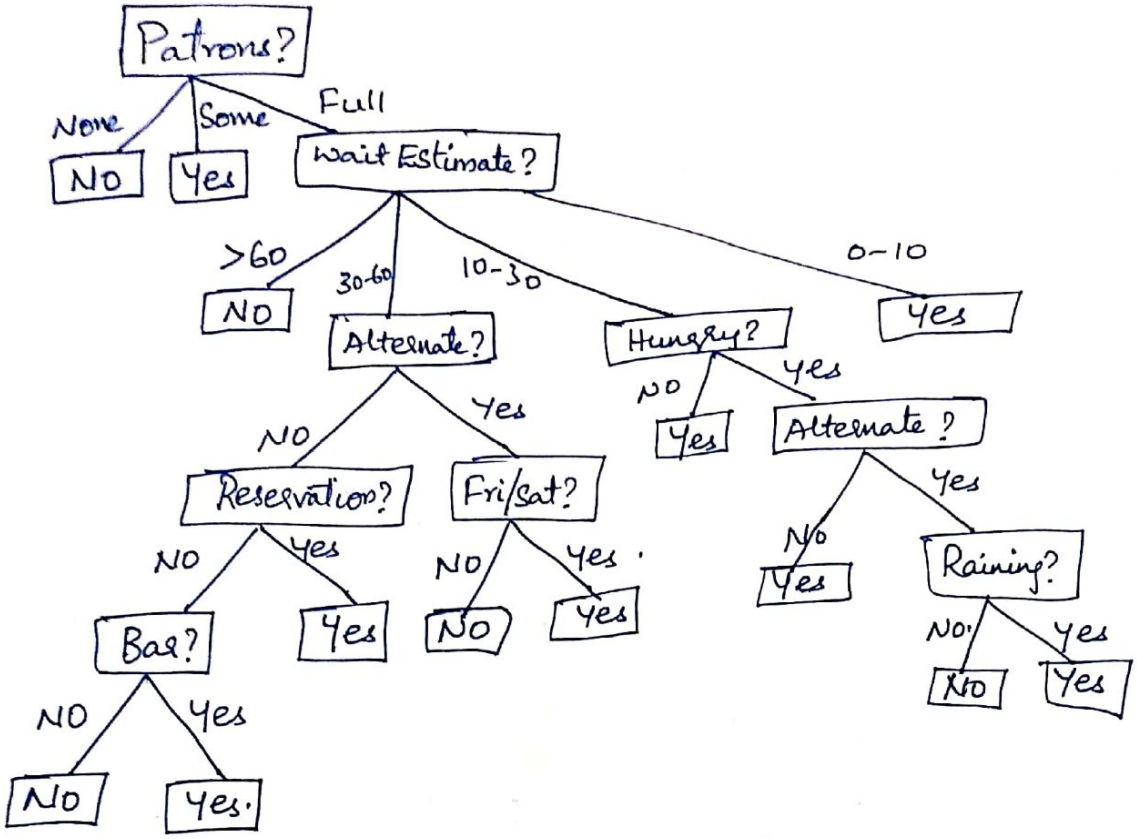
1st we list the attributes:

1. Alternate: whether there is a suitable alternative restaurant nearby.
2. Bar: whether the restaurant has a comfortable bar area to wait in.
3. Fri/Sat: true on Fridays and Saturdays.
4. Hungry: whether we are hungry.
5. Patrons: how many people are in restaurant (values are None, Some & Full).
6. Price: the restaurants price range ($, $$, $$$)
7. Raining: whether it is raining outside
8. Reservation: whether we made a reservation
9. Type: the kind of restaurant (French, Italian, Thai or burger).
10. Wait Estimate: the wait estimated by host (0-10 minutes, 10-30, 30-60, or >60).

Every variable has a small set of possible values; value of WaitEstimate, for Ex: is not an integer, rather it is one of 4 discrete values 0-10, 10-30, 30-60 or >60.

- Notice that tree ignore Price & type attributes.
- Examples are processed by tree starting at root and following the appropriate branch until a leaf is reached.
- Ex: Patrons = Full & WaitEstimate = 0-10 will be classified as true (i.e. yes, we will wait for a table)

A decision tree for deciding whether to wait for a table.




## Expressiveness of decision trees

→ A boolean decision tree is logically equivalent to assertion that the goal attribute is true if & only if i/p attributes satisfy one of the paths leading to a leaf with value true,

In propositional logic, we have

$$Goal \iff (Path_1 \vee Path_2 \vee \cdots),$$

Any $f^n$ in propositional logic can be expressed as a decision tree.

Ex: $Path = (Patrons = Full \wedge WaitEstimate = 0-10)$.

- For variety of problems, decision tree format yields a nice, concise result, but some $f^n$s can't be represented concr

# Inducing decision trees from examples

An example for a Boolean decision tree consists of an $(x, y)$ pair where $\underline{x}$ is a vector of values for i/p attributes,
$y$ is a single Boolean o/p value.

- Positive examples are the ones in which goal will Wait is true $(X_1, X_3 \ldots)$
- Negative examples are the ones in which it is false $(X_2 X_5 \ldots)$

→ We need a tree i.e, consistent with examples & is as small as possible.

→ It is an intractable problem to find the Smallest Consistent tree

→ Decision-Tree-LEARNING Algorithm adopts a greedy divide-and-Conquer strategy: always test the most important attribute first.

→ Type is a poor attribute, b'coz it leaves us with 4 possible outcomes

→ Patrons is fairly important, ~~b'coz if value is~~

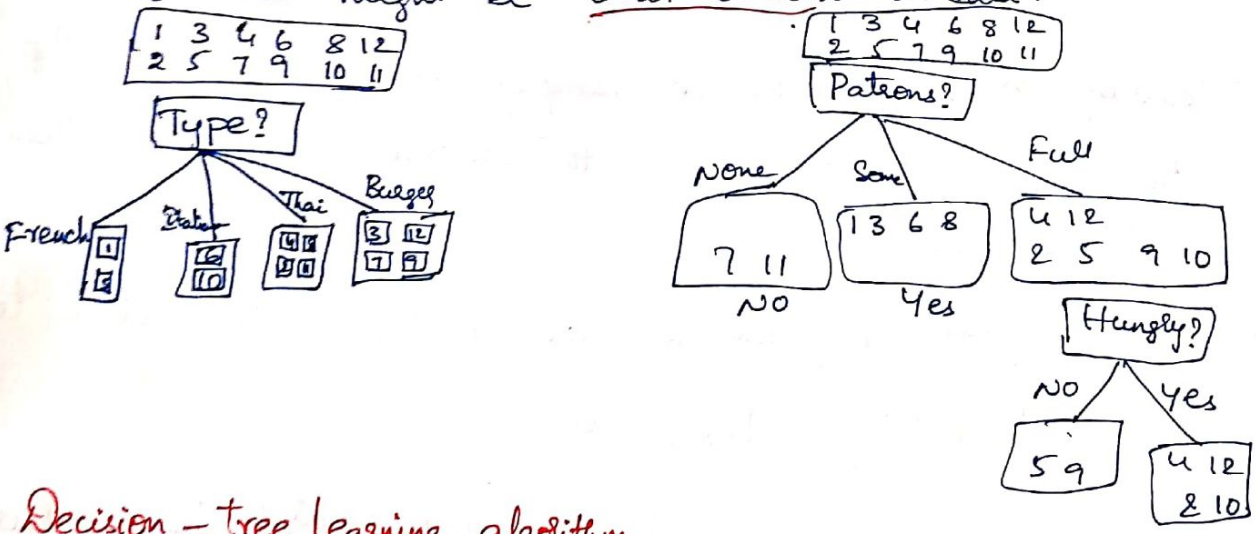
→ Four Cases to Consider recursive problems:

1) If remaining examples are all positive (or all-ve) then we are done: we can answer Yes or NO.
like None & Some.. in the ex

2) If there are some +ve & some -ve examples, then choose the best attribute to split them.
Hungry split to remaining examples

3. If there are no examples left, it means that no example has been observed for this combination of attribute values, & we return a default value calculated from the plurality classification of all examples that were used in constructing the node's parent. These are passed along in variable parent_examples.

4. If there are no attributes left, but both +ve & -ve examples, it means that these examples have exactly the same description but diff classifications. This can happen b'coz There might be error or noise in data.




## Decision-tree learning algorithm

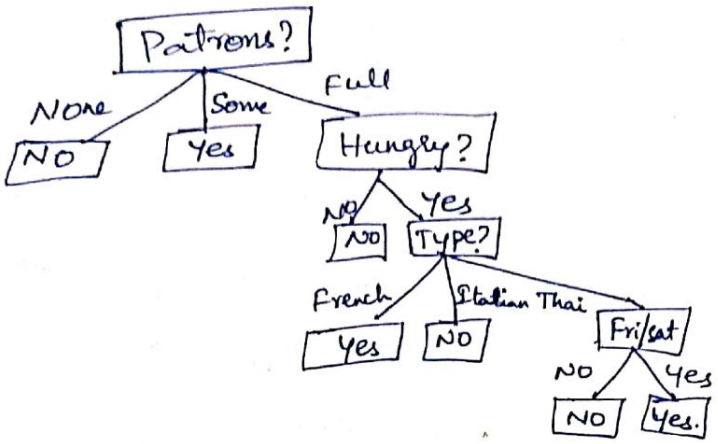
Function Decision-Tree-learning(examples, attributes, parent_examples) returns a tree

if examples is empty then return PLURALITY_VALUE(parent_examples)
else if all examples have same classification then return the classification
else if attributes is empty then return PLURALITY_VALUE(examples)
else
$A \leftarrow \underset{a \in attributes}{\text{argmax}}$ IMPORTANCE(a, examples)
tree ← a new decision tree with root test A.
for each value $v_k$ of A, do
exs ← {e : e ∈ examples and e.A = $v_k$}

↓ selects most common o/p value among a set of examples breaking ties randomly.

Subtree ← Decision-TREE-LEARNING(exs, attributes − A, examples)
add a branch to tree with label (A = $v_k$) and Subtree
return tree

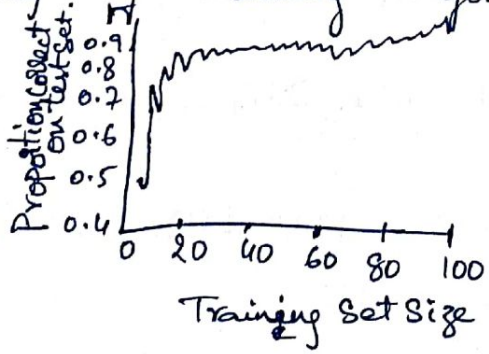
Decision tree




→ learning algorithm looks at examples, not at correct $f^n$ & its hypothesis is consistent with all examples & simpler than original tree.

→ the learning algorithm has no reason to include tests for Raining & Reservation.

→ Accuracy of learning algorithm is evaluated Learning curve.




For 100 exs, we split into a training set & a test set.
we learn a hypothesis h. with training set & measure its accuracy with test set; this is done 1st for size 1 & increasing up to size 99.
For each size we repeat process splitting 20 times & avg results 20 times
Curve shows training set size grows, accuracy increases
So learning curves are called happy graphs.

# Choosing Attribute tests.

- Greedy Search used in decision tree learning is to minimize the depth of final tree.
- The Patrons attribute is not perfect but fairly good. useless attribute like Type leaves the example sets with same proportion of +ve & -ve examples.

→ So we need a formal measure of "fairly good" & "really useless" & can implement IMPORTANCE f^n.

→ we use notion of information gain defined in terms of entropy.

→ Entropy is a measure of uncertainty of a random variable

→ acquisition of information corresponds to a reduction in entropy.

→ A random variable with only one value - a coin that always comes up heads - has no uncertainty & - thus entropy is defined as 0.

→ A flip of a fair coin is equally likely to come up heads & tails, 0 & 1 & we count as "1 bit" of entropy.

→ four-sided die has 2 bits of entropy, b'coz it takes 2 bits to describe one of 4 equally probable choices.

- Consider unfair coin that comes up heads 99% of time. if we guess heads will be wrong only 1% of time. then entropy measure i.e., can be close to zero. but +ve

- Entropy of a random variable V with values $V_K$, each with probability $P(V_K)$.

$$\text{Entropy: } H(V) = \sum_K P(V_K) \log_2 \frac{1}{P(V_K)} = -\sum_K P(V_K) \log_2 P(V_K)$$

Entropy of fair coinflip is 1 bit.

$$H(\text{Fair}) = -(0.5 \log_2 0.5 + 0.5 \log_2 0.5) = 1.$$

if coin is loaded to give 99% heads, we get

$$H(\text{loaded}) = -(0.99 \log_2 0.99 + 0.01 \log_2 0.01)$$
$$\approx 0.08 \text{ bits}.$$

$B(q)$ as entropy of a Boolean random variable i.e., T with probability q:

$$B(q) = -(q \log_2 q + (1-q) \log_2 (1-q))$$

If a training set contains p +ve examples & n -ve examples then entropy of goal attribute on whole set is

$$H(\text{Goal}) = B\left(\frac{p}{p+n}\right).$$

An attribute A with distinct values divides training set E into each subset $E_K$ has $p_K$ +ve & $n_K$ -ve examples. $E_1, \ldots E_d$.

So $B\left(\frac{p_K}{p_K+n_K}\right)$ we need an additional bits of inf to answer question.

randomly chosen examples from training set has $k^{th}$ value for attribute with probability $(p_k + n_k)/(p+n)$.

Expected entropy remaining after testing attribute A is

$$\text{Remainder}(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p+n} B\left(\frac{p_k}{p_k + n_k}\right).$$

Information gain from attribute test on A is expected reduction in entropy:

$$\text{Gain}(A) = B\left(\frac{p}{p+n}\right) - \text{Remainder}(A).$$

## Generalization & Overfitting

The Decision-Tree-Learning Algo will generate a large tree when there is actually no pattern to be found.

- Overfitting occurs when target f^n is not random.
- In overfitting no. of i/p attributes grows & we increase the no. of training examples.
- Decision tree pruning combats overfitting.
  - → Pruning eliminates nodes that are not relevant.
  - → we start with a full tree generated by Decision-Tree-learning we look at test node that has only leaf nodes as descendants, If test appears to be irrelevant - detecting only noise in data - then we eliminate the test, replacing it with a leaf node. like this we consider each test with only leaf descendants, until each one has either pruned or accepted.

How do we detect that a node is testing an irrelevant attribute?

Suppose we are at a node consisting of P +ve & n -ve examples.

If attribute is irrelevant, we would except that it would Split the examples into Subsets that each have roughly the Same proportion of +ve examples as whole set P/(P+n) & so information gain will be close to zero.

information gain is a good clue to irrelevance.

How large a gain should we require in order to Split on a particular attribute?

- we can answer this question by using a Statistical Significance test. Such a test begins by assuming there is no underlying pattern (null hypothesis).
- Then actual data are analyzed to calculate the extent to which they deviate from a perfect absence of pattern.
- If degree of deviation is statistically unlikely (means 5% probability or less), then it is considered to be good evidence for presence of a Significant pattern in data.
- we need to calculate the probability under null hypothesis, a Sample of Size $v = n+p$. which is excepted distribution of +ve & -ve examples.

- we can measure the deviation by comparing actual numbers of +ve & -ve examples in each subset $P_k$ & $n_k$ with expected nos $\hat{P}_k$ & $\hat{n}_k$ assuming true irrelevance:

$$\hat{P}_k = P \times \frac{P_k + n_k}{P+n} \qquad \hat{n}_k = n \times \frac{P_k + n_k}{P+n}$$

# Evaluating and Choosing the Best Hypothesis

→ We want to learn a hypothesis that fits the future data set.

– To make it precise we need to define "future data" and "best fit"

– We make a Stationarity assumption: that there is a probability distribution over examples that remains Stationary over time.

– Each example data point is a random variable $E_j$ whose observed value $e_j = (x_j, y_j)$ is sampled from that distribution & is independent of previous examples.

$$P(E_j \mid E_{j-1}, E_{j-2}, \dots) = P(E_j).$$

& each example has an identical prior probability distribution:

$$P(E_j) = P(E_{j-1}) = P(E_{j-2}) = \dots$$

– Examples that satisfy these assumptions are called independant & identically distributed or i.i.d.

– i.i.d. assumption connects past to future. future could be anything.

→ Next step is to define "best fit" we define error rate of a hypothesis as the proportion of mistakes it makes - proportion of times that $h(x) \neq y$ for any $(x, y)$ example

→ just because a hypothesis h has a low error rate on training set doesn't mean that it will generalize well.

+ zero. but +ve.

→ Professor knows that an exam will not accurately evaluate students if they have been already seen the exam questions

→ Ully to get an accurate evaluation of a hypothesis, we need to test it on a set of examples it has not seen yet.

→ Simplest approach is one we have seen already: randomly split the available data into a training set from which learning algorithm produces h & a test set on which accuracy of h is evaluated. This method, sometimes called 'hold-out cross-validation' which has a disadvantage that it fails to use all the available data.

→ We can squeeze more out of data & still get an accurate estimate using a technique called K-fold cross-validation.

→ each examples serves double duty – as training data and test data.

data → K subsets – K rounds learning, 1/K data as test

→ 1st we split the data into K equal subsets. then perform K rounds of learning. on each round 1/K of data is held out as a test set & remaining example are used as training data.

→ The average test set score of K rounds should then be a better estimate than a single score.

→ Popular values for K are 5 & 10 – enough to give an estimate i.e., statistically like to be accurate, at a cost of 5 to 10 times longer computation time.

extreme is K=n, known as leave-one-out cross-validation or LOOCV

Users frequently invalidate their results by peeking at test data
- Peeking: A learning algorithm has various "knobs" that can be twiddled to tune its behaviour.
- Researcher generates hypotheses for various different settings of knobs, measures their error rates on test set and reports the error rate of best hypothesis. Alas, Peeking has occured!
- The reason is hypothesis was selected on basis of its test set error rate, so information about test set has leaked into learning algorithm.
- Peeking is a consequence of using test-set performance to both choose a hypothesis and evaluate it.
- The way to avoid this is to really hold the test set out - lock it away until you are completely done with learning & simply wish to obtain an independent evaluation of final hypothesis.
- If test set is locked away, but you still want to measure performance on unseen data as a way of selecting a good hypothesis, then divide the available data into a training set & a validation set.

## Model Selection: Complexity Versus Goodness of fit

- The Higher-degree polynomials can fit the training data better, but if degree is too high they will overfit & perform poorly on validation data.
- Choosing degree of polynomial is an instance of problem of model selection.

Task of finding the best hypothesis may have two tasks:

model selection defines the hypothesis space & then optimization finds the best hypothesis within that space.

- In this section we will see how to select among models that are parameterized by size.

→ Ex:- with polynomials size = 1 for linear $f^n$
2 for quadratics.

→ In decision trees, size could be no. of nodes in tree.

→ In all cases we want to find value of size parameter that best balances underfitting & overfitting to give the best test set accuracy.

→ Wrapper takes a learning algorithm as an argument (Decision-Tree-learning$^{Ex}$)

→ Wrapper enumerates models according to a parameter, size.

→ for each size, it uses cross validation on learner to compute the average error rate on training & test sets.

→ The training set error decreases monotonically, while the validation set error decreases at first, then increases when model begins to overfit.

→ cross validation procedure picks the value of size with lowest validation set error; bottom of U-shaped curve.

## From error rates to loss

- We have been trying to minimize error rate.

→ Consider problem of classifying email messages as Spam or non-Spam.

→ It is worse to classify non-Spam as Spam. than classify Spam as non Spam.

→ So a classifier with a 1% error rate, where almost all errors were classifying Spam as non-Spam, would be better than a classifier with only a 0.5% error rate, if most of those errors were classifying nonSpam as Spam.

→ In M/c learning it is traditional to express utilities by means of a loss f^n.

→ loss f^n $L(x, y, \hat{y})$ is defined as amount of utility lost by predicting $h(x) = \hat{y}$ when the correct answer is $f(x) = y$:

$L(x, y, \hat{y})$ = utility (result of using $y$ given an i/p $x$) − utility (result of using $\hat{y}$ given an i/p $x$).

$L(\text{Spam}, \text{noSpam}) = 1$, $L(\text{noSpam}, \text{Spam}) = 10$.

Let $\mathcal{E}$ be set of possible i/p-o/p examples.
Then expected generalization loss for a hypothesis $h$.
is

$$\text{GenLoss}_L(h) = \sum_{(x,y)\in\mathcal{E}} L(y, h(x))\, P(x,y),$$

---

& best hypothesis, $h^*$ is one with minimum expected generalization loss:

~~$h^* = \underset{h\in H}{\text{argmin}}\ \text{GenLoss}(h)$~~

$$h^* = \underset{h\in H}{\text{argmin}}\ \text{GenLoss}_L(h).$$

& $P(x,y)$ is not known, learning agent can only estimate generalization loss with empirical loss on a set of examples, $E$:

$$\text{EmpLoss}_{L,E}(h) = \frac{1}{N} \sum_{(x,y)\in E} L(y, h(x)).$$

estimated best hypothesis $\hat{h}^*$ is then the one with min empirical loss:

$$\hat{h}^* = \underset{h\in H}{\text{argmin}}\ \text{EmpLoss}_{L,E}(h).$$

There are 4 reasons why $\hat{h}^*$ may differ from true function, $f$: unrealizability, variance, noise, computational complexity.

1st – f may not be realizable – may not be in H – or may be present in such a way that other hypotheses are preferred.

2nd – a learning algorithm will return different hypotheses for different sets of examples, even if those sets are drawn from same true f^n. f & those hypothesis will make different predictions on new examples.

~~3rd~~. higher the variance, higher the probability of error.

3rd – f may be nondeterministic or noisy – it may return different values for f(x) each time x occurs.
noise can't be predicted.
in many cases, it arises b'coz observed labels y are result of attributes of environment not listed in x.

4th – H is complex, it can be computationally intractable to systematically search the whole hypothesis space.
Best we can do is a local search that explores only part of space.
That gives an appr.^n error,

... of time.



## Regularization

We saw how to do model selection with cross-validation on model size.

An alternate approach to search for a hypothesis that directly minimizes the weighted sum of empirical loss & complexity of hypothesis, total cost.

$$Cost(h) = EmpLoss(h) + \lambda Complexity(h)$$

$$\hat{h}^* = \underset{h \in H}{\operatorname{argmin}}\ Cost(h).$$

$\lambda$ is a parameter, a +ve number that serves as a conversion rate between loss & hypothesis Complexity

This approach combines loss & complexity into one metric, to find best hypothesis all at once.

# Theory of Learning

- How can we be sure that our learning algorithm has produced a hypothesis that will predict correct value for previously unseen i/ps?
- How do we know that hypothesis h is close to target fn. 'f' if we don't know what f is?
- How many examples do we need to get a good h?
  - → What hypothesis space should we use?
  - → If hypothesis space is very complex, can we find best h. do we have to settle for local maximum in space of hypotheses?
  - → How complex should h be?
  - → How do we avoid overfitting?

- we start with how many examples are needed for learning?
- The learning curve for decision tree on restaurant problem improves with more training data.
  - → learning curves are useful, but specific to a particular learning algorithm on a particular problem.
  - → Computational learning theory will address the principles governing the no. of examples needed.
  - → Principle is that any hypothesis that is seriously wrong will almost certainly be "found out" with high probability after a small no. of examples, b'coz it will make an incorrect predict

that

- holds 99% of time.

- Thus, any hypothesis that is consistent with a sufficiently large set of training examples is unlikely to be seriously wrong: i.e., it must be probably approximately correct.

→ Any learning algorithm that returns hypotheses that are probably approximately correct is called a PAC learning algorithm.

→ With this we can provide bounds on performance of various learning algorithms.

→ PAC-learning theorems, are logical consequences of axioms.

→ Simplest PAC theorems deal with Boolean functions for which the 0/1 loss is appropriate.

→ Error rate of hypothesis h, is defined as expected generalization error for examples drawn from stationary distribution.

$$error(h) = GenLoss_{L_{0/1}}(h) = \sum_{x,y} L_{0/1}(y, h(x)) P(x, y).$$

→ A hypothesis h is called approximately correct if $error(h) \leq \epsilon$, where $\epsilon$ is a small constant.

- Approximately correct hypothesis can be close to true $f^n$ in hypothesis space: it lies inside "$\epsilon$-ball around the true $f^n$ f. hypothesis space outside this ball is called $H_{bad}$.

→ Probability that a seriously wrong hypothesis $h_b \in H_{bad}$ is consistent with 1st N examples.

$error(h_b) > \epsilon$.

→ probability that it agrees with a given example is at most $1-\epsilon$.

→ Since examples are independent, the bound for N examples is

$$P(h_b \text{ agrees with } N \text{ examples}) \leq (1-\epsilon)^N.$$

→ Probability that $H_{bad}$ contains at least one consistent hypothesis is bounded by sum of individual probabilities

$$P(H_{bad} \text{ Contains a Consistent hypothesis}) \leq |H_{bad}|(1-\epsilon)^N \leq |H|(1-\epsilon)^N.$$

→ To reduce the probability of this event.

$$|H|(1-\epsilon)^N \leq \delta.$$

Given that $1-\epsilon \leq e^{-\epsilon}$, we can achieve this if we allow algorithm to see $N \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + \ln|H|\right)$. algorithms.

## PAC learning examples: Learning Decision lists.

→ How to apply PAC learning to new hypothesis space: decision lists.

→ A decision list consists of a series of tests, each of which is a conjunction of literals.

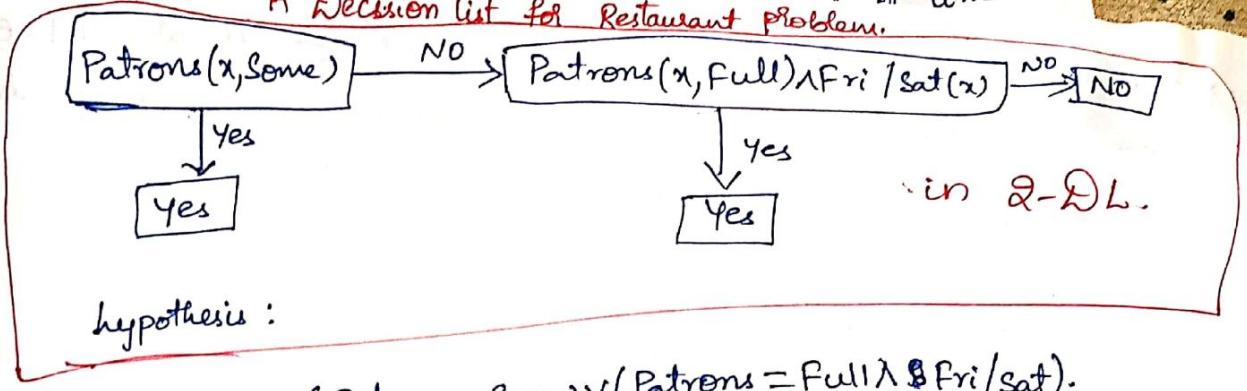
→ If a test succeeds ~~the~~ when applied to an example description, the decision list specifies the value to be returned.

→ If test fails, processing continues with next test in list.

→ Decision list resembles decision trees, but overall structure is simpler: they branch only in one direction.

→ The individual tests are more complex.

A Decision list for Restaurant problem. of time.




in 2-DL.

Hypothesis :

WillWait ⇔ (Patrons = Some) ∨ (Patrons = Full ∧ Fri/Sat).

→ If we allow tests of arbitary size, then decision lists can represent any Boolean fⁿ.

→ If we restrict size of each test to at most K literals, then it is possible for learning algorithm to generalize successfully from a small no. of examples. we call this language K-DL.

→ language refered to by K-DL depends on attributes used to describe the examples.

→ K-DL(n) to denote a K-DL language using n Boolean attributes.

→ 1st task is show that K-DL is learnable. - i.e., any fⁿ in K-DL can be approximated accurately after training on a reasonable no. of examples.

→ So, we need to calculate no. of hypothesis in lang.

→ let language of tests - Conjunctions of at most K literals using n attributes - be Conj (n, K).

→ decision list is constructed of tests & b'coz each test can be attached to either Yes or NO or can be absent from decision list, there are at most $3^{|Conj(n,K)|}$ distinct sets of Component tests.

No. of Conjunctions of K literals from n attributes is given by

$$|Conj(n,k)| = \sum_{i=0}^{k} \binom{2n}{i} = O(n^k).$$

$$|k\text{-}DL(n)| = 2^{O(n^k \log_2(n^k))}$$

To show that no. of examples needed for PAC-learning a k-DL $f^n$ is polynomial in n:

$$n \geq \frac{1}{\epsilon}\left(\ln\frac{1}{\delta} + O\left(n^k \log_2(n^k)\right)\right)$$

→ Any algorithm that returns a consistent decision list will PAC-learn a k-DL function in a reasonable no. of examples, for small k.

→ Next task is to find an efficient algorithm that returns a consistent decision list.

→ we use a greedy algorithm called Decision-list-learning that repeatedly finds a test that agrees exactly with some subset of training set.

→ It finds such a test, it adds it to the decision list under construction & removes corresponding examples.

→ This is repeated until there are no examples left.

DECISION-LIST-LEARNING

function Decision-List-Learning (Examples) returns a decision list, or failure

if examples is empty then return the trivial decision list NO

$t \leftarrow$ a test that matches a nonempty subset $examples_t$ of examples.

such that the members of $examples_t$ are all positive or all negative

if there is no such t then return failure

if the examples in $examples_t$ are positive then

$o \leftarrow$ Yes else $o \leftarrow$ NO.

return a decision list with initial test t and outcome o and remaining tests given by

DECISION-LIST-LEARNING (examples − $examples_t$).

## Algorithm for Learning Decision Lists

- This algorithm doesn't specify method for selecting next test to add to the decision list.
- Although the formal results given earlier don't depend on selection method, it would seem reasonable to prefer small tests that match large sets of uniformly classified examples, so that overall decision list will be compact.

## ...ion and Classifications with Linear Models

- Here see class of linear-f^ns of Continuous valued I/p's.

→ regression with a univariate linear f^n or "fitting a Straight line"

→ Then Multivariate case

→ Then How to turn linear f^ns into classifiers by applying hard and Soft thresholds.

## Univariate Linear Regression

→ A univariate linear f^n with i/p x & o/p y has form $y = w_1 x + w_0$. where $w_0$ & $w_1$ are real-valued coefficients. to be learned

→ we use w b' coz we think coefficients as weights.

→ y is changed by changing relative weight of one term & another.

→ we define **w** to be vector $[w_0, w_1)$. & define

$$h_{\mathbf{w}}(x) = w_1 x + w_0.$$

→ figure shows an example of a training set of n points in the x, y plane, each point representing the size in Square feet & price of a house offered for Sale.

→ Task of finding the $h_w$ that best fits these data is called linear regression.

→ To fit a line to the data, we have to do is find the values of weights $[w_0, w_1]$ that minimize the empirical loss.

To use Squared loss f^n, L_2 Sum over all training examples:

$$\text{Loss}(h_w) = \sum_{j=1}^{N} L_2(y_j, h_w(x_j)) = \sum_{j=1}^{N} (y_j - h_w(x_j))^2$$
$$= \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2.$$

$$W^* = \text{argmin}_w \text{Loss}(h_w).$$

Sum $\sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$ is minimized when $w_0$ & $w_1$ are zero.

$$\left.\begin{aligned} &\frac{\partial}{\partial w_0} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 = 0 \\ &\quad \& \\ &\frac{\partial}{\partial w_1} \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2 = 0. \end{aligned}\right\} 18.2$$

Equations have unique Sol^n.

$$\left.\begin{aligned} w_1 &= \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2} \\ w_0 &= \left(\sum y_j - w_1 (\sum x_j)\right)/N. \end{aligned}\right\} 18.3.$$

Sol^n for figure $w_1 = 0.232$, $w_0 = 2.46$. & line with those weights is shown as a dashed line

→ Learning involve adjusting weights to minimize a lo
So, it helps to have a picture of what's going on in weight Spac

→ Weight Space is defined by all possible Setting of weights.

→ Alternatively, we can ...

→ For univariate linear regression, weight space defined by $w_0$ & $w_1$ is two-dimensional.

→ loss as a $f^n$ of $w_0$ & $w_1$ in 3D plot. in figure 18.13(b).

→ loss $f^n$ is convex, it is true for every linear regression problem with an $L_2$ Loss $f^n$ & implies that there are no local minima.

→ If we need to fit lines to data, we apply (18.3).

→ Equations defining min loss (18.2) will often have no closed-form Sol$^n$.

→ we use gradient descent.

we choose any starting point in weight space, here a point in $(w_0, w_1)$ plane - and then move to a neighboring point that is downhill, repeating until we converge on min possible loss.

```
w ← any pt in parameter space
loop until Convergence do
    for each w_i in w do
        w_i ← w_i − α ∂/∂w_i Loss(w).
```

→ $\alpha$ is stepsize usually called the learning rate when we are trying to minimize loss in a learning problem.

99% of time.

→ For univariate regression, loss fⁿ is a quadratic fⁿ, so partial derivation will be a linear fⁿ.

on one training example (x, y)

$$\frac{\partial}{\partial w_i} Loss(w) = \frac{\partial}{\partial w_i} (y - h_w(x))^2$$

$$= 2(y - h_w(x)) \times \frac{\partial}{\partial w_i} (y - h_w(x))$$

$$= 2(y - h_w(x)) \times \frac{\partial}{\partial w_i} (y - (w_1 x + w_0)) \quad -18.5$$

apply to both $w_0$ & $w_1$ we get

$$\frac{\partial}{\partial w_0} Loss(w) = -2(y - h_w(x));$$

$$\frac{\partial}{\partial w_1} Loss(w) = -2(y - h_w(x)) \times x.$$

In 18.4. folding the 2 into unspecified learning rate $\alpha$. we get.

$w_0 \leftarrow w_0 + \alpha(y - h_w(x))$; $w_1 \leftarrow w_1 + \alpha(y - h_w(x)) \times x$.

if $h_w(x) > y$ i.e., o/p of hypothesis is too large.

Reduce $w_0$ a bit & reduce $w_1$ if x was a +ve i/p.

but increase $w_1$ if x ~~loss~~ was a -ve i/p.

→ Derivative of Sum is Sum of derivatives.

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_w(x_j));$$

$$w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_w(x_j)) \times x_j.$$

→ These update constitute batch gradient descent learning rule for univariate linear regression.

→ Convergence to unique global min is guaranteed, but may be very slow.

→ Another possibility, called Stochastic gradient descent, where we consider only a single training pt at a time,

→ It can be used in an online setting.

## Multivariate linear Regression

→ each example $x_j$ is an $n$-element vector.

→ hypothesis space is set of $f^n$s of form

$$h_w(x_j) = w_0 + w_1 x_{j,1} + \cdots + w_n x_{j,n} = w_0 + \sum_i w_i x_{j,i}.$$

→ Matrix product of transpose of weights & i/p vector.

$$h_w(x_j) = w \cdot x_j = w^T x_j = \sum_i w_i x_{j,i}.$$

best vector of weights, $w^*$, minimizes squared-error loss over exam

$$w^* = \underset{w}{\operatorname{argmin}} \sum_j L_2(y_j, w \cdot x_j).$$

→ Multivariate linear regression is actually not much more complicated than univariate case, we covered.

→ Gradient descent will reach (unique) minimum of loss $f^n$, the update equation for each weight $w_i$ is

$$w_i \leftarrow w_i + \alpha \sum_j x_{j,i} (y_j - h_w(x_j)).$$

→ It is possible to solve for w that minimizes loss.

→ let Y be vector of o/ps for training examples, & X be the data matrix, i.e., the matrix of i/ps with one-n-dimensional example per row.

Sol^n,

$w^* = (X^T X)^{-1} X^T y$ minimizes squared error.

→ In multivariate linear regression in high-dimensional spaces if irrelevant dimension appears by chance to be useful, which results in Overfitting.

→ It is common to use regularization on multivariate linear f^n's to avoid overfitting.

→ With regularization we minimize total cost of a hypothesis, counting both empirical loss & complexity of hypothesis:

$$\text{Cost}(h) = \text{Emploss}(h) + \lambda \text{Complexity}(h).$$

$$\text{Complexity}(h_w) = L_q(w) = \sum_i |w_i|^q.$$

→ $q=1$, $L_1$ regularization, which minimizes the sum of absolute values.

→ $q=2$, $L_2$ regularization minimizes the sum of squares.

→ Regularization which we pick depends on specific problem, $L_1$ has important adv., it tends to produce a sparse model, it often sets many weights to zero, declaring corresponding attributes to be irrelevant.

→ Minimizing Loss(w) + λ Complexity(w) is equivalent to minimizing Loss(w). Subject to constraint that Complexity(w) ≤ C.

→ Diamond-shaped box represents the set of points w in two dimensional weight space that have $L_1$ complexity less than C, our soln will have to be somewhere inside this box.

→ Concentric ovals represent contours of loss fn. with min loss at center.

→ we have to find pt in the box i.e., closest to minimum.

→ for an arbitrary position of minimum & its contours, it will be common for corner of box to find its way closest to minimum, just b'coz corners are pointy.

→ fig 18.14(b) represents a circle rather than a diamond. ~~in~~ there is no reason for intersection to appear on one of the axes, thus $L_2$ does not tend to produce zero weights.

→ Result is that no. of examples required to find a good h is linear in no. of irrelevant features for $L_2$.

→ but only Logarithmic with $L_1$ regularization. Empirical evidence on many problems supports this analysis.

# Linear Classifiers with a hard threshold

→ Linear f^n's can be used to do classification as well as regression.

→ Example fig 18.15(a) shows data points of two classes earthquakes & underground explosions.

→ Each point is defined by 2 i/p values $x_1$ & $x_2$, that refer to body & surface wave magnitudes computed from seismic signal.

→ Given these training data, the task of classification is to learn a hypothesis h that will take new $(x_1, x_2)$ pts & return either 0 for earthquakes or 1 for explosions.

→ Decision boundary is a line that separates the two classes.

→ In fig (a) decision boundary is a straight line.

→ A linear decision boundary is called a linear separator & data that admit such a separator are called linearly separable.

$$x_2 = 1.7x_1 - 4.9 \quad \text{or} \quad -4.9 + 1.7x_1 - x_2 = 0.$$

→ Explosions, which we want to classify with value 1, are to right of this line with higher values of $x_1$ & lower values of $x_2$,

→ So they are points for which $-4.9 + 1.7x_1 - x_2 > 0$, while earthquakes have $-4.9 + 1.7x_1 - x_2 < 0$.

→ with dummy i/p $x_0 = 1$. we can write classification hypothesis as

$$h_w(x) = 1 \text{ if } w \cdot x \geq 0 \text{ and } 0 \text{ otherwise.}$$

→ Alternatively, we can think of h as result of passing the linear $f^n$ $w \cdot x$ through a threshold $f^n$.

$h_w(x) = \text{Threshold}(w \cdot x)$ where $\text{Threshold}(z) = 1$ if $z \geq 0$ & 0 otherwise.

→ Now Hypothesis $h_w(x)$ has a well defined mathematical form, we can think about choosing the weights $w$ to minimize the loss.

→ In 18.6.1 & 18.6.2, we did this both by setting the gradient to zero & solving for weights.

→ Here, we can't do either of those because the gradient is '0' almost everywhere in weight space except at those pts where $w \cdot x = 0$, & at those pts the gradient is undefined.

→ Linear Separator classifies the data perfectly - provided the data are linearly separable.

Ex:- Single example $(x, y)$ we have

$$w_i \leftarrow w_i + \alpha (y - h_w(x)) \times x_i$$

the update rule for linear regression. This rule is called perceptron learning rule.

→ we are considering 0/1 classification problem.

Both true value $y$ & hypothesis o/p $h_w(x)$ are either 0 or 1.

So there are 3 possibilities.

- If o/p is correct i.e., $y = h_w(x)$, then the weights are not changed.

- If $y$ is 1 but $h_w(x)$ is 0, then $w_i$ is increased when corresponding i/p $x_i$ is +ve & decreased when $x_i$ is -ve.

  This makes sense, b'coz we want to make $W.x$ bigger so that $h_w(x)$ o/ps a 1.

- If $y$ is 0 but $h_w(x)$ is 1, then $w_i$ is decreased when corresponding i/p $x_i$ is +ve & increased when $x_i$ is -ve.

  b'coz we want to make $W.x$ smaller so that $h_w(x)$ o/ps a 0.

→ Fig 18.16(a) shows a training curve for learing rule applied to earthquake/explosion data.

→ Training Curve measures the classifier performance on a fixed training set as learning process proceeds on that same training set.

→ The curve shows update rule converging to a zero error linear Separator.

→ The "convergence" process is pretty.

→ This particular run takes 657 Steps to Converge, for a data set with 63 examples, so each example is presented roughly 10 times on average.

→ The perceptron learning rule converges to a perfect linear Separator when data pts are linearly Separable, what if they are not

→ This is too common in real world.

## Linear Classification with Logistic Regression

→ Passing the o/p of a linear $f^n$ through threshold $f^n$ creates a linear classifier, yet hard nature of threshold cause some problems.

→ The hypothesis $h_w(x)$ is not differentiable & is in fact a discontinuous $f^n$ of its i/p & its weights.

→ This makes learning with perceptron rule a very unpredictable adventure.

→ Linear classifier always announces a completely confident prediction of 1 or 0, even for examples that are very close to boundary., we need more graded predictions.

→ All these can be resolved to a large extent by softening threshold $f^n$ - approximating hard threshold with a continuous, differentiable $f^n$.

Logistic $f^n$ $\text{Logistic}(z) = \frac{1}{1+e^{-z}}$. has convenient mathematical properties.

→ Eg. The $f^n$ 18.17(b) with logistic $f^n$ replacing threshold $f^n$ we now have

$$h_w(x) = \text{Logistic}(w.x) = \frac{1}{1+e^{-w.x}}.$$

→ example of for two i/p earthquake/explosion problem shown in 18.17(c).

→ Notice that o/p being a no. b/w 0 & 1, can be interpreted as a probability of belonging to the class labeled 1.

→ The process of fitting the weights of this model to minimize loss on a data set is called logistic regression.

→ There is no easy closed-form Soln to find optimal value of W with this model, gradient descent is straightforward.

→ B'coz our hypotheses no longer O/p just 0 or 1, we will use L2 loss fn

→ g for logistic fn, g' its derivative.

→ For a single example (x, y) the derivation of gradient is same as for linear regression up to the point where actual form of h is inserted.

we need chain rule.

$$\partial g(f(x))/\partial x = g'((f(x))\partial f(x)/\partial x).$$

$$\frac{\partial}{\partial w_i} Loss(w) = \frac{\partial}{\partial w_i}(y - h_w(x))^2$$

$$= 2(y - h_w(x)) \times \frac{\partial}{\partial w_i}(y - h_w(x))$$

$$= -2(y - h_w(x) \times g'(w \cdot x) \times \frac{\partial}{\partial w_i} w \cdot x$$

$$= -2(y - h_w(x) \times g'(w \cdot x) \times x_i$$

g' of logistic fn satisfies $g'(z) = g(z)(1 - g(z))$.

$$g'(w \cdot x) = g(w \cdot x)(1 - g(w \cdot x)) = h_w(x)(1 - h_w(x))$$

So, weight update for minimizing loss is

$$w_i \leftarrow w_i + \alpha (y - h_w(x)) \times h_w(x) \cdot (1 - h_w(x)) \times x_i.$$