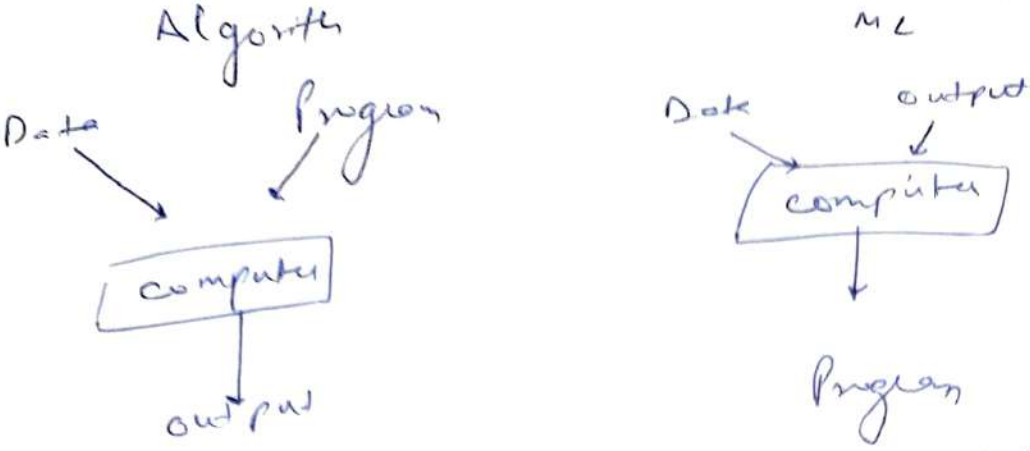
Introduction Machine Learning

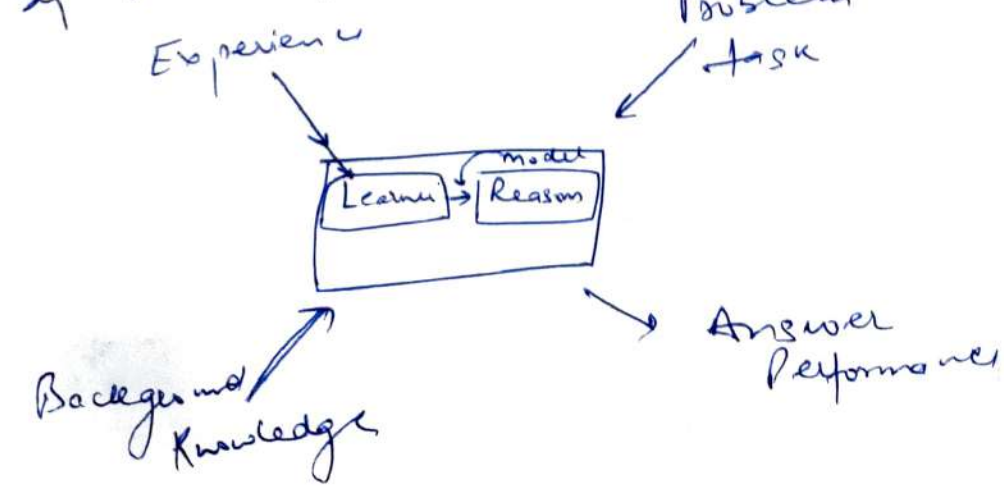


Learning: is the ability to improve one behaviour with experience

ML explor algorithm

- Learn / build model for data
- model used for prediction, decision making or solving tasks

Mitchell A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance on tasks in T as measured by P improves with experience E



Many domain and application

Medicines

- Diagnosis a disease
- Data mine historical medical ~~report~~ record to learn which future patients will respond best to which treatment

Vision —, Robot control, NLP, speech recognition, Machine translation, financial

(1) Choose the training experience features

(2) Choose the target function (that to be learned.

(3) Choose how to represent the target function (class of function)

(4) Choose a learning algorithm to infer the target function

→ Rich representation more difficult to learn

→ hypothesis language

Chapter 3 Different type of Learning

Supervised → Semisupervised

X , Y

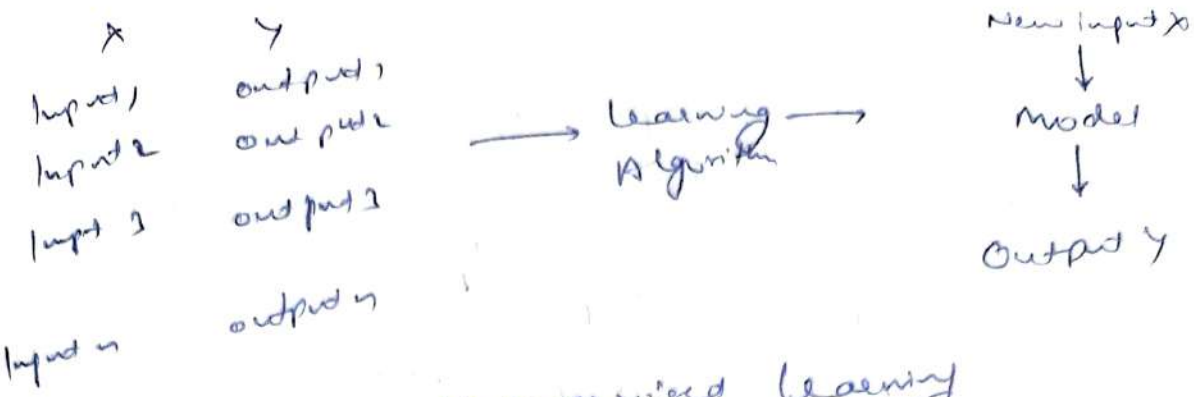
inp op.

Unsupervised

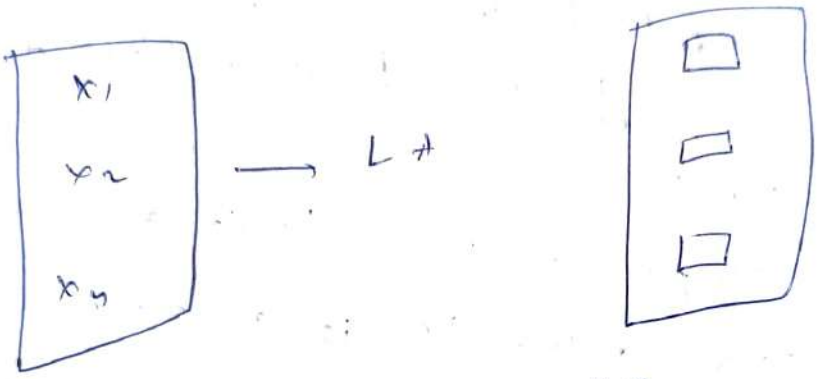
X

Reinforcement Learning

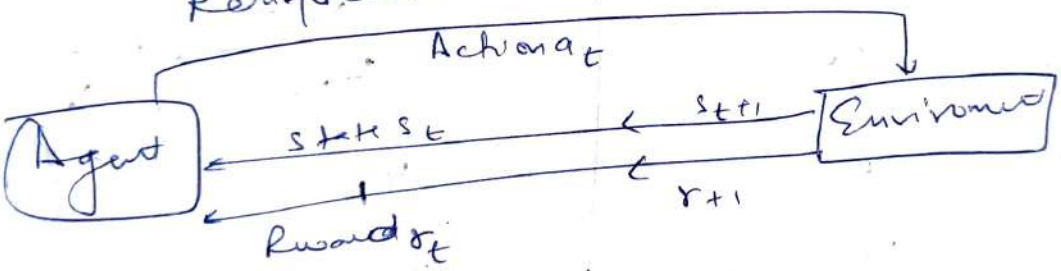
Supervised Learning



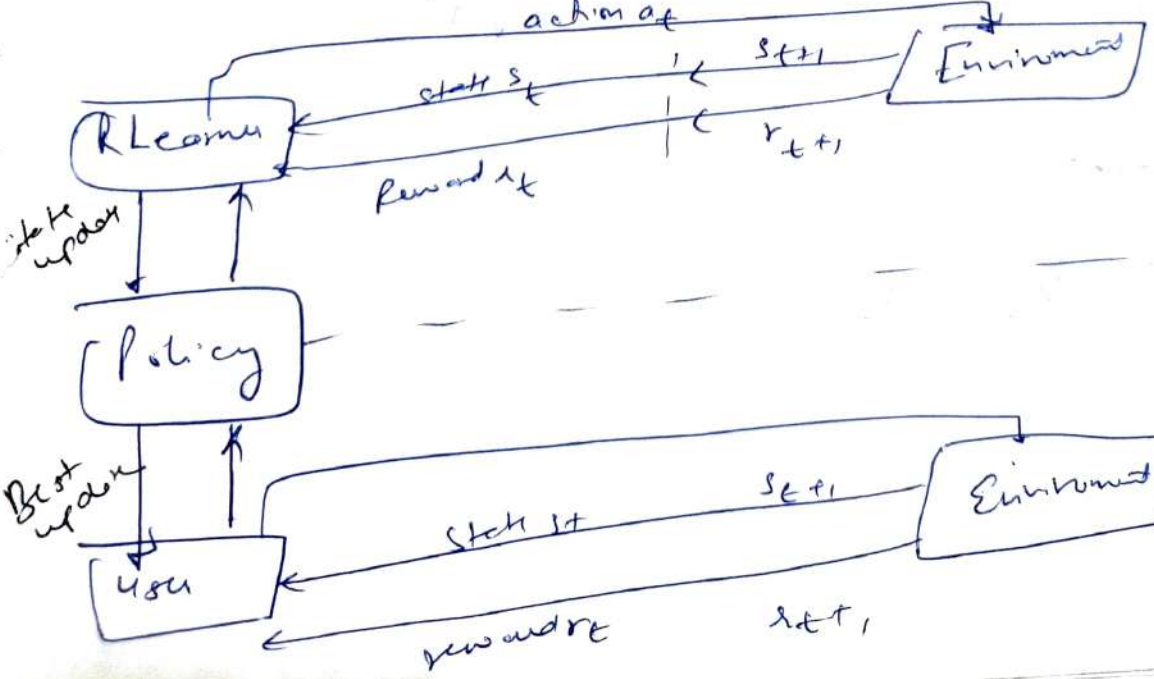
Unsupervised Learning



Reinforcement Learning



Reinforcement Learning



Supervised learning

Given
- a set of of input features $x_1, \ldots, x_n$
- A target feature y
- a set of training examples where the values for the input features and the target features are given for each example
- a new example, where only the values for the input features are given

Predict the values for the target features for the new example
- Classification when y is discrete
- regression when y is continuous

y → discrete → classification
y → continuous → regression

	x_1	x_2	...	x_n	y
I_1	a_1	a_2		a_n	y_1
I_2	b_1	b_2		b_n	y_2
I_3	c_1	c_2		c_n	y_3

Test instance

$z_1, z_2, z_3 \ldots z_n$

Classification

Example: Credit scoring

Differentiating b/w low-risk and high-risk customers from their income and saving

— Discriminant: If income $> \theta_1$ and saving $> \theta_2$ then low-risk else high-risk

Regression

Example: Price of a used car

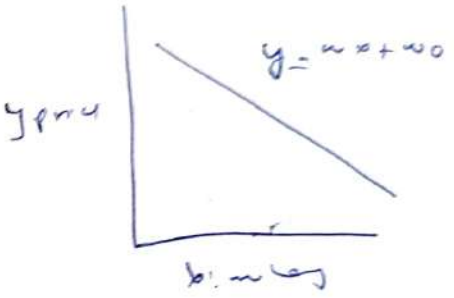
x : car attributes

y : Price

$$y = g(x, \theta)$$

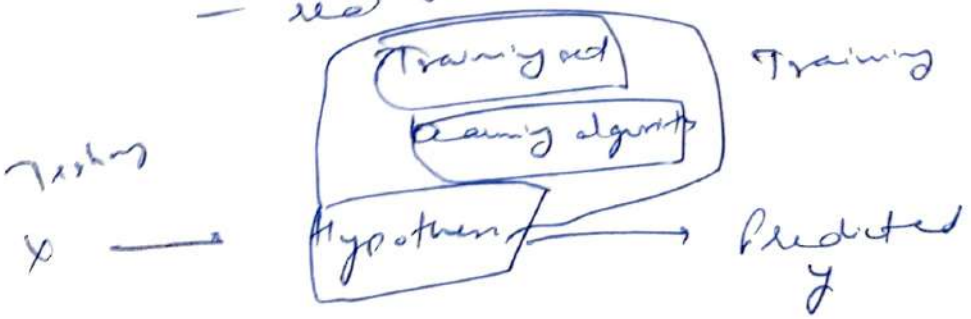
$g()$ = model

θ Parameters



Features

- categorical
- ordinal
- integer valued
- real valued



classification learning

- Task T
  - Input: a set of instances $d_1, \ldots d_n$
    - an instance has a set of features
    - we can represent an instance as a vector $d = \langle x_1, \ldots, x_m \rangle$
  - output: a set of predictions $\hat{y}_1, \ldots \hat{y}_n$
    - one of a fixed list of a constant values:
      - $\{+1, -1\}$ or {cancer, healthy}
- Performance metric P:
- Experience E:

first → Microsoft corporation ← last

acquired XYZ

↑

first

- a set of labeled examples (x, y) where y is the true label for x
- ideally, examples should be sampled iid from some fixed distribution D

Representation

1. Decision tree
2. Linear function
3. Multivariate Linear function
4. Single layer perceptron
5. multilayer neural network

Hypothesis space :-

One way to think about a supervised learning machine is as a device that explores a "hypothesis space"

- Each setting of the parameters in the machine is a different hypothesis about the function that maps input vectors to output vectors

Chapter 4 Hypothesis Space and Inductive Bias

Inductive Learning or Prediction

Given examples $(\vec{x}, y)$

$(\vec{x}, f(\vec{x}))$

Classification $f(\vec{x})$ is discrete

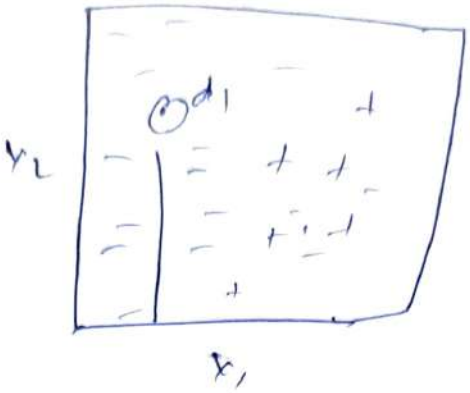
Regression $f(\vec{x})$ is continuous

Probability estimation = probability of $\vec{x}$

features properties that describe each instances

feature vector

Language



hypothesis space

H $\quad h \in H$

output of a learning alg

Examples (x, y)

Training Data set of examples

Instance Space

Concept C $\quad C \subseteq X$

Target function $f : x \to y$

find
hypothesis $h \in H$ that approximates f

H restrict hypothesis
by defining bias
- Constraints
- Preferences

Input $S \subseteq X$

output $h \in H$

x_1, x_2, x_3 ...	Possible instances $2^4 / 2^H$ Honorary Boolean fns possible. $2^{16} / 2^{2^4}$

- Hypothesis language reflect an inductive bias of the learner

Hypothesis language

Occam's Razor

A classical example of Inductive Bias,

. A simplest consistent hypothesis about the target function is actually the best

Underfitting and overfitting

- Underfitting: model is too "simple" to represent all the relevant class characteristics
  - High bias and low variance
  - High training error and high test error
- Overfitting: model is too "complex" and fits irrelevant characteristic (noise) in the data
  - Low bias and high variance
  - low training error and high test error

Chapter 4 Evaluation and cross validation

- Evaluating the performance of learning is important because
  - learning systems are usually designed to predict the class of "future" unlabeled data points.
- Typical choices for performance evaluation
  - Error
  - Accuracy
  - Precision/Recall
- Typical choices for sampling methods
  - Train/Test sets
  - k-fold cross validation

H, s

h

Experimental Evaluation

Error Accuracy Precision/Recall	Evaluation on Sample Train set Test set Cross validation

Evaluating Prediction

- Suppose we want to make a prediction of the value for a target feature on example x
  - y is the observed value of target feature on example x
  - $\hat{y}$ is the predicted value of target feature on example x.
- How is the error measured?

$\hat{y} = h(x)$ ① Absolute error: $\frac{1}{n}\sum |h(x) - y|$

$\hat{y}$ y ② Sum of sq. error $\frac{1}{n}\sum_{i=1}^{n} (h(x) - y)^2$

③ Number of misclassifications

$$\frac{1}{n}\sum_{i=1}^{n} \delta(h(x), y)$$

Hypothesis \ True class	Pos	Neg
Pos	TP	FP
Neg	FN	TN
	P	N

$$\text{Accuracy} = \frac{TP + TN}{P + N}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{P}$$

Sample Error and True Error

- The sample error of hypothesis h with respect to target function c and data sample S is:

$$error_S(h) = 1/n \sum_{x \in S} \delta(f(x), c(x))$$

- The true error (or $error_D(h)$) of hypothesis h with respect to target function c and distribution D, is the probability that h will misclassify an instance drawn at random according to D

$$error_D(h) = Pr_{x \in D}[f(x) \neq c(x)]$$

Difficulties in evaluating hypotheses with limited space

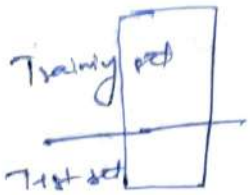
- Bias in the estimate :- The sample error is a poor estimator of the error
  - ==> test the hypothesis on an independent test set
- We divide the example into:
  - Training examples that are used to train the learner
  - Test examples that are used to evaluate the learner

Variance is the estimate : The smaller the test set, the greater the expected variance.

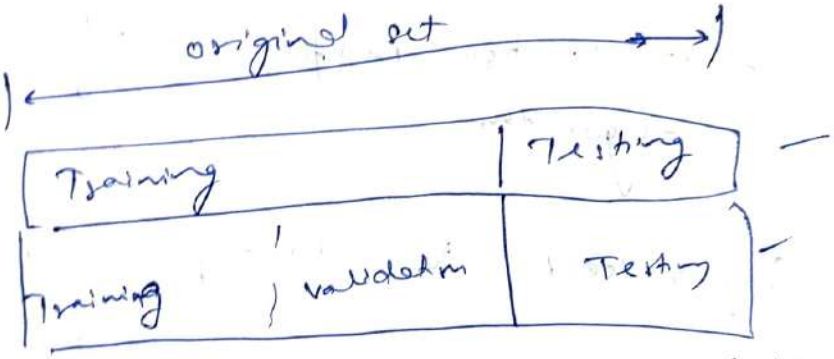
Split the examples:

Training set – to train the learner

Test set – to test the learner

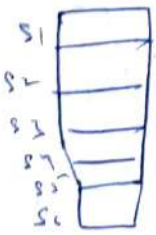


Validation set



Validation fails to use all the available data

k fold cross validation –



Round i use S_i for testing –
$S - S_i$ for training

Trade off

In ML there is always a trade off b/w
– complex hypothesis that fit the training data well

Simpler hypothesis ... generalize ...

- As the amount of training data ... generalization error ...

Chapter 1

Tutorial 1

- Supervised Vs Unsupervised learning
- Categorical Vs Continuous features
- Regression Vs Classification
- Bias Vs Variance
- Generalization Performance of a learning algorithm

Supervised Vs Unsupervised learning

Supervised

Training sample

(Input, Target)

The description of Picture

Target

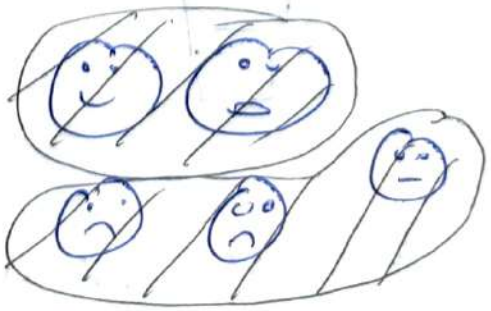
Happy/Sad

Unsupervised

- You don't have targets values

Training example

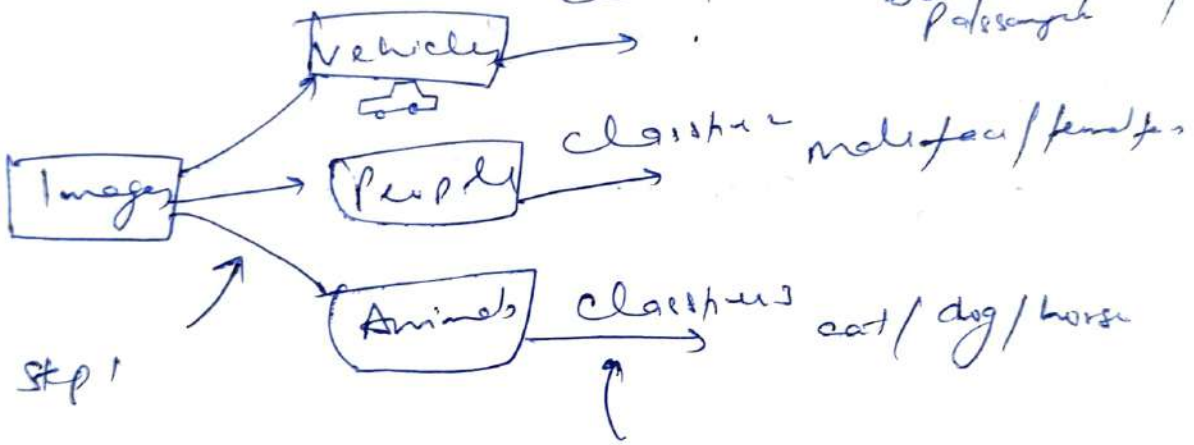
(Input)



Supervised Vs Unsupervised learning

Q:- Suppose I have been given 1 million images with their ground truth labels. I looked at the dataset and found that the images all fall in a wide range of categories. There are vehicles, plants, animals, human faces etc.

I decided to first go for clustering the images into groups of similar looking objects. Then I would train one classifier per cluster.



Class of learning?
Unsupervised

Class of learning?
Supervised

Kind of features

Categorical Vs Continuous

Categorical features

↳ finite no of categories or classes

eg: Gender, age-group

continuous features

Infinite no. of values

eg: age, height, weight, price, etc

Types of supervised learning

Regression & classification

Type of output variable

Classification

output / Target variable is categorical / Discrete

eg. Given images of 5 animals → Predict the animal

Given image of a tumor → Predict → cancerous / not cancerous

Regression

output / Target variable is continuous

eg Given certain features of a car → Predict → Price of car

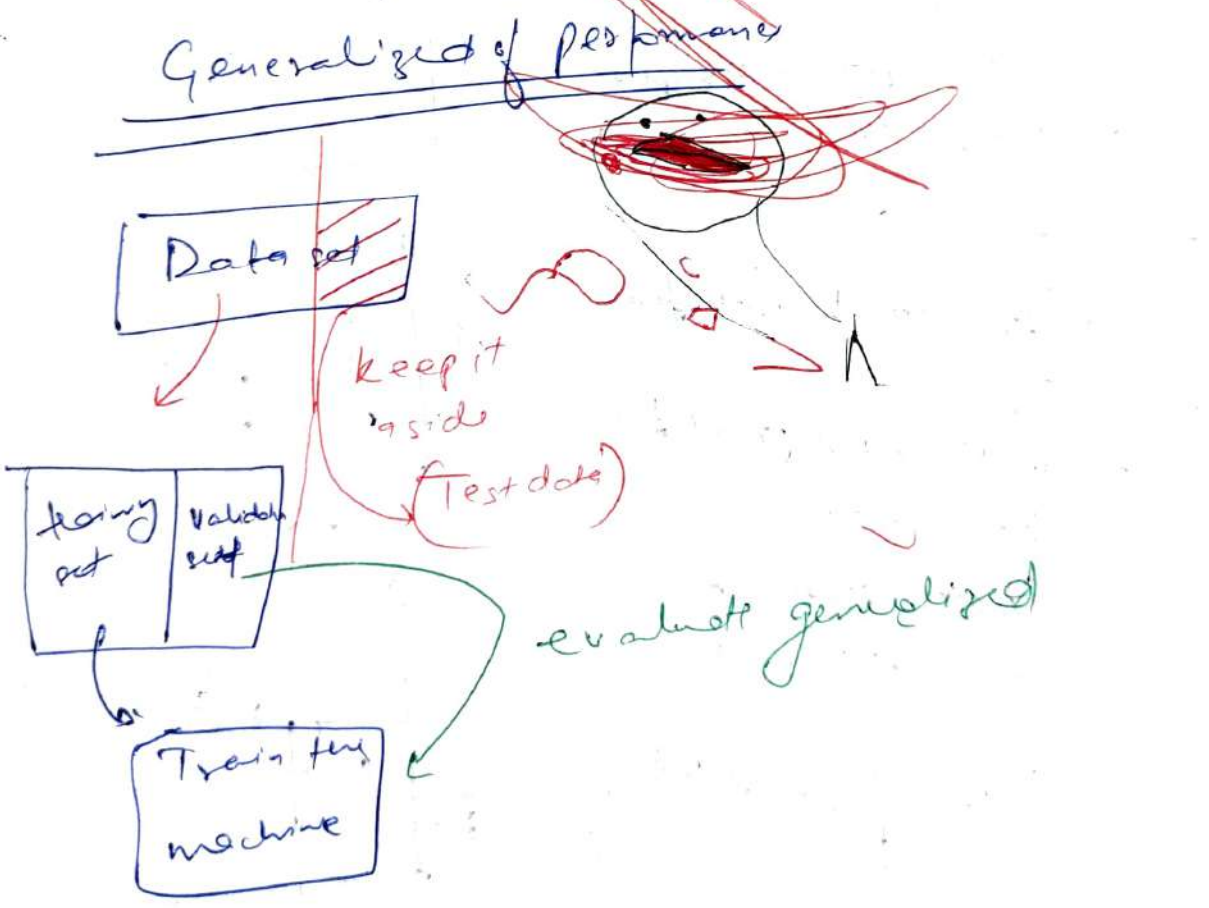
Predict age / weight / height of person → continuous

Bias Vs Variance

Bias: erroneous Assumption in the learning algorithm

Variance sensitivity of model towards Noise rather than important features of the relationship b/w input and output

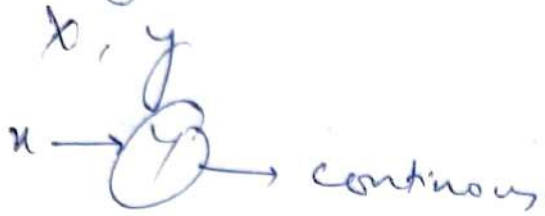
	No of features	No of parameters	No of training eg
Bias	↓ res	↓ res	Remains the same
Variance	↑ res.	↑ res	↓ res



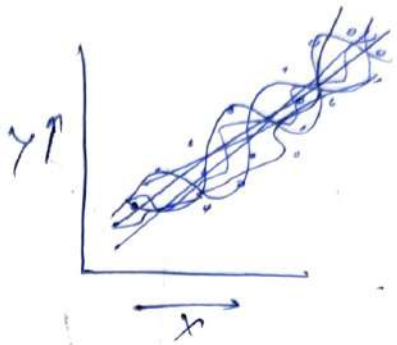
Chapter 5

Linear Regression

Regression is a supervised learning problem

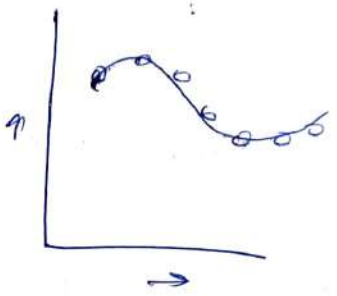


Simple regression

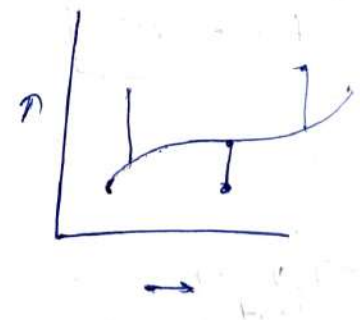


Linear regression

- Given an input x compute an output y

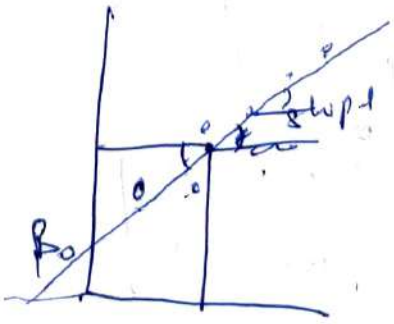


- Eg;
  - Predict height from age
  - Predict house price from house area
  - Board distance from wall from sensors.



$$Y = \beta_0 + \beta_1 X + \varepsilon$$

- Y- intercept
- slope
- Random error



$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \dots + \beta_p x^p + \varepsilon$

$E(y|x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \dots + \beta_p x^p$

Least sq. line $\hat{\beta}_0 + \hat{\beta}_1 x + \hat{\beta}_2 x^2 + \dots + \hat{\beta}_p x^p$

minimize SSE

β_0 is the intercept, β_j is the slope for j^{th} variable x_j

Assumption $y = \beta_0 + \beta_1 x + \varepsilon$

$E(\varepsilon_i) = 0$

$\sigma(\varepsilon_i) = \sigma_\varepsilon$

Errors are independent

ε_i are normally distributed

$d_1 : y_1 = \beta_0 + \beta_1 x_1 + \varepsilon_1$

$d_2 \; y_2 = \beta_0 + \beta_1 x_2 + \varepsilon_2$

$\vdots$

d_n

$\text{Loss} = \underset{\theta_0, \theta_1}{\text{argmin}} \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

LSC func

The regression line the least sq.

regression line is the unique line such that the sum of the squared vertical (d) distances b/w the data pt and the line is the smallest possible

$d_i = (x_i, y_i) \quad \sum_{d \in D} \left(y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_1) \right)^2$

How to learn β_0 , β_1 (parameters)

- from 2d problem

$$y = \beta_0 + \beta_1 x$$

- To find the value for the coefficients which minimize the objective function we take the partial derivates of the objective function (SSE) with respect to the coefficient. Set these to 0, and solve

$$\beta_1 = \frac{n \sum xy - \sum x \sum y}{n \sum x^2 - (\sum x)^2}$$

$$\beta_0 = \frac{\sum y - \beta_1 \sum x}{n}$$

Multiple Linear Regression

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

$$h(x) = \sum_{i=0}^{n} \beta_i x_i$$

- There is a closed form which requires matrix inversion etc
- There are iterative techniques to find weights

- Delta rule (also called LMS method) which will update towards the objective of minimizing the SSE

Linear Regression

$$h(x) = \sum_{i=0}^{n} \beta_i x_i$$

To learn the parameters $\theta(\beta_i)$?

- Make $h(x)$ close to y, for the available training examples
- Define a cost function $J(\theta)$

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h(x)^{(i)} - y^{(i)})^2$$

Find θ that minimizes $J(\theta)$

LMS Algorithm

- Start a search algorithm (e.g. gradient descent algorithm) with initial guess of θ
- Repeatedly update θ to make $J(\theta)$ small, until it converges to minimum

$$\beta_j = \beta_j - \alpha \frac{\partial}{\partial \beta_j} J(\theta)$$

- J is a convex quadratic function, so has a single global minimum. gradient descent eventually at the global minimum

- At each iteration of this algorithm, taking a step in the direction of steepest descent (-ve direction of gradient)

$$\frac{\partial}{\partial \theta} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2} (h(x) - y)^2$$

LMS update Rule

- if you have only one training example (x, y)

$$\frac{\partial}{\partial \theta} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2} (h(x) - y)^2$$

$$= 2 \cdot \frac{1}{2} (h(x) - y) \frac{\partial}{\partial \theta_j} (h(x) - y)$$

$$= (h(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right)$$

$$= (h(x) - y) x_j$$

- for a single training eg., this gives the update rule

$$\beta_j = \beta_j + \alpha (y^{(i)} - h(x^{(i)})) x_j^{(i)}$$

$$h(x) = \sum_{i=0}^{n} \beta_i x_i$$

To learn the parameters $\theta(\beta_i)$?

- Make $h(x)$ close to y, for the available training examples
- Define a cost $J(\theta)$ /

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h(x)^{(i)} - y^{(i)})^2$$

- Find θ that minimizes $J(\theta)$ /

LMS update Rule

- If you have only one training example (x, y)

$$\frac{\partial}{\partial \theta} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2} (h(x) - y)^2$$

$$= 2 \cdot \frac{1}{2} (h(x) - y) \frac{\partial}{\partial \theta_j} (h(x) - y)$$

$$= (h(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right)$$

$$= (h(x) - y) x_j$$

- For a single training eg, this gives the update rule

$$\beta_j' = \beta_j + \alpha (y^{(i)} - h(x^{(i)})) x_j^{(i)}$$

m training examples

Repeat until convergence {

$$\theta_j = \theta_j + \alpha \sum_{i=1}^{m} (y^{(i)} - h(x^{(i)}))x_j^{(i)}$$

}

Batch Gradient Descent: Looks at every example on each step

Stochastic gradient descent

- Repeatedly runs through the training set
- whenever a training pt is encountered update the parameters according to the gradient of the error with respect to that training example only

Repeat {

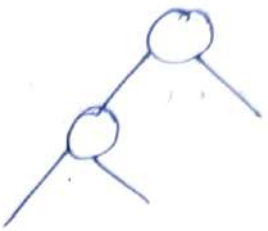
for $I = 1$ to m do

$$\theta_j := \theta_j + \alpha (y^{(i)} - h(x^{(i)}))x_j^{(i)}$$

for every j | end for until convergence

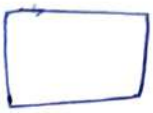
~~Decision tree~~ Decision trees

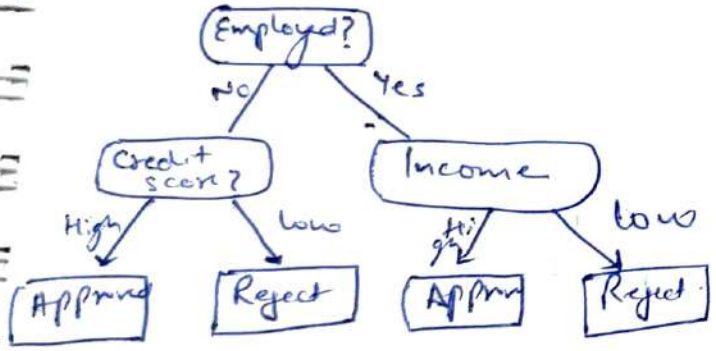
Decision Tree or classifier

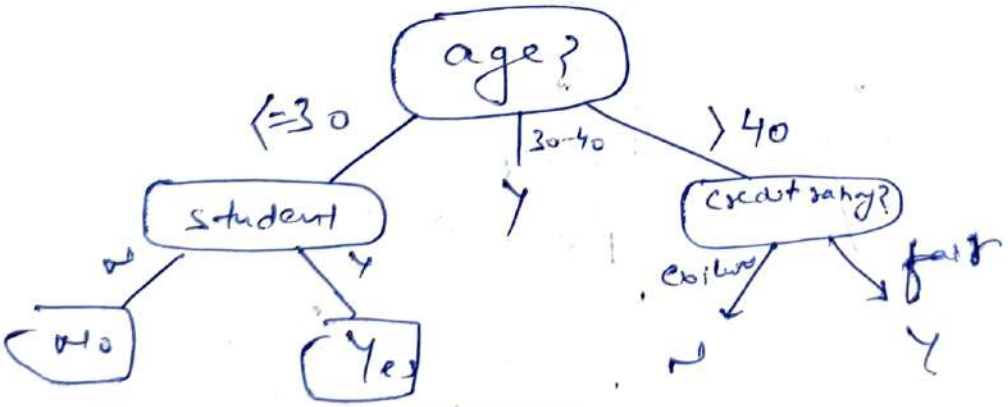


tree structure

- Decision Nodes → 
- Leaf Nodes

 classification value





Issues

- Given some training examples, what decision tree should be generated
- One proposal: prefer the smallest tree that consistent with the data (Bias)

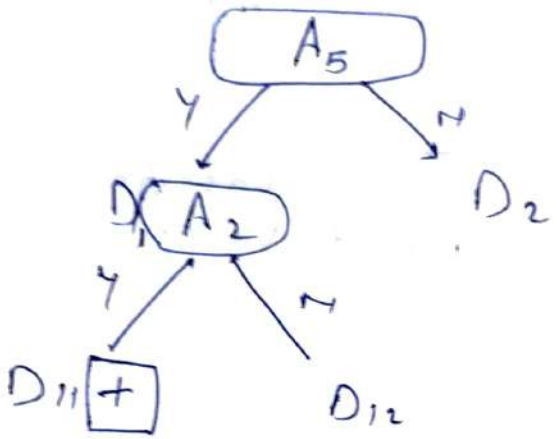
— Possible method.

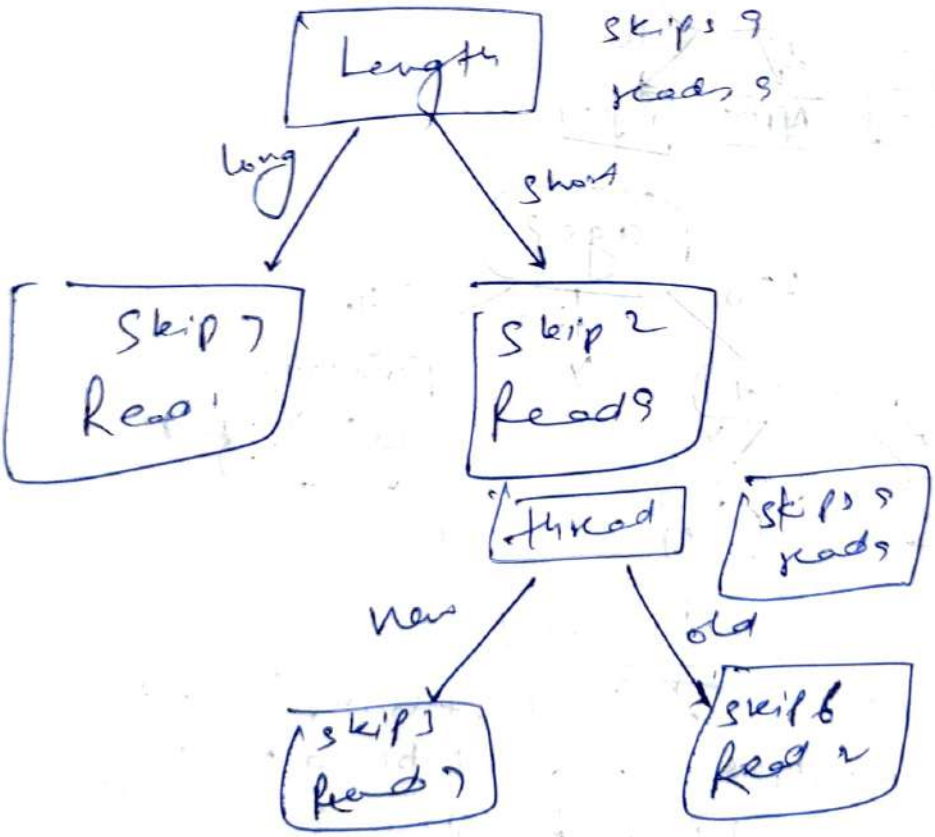
- Search the space of decision trees for the smallest decision tree that fits the data.

Prefer smaller tree

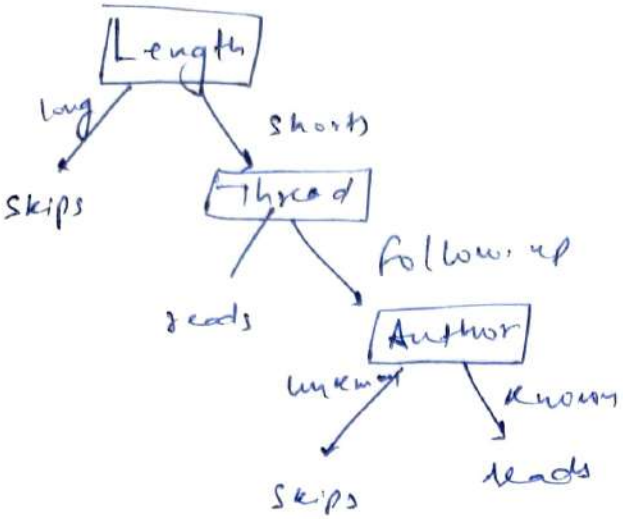
- low depth
- small no. of nodes

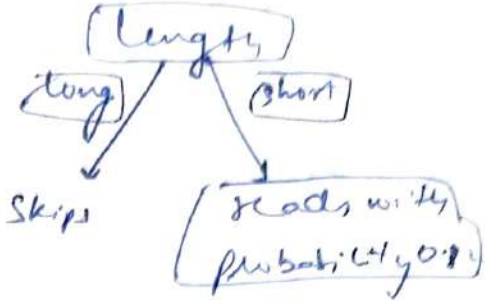
Possible split





Two examples DTs





Decision Tree for PlayTennis

- Attributes and their values
  - outlook: Sunny, Overcast, Rain
  - Humidity: High, Normal
  - Wind: Strong, weak
  - Temperature: Hot, Mild, Cool
- Target concept - Play tennis: Yes, No.

Searching for a good tree

- The space of decision tree is too big for systematic search.
- stop and return.
  - return the value for the target feature or
  - a distribution over target feature value
- Choose a test (eg. an input feature) to split on
  - for each value of the test, build a subtree for those examples with this value for the test

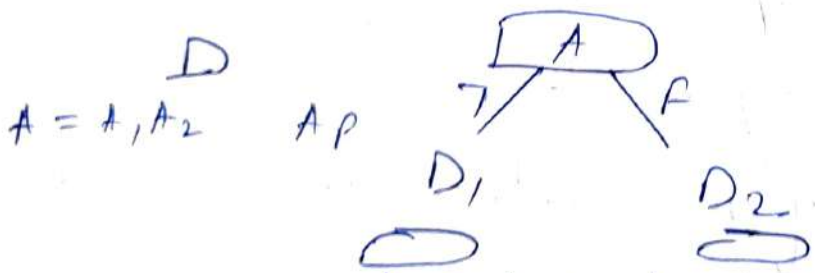
Top-Down Induction of Decision Tree ID3

1. $A \leftarrow$ the "best" decision attribute for next node
2. Assign A as decision attribute for node
3. For each value of A, create new descendant
4. Sort training examples to leaf node according to the attribute value of the branch
5. If all training examples are perfectly classified (same value of target attribute) stop, else iterate over leaf nodes.

(1) which node to proceed with?

(2) when to stop.

Chapter 7: Learning Decision tree



Top Down Induction of Decision Tree ID3

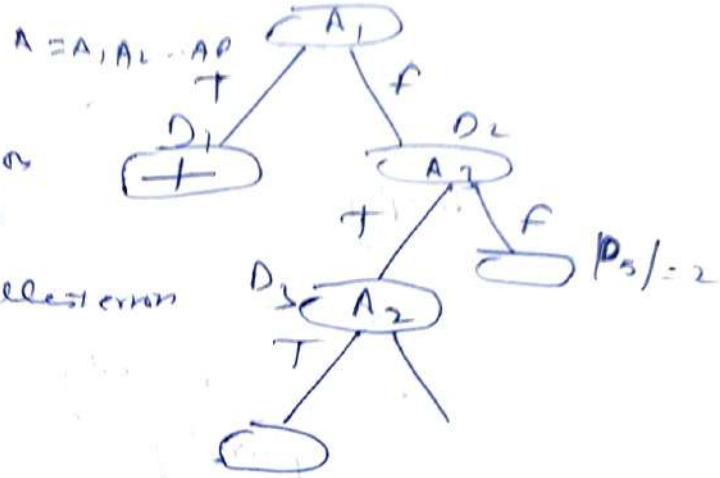
Choices

- when to stop
  - no more input features
  - all examples are classified the same
  - too few examples to make an informative split.

- which test to split on
  - split gives smallest error
  - with multi valued features
    - split on all values or
    - split values into half

① When to stop

② Which attribute to split on



1. split gives smallest error

which Attribute is best?

A_1 = ? 29+, 35−

T: 21+, 5−

F: 8+, 30−

A_2 = ? 29+, 35−

T: 18+, 33−

F: (11+, 2−)

Principled Criterion

- selection of an attribute to test at each node, choosing the most useful attribute for classifying examples
- Information gain

- measures how well a given attribute separates the training examples according to their target classification
- This measure is used to select among the candidate attributes at each step while growing the tree.
- Gain is measure of how much we can reduce uncertainty (value lies b/w 0,1)

Entropy

- A measure for
  - uncertainty
  - purity
  - information content
- Information theory: optimal length code assigns $(-\log_2 p)$ bits to message having probability p.
- S is a sample of training examples
  - p_+ is the proportion of +ve examples in S
  - p_- is the proportion of -ve examples in S
- Entropy of S: average of optimal no. of bits to encode information about certainty / uncertainty about S

Entropy (S) = $p_+(-\log_2 p_+) + p_-(-\log_2 p_-) = -p_+ \log_2 p_+ - p_- \log_2 p_-$

entropy → measure of purity

p +ve eg
-p -ve eg

$-\log_2 p$ bit

$p_+(-\log_2 p_+) + p_-(-\log_2 p_-)$

$= -p_+ \log p_+ - p_+ \log_2 p_-$

- S is a sample of training examples
- p_+ is the proportion of +ve examples
- p_- is the proportion of -ve examples
- Entropy measures the impurity of S

Entropy (S) = $-p_+ \log_2 p_+ - p_- \log_2 p_-$

- The entropy is 0 if the outcome is certain
- The entropy is maximum if we have no knowledge of the system (or any outcome is equally possible)

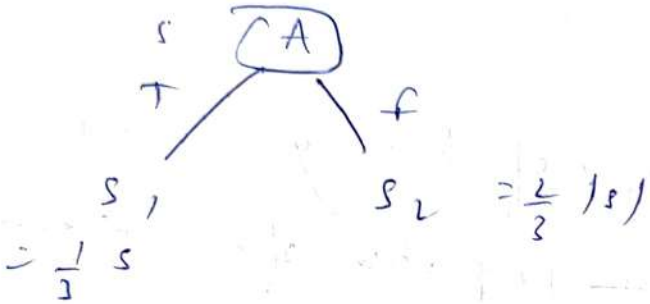
Information Gain

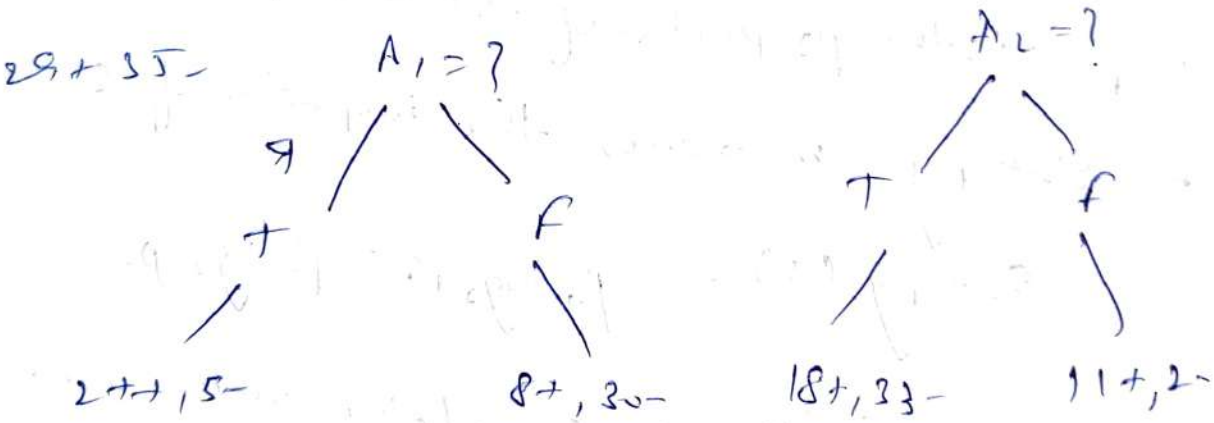
Gain (S, A): expected reduction in entropy due to partitioning S on attribute A

$$Gain(S, A) = Entropy(S) - \sum_{v \in values(A)} |S_v| / |S| \, Entropy(S_v)$$

$$Entropy([29+, 35-]) = -29/64 \log_2 29/64 - 35/64 \log_2 35/64$$

$$= 0.99$$





- Entropy ([21+, 5-]) = 0.71

Entropy ([8+, 30-]) = 0.74

Gain (S, A1) = Entropy (S)

$$- 26/64 * Entropy([21+, 5-])$$
$$- 38/64 * Entropy([8+, 30-])$$
$$= 0.27$$

Entropy ([18+, 33-]) = 0.94

Entropy ([8+, 30-]) = 0.62

Gain (S, A_2) = Entropy (S)

- 51/64 * Entropy ([18+, 33-])

- 13/64 * Entropy ([11+, 2-])

= 0.12

ID3 Algorithm

Splitting Rule: GINI Index

- GINI Index
  - Measure of node impurity

$$GINI_{node}(Node) = 1 - \sum_{c \in classes} [p(c)]^2$$

$$GINI_{split}(A) = \sum_{v \in Values(A)} \frac{|S_v|}{|S|} GINI(N_v)$$

- Splitting Based on continuous Attribute

- Continuous Attribute - Binary split
  - For continuous attribute
  - Partition the continuous value of attribute A into a discrete set of intervals
  - Create a new boolean attribute A_c, looking for a threshold c,

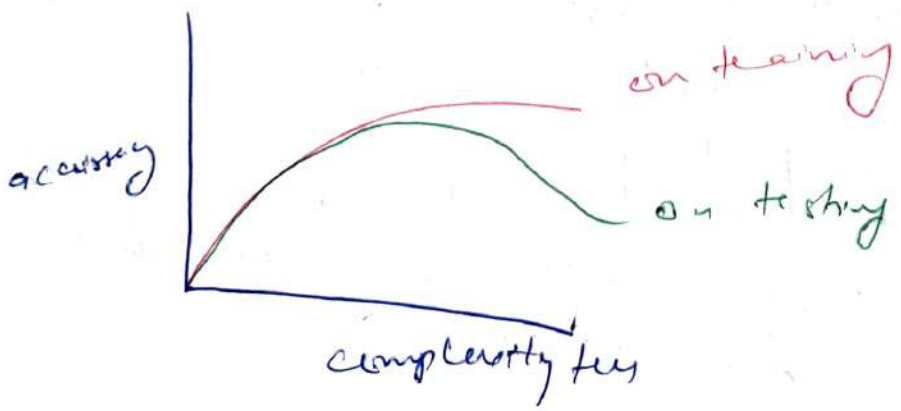
$A_c = \begin{cases} \text{true} & \text{if } A_c < c \\ \text{false} & \text{otherwise} \end{cases}$

How to choose c?
- consider all possible splits and find the best cut

Lecture 8 Overfitting

A hypothesis h is said to overfit the training data if there is another hyp. h' such that h' has more error than h on training data, h' has less error than h on test data

— Learning a tree that classifies the training data perfectly may not lead to the tree with the best generalization performance
— There may be noise in the training data
- may be based on insufficient data



overfitting due to insufficient examples

Notes on Overfitting

- Overfitting results in decision trees that are more complex than necessary
- Training error no longer provides a good estimate of how well the tree will perform on previously unseen records.

Avoid Overfitting

- How can we avoid overfitting a decision tree?
  - Prepruning: stop growing when data split not statistically significant
  - Postpruning: grow full tree then remove nodes
    error(T), error(T − sub)
- Methods for evaluating subtree to prune
  - Minimum description length (MDL)

minimize: size(tree) + size(misclassification(tree))

  - cross validation

Pre-Pruning (Early stopping)

- Evaluate splits before installing them
- Don't install splits that don't look worthwhile
  - when no worthwhile split to install, done

- Typical stopping conditions for a node:
  - stop if all instances belong to the same class
  - stop if all the attribute values are the same
- more restrictive conditions
  - stop if no. of instances is less than some user specified threshold
  - stop if class distribution of instances are independent of the available features (e.g. using χ^2 test)
  - stop if expanding the current node does not improve impurity measures (e.g. Gini or information gain)

Reduced error Pruning

- A Post-pruning, cross validation approach
  - Partition training data into grow set and validation set.
  - Build a complete tree for the grow data
  - until accuracy on validation set ↓ do:
    for each non leaf node in the tree.
    Temporarily prune the tree below; replace it by majority vote

Test the accuracy of the hypothesis on the validation set

Permanently Prune the node with the greatest ↑ in accuracy on validation set. Test

— Problem: Uses less data to construct the tree sometimes done at the rules level

— General strategy: overfit and simplify

Triple Trade off

— There is a trade-off b/w three factors.
  - complexity of H, c(H)
  - Training set size, N
  - Generalization error, E on new data

— As N ↑ ⇒ E ↓
— As c(H) ↑ ⇒ first E ↓ and then E ↑
— As c(H) ↑ ⇒ the training error ↓ for sometime and then stays constant (frequently at 0)

Dealing with overfitting

— Use more data
— Use a tuning set
— Regularization
— Be a Bayesian

Regularization In a Linear regression model overfitting is characterized by large weights

Penalize large weights in Linear Regression

— Introduce a penalty term in the loss function.

$$E(\vec{w}) = \frac{1}{2} \sum_{n=0}^{N-1} \{t_n - y(x_n, \vec{w})\}^2$$

Regularized Regression

1. (L2 - Regularization or Ridge Regression)

$$E(\vec{w}) = \frac{1}{2} \sum_{n=0}^{N-1} (t_n - y(x_n, \vec{w}))^2 + \frac{\lambda}{2} \|\vec{w}\|^2$$

1. L1 Regularization

$$E(\vec{w}) = \frac{1}{2} \sum_{n=0}^{N-1} (t_n - y(x_n, \vec{w}))^2 + \lambda |\vec{w}|,$$

Chapter 9 Python Exercise on Decision tree and Linear Regression

Scikit

Linear Regression

Step 1 :- Visualization of the dataset

This is a synthetic dataset. The function used is $y = \frac{x}{2} + \sin(x) + \mathcal{N}(0, 1)$

Step 2: split the data

We split the data into

- training 70%
- validation 15%
- Test 15%

etc.

Step 3: fit a linear model

We fit a line that minimizes the squared error on the training set

— Visualization of the model

The best line is the line that the model has learnt to fit to the data. It minimizes the squared error on the training set

Step 4: Evaluation of Mean Sq. error (MSE)

The validation and test error give an estimate of how well the model would generalize to unseen data: the lower the better

(2) Logistic Regression

Classification by a linear decision boundary

Step 1: Visualization of dataset

- We will use a truncated version of the iris dataset in this example

- We will consider only two input features
  - sepal length (in cm)
  - sepal width (in cm)
- And only two output classes
- Iris setosa
- Iris versicolour

Step 2 Make training validation and test split

Training - 70%, validation - 15%, Test - 15%

Step 3: Fit the logistic regression model

We fit a decision boundary that minimizes the classification error

Step 4 Visualize the decision boundary

We show the area of the feature space assigned to the two different classes

Blue belong to class 0 and orange class 1

Step 5 Evaluate the model

0% validation and test error was only possible because the classes were linearly separable

Decision tree regression

are used to learn piecewise linear models to fit the data

Step 1: Data preparation

We use the same synthetic dataset and make the same kind of train, validation and test splits as in the linear regression example

Step 2: Fit the model

We study models of different maximum tree depth

Step 3: Evaluation and visualization

With rising depth, the model fits tighter to the data

Variation of error with max. depth of tree

We see that overfitting sets in for max_depth > 7

Decision tree classification

Learn piecewise linear decision boundaries

Step 1: Prepare dataset

Step 2: Fit model

Step 3: Visualize the model

Evaluate model

Tutorial 11

- Linear regression
- decision tree

Linear regression

Q:- Prepare you have given a set of training examples

$\{(x_1, y), (x_2, y_2), \ldots (x_n, y_n)\}$

Find the eqn. of the line that best fit the data in the sense that it minimize the sq errors

Ans:- Estimation of sq. errors

$$J = \sum_{i=1}^{N} [y_i - f(x_i)]^2$$

where $f()$ is your linear regression hypothesis

then $f(x) = mx + c$

$f : X \rightarrow Y$

$y = mx + c, = fx$

$$J = \sum_{i=1}^{N} [y_i - m_i - c]^2$$

→ minimize this w.r.t model parameters of straight line

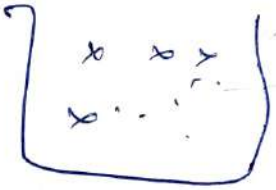
1. $\frac{\partial J}{\partial m} = 0 \Rightarrow 2 \sum_{i=1}^{n} [y_i - mx_i - c](-x_i) = 0$

$$\Rightarrow \boxed{\sum_{i=1}^{n} [y_i - mx_i - c] x_i = 0} \quad \text{(1)}$$

2. $\frac{\partial J}{\partial c} = 0 \Rightarrow 2 \sum_{i=1}^{n} [y_i - mx_i - c][-1] = 0.$

$$\Rightarrow \boxed{\sum_{i=1}^{n} y_i - mx_i - c] = 0} \quad \text{(2)}$$

Decision tree



Data point $\in R^N$

N : dimensionality of the feature space

you have N features of input data

1. A feature f_i
2. A pt. of split on that feature axis f_i

such that the set of data pt. going into the child nodes are homogeneous the most

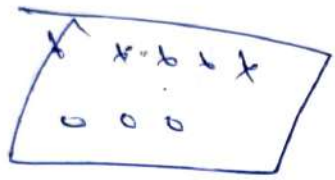
Entropy of set say we have pt. from N different

classes in a set. Say the probability/fraction of pt of the different classes are

$\{f_1, f_2 \ldots f_n\}$ such that $f_1 + f_2 + \ldots + f_n = 1$

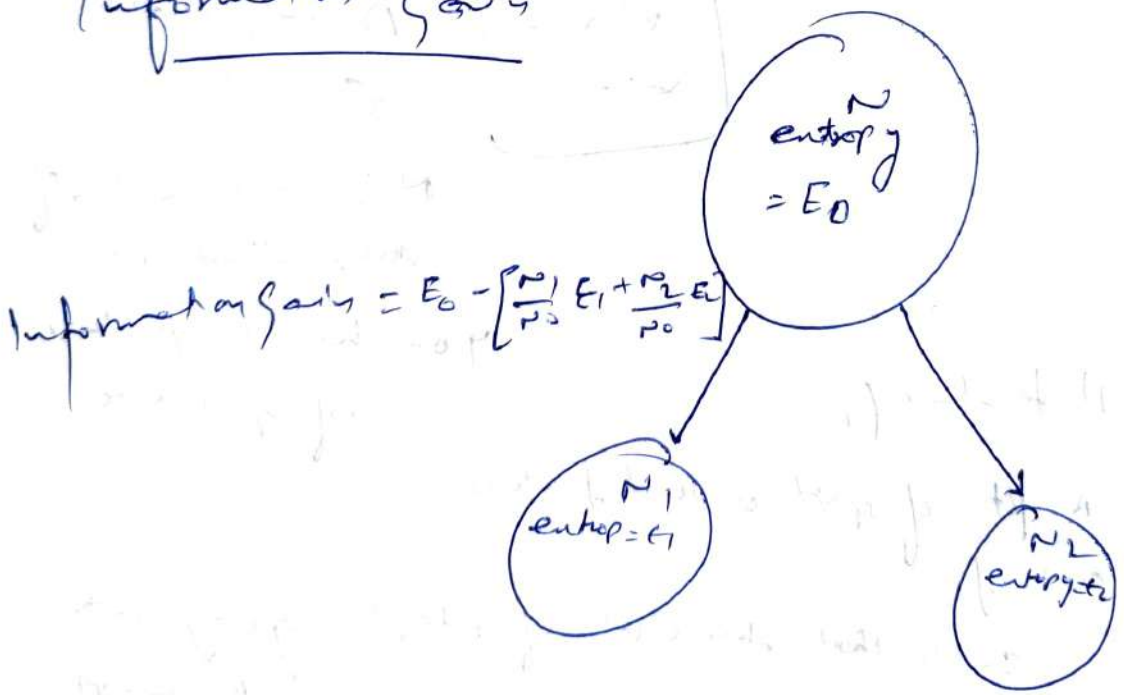
Entropy of this set is given by

$$E = \sum_{i=1}^{n} f_i \log_2 f_i \quad \text{bits}$$

eg  $f_x = \frac{5}{8}$, $f_o = \frac{3}{8}$

$$\text{Entropy} = -\frac{3}{8}\log\frac{3}{8} - \frac{5}{8}\log\frac{5}{8}$$

Information Gain



$$\text{Information gain} = E_0 - \left[\frac{N_1}{N_0} E_1 + \frac{N_2}{N_0} E_2\right]$$

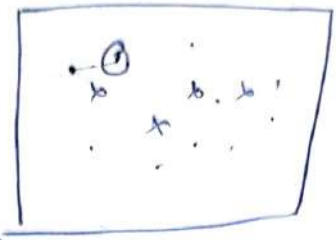
Chapter 10 K-Nearest Neighbour

Instance based learning

Lazy algorithm.

(x_i, y_i)

$(x_i, f(x_i))$



Find similar instances

Training phase :- Save the examples

Prediction time :- Get the test instances x_t

find the training ex. $(x_1, y_1), (x_2, y_2)$

that is closest to x_t

Predict Y_1 as the output y_t

Classification predict the majority class from $[y_1, y_2 \cdots y_3]$

Regression predict the average of $[y_1, y_2 \cdots y_4]$

Voronoi diagram

Basic K-nearest neighbour classification

- Training method
  - Save the training examples
- At prediction time:
  - Find the k training examples $(x_1, y_1) \cdots$ (x_k, y_k) that are closest to the test example x
- Classification: Predict the average of among the y_i's

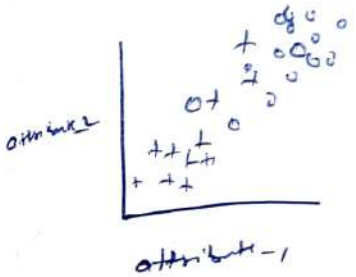
- Improvements:
  - Weighting examples from the neighborhood
  - Measuring "closeness"
  - finding "close" examples in large training set quickly

K - Nearest Neighbor

$$Dist(c_1, c_2) = \sqrt{\sum_{i=1}^{n} (attr_i(c_1) - attr_i(c_2))^2}$$

Prediction $x_{test} = ?$

- Average of k pt. most reliable when:
  - noise in attributes
  - noise in class labels
  - classes partially overlap



$x_i = (x_{i1}, x_{i2} \cdots, x_{iN})$

$x_j = (x_{j1}, x_{j2} \cdots x_{jN})$

Dist. Euclidean (x_i, x_j)

$$= \sqrt{\sum_{i=1}^{n} (x_{ik} - x_{jk})^2}$$

- Noise in attributes
- Noise in class label
- class may partially overlap

Equal weights to all attributes?

- only if the scale of the attributes all similar and the difference
- Scale attributes to equal range and equal variance

- Classes are spherical in shape

- use larger k
- use weighted distance function

Small k captures fine structure of problem space better may be necessary for small training set

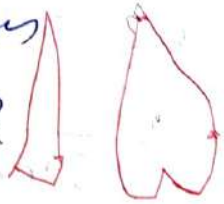
Large k less sensitive noise (particularly class noise)
better prob estimates for discrete classes

- Larger training set allows use of large k

Weighted Euclidean Distance

$$D(c1, c2) = \sqrt{\sum_{i=1}^{n} w_i \cdot (attr_i(c1) - attr_i(c2))^2}$$

- Large weights ⇒ attribute is more important
- Small weights ⇒ attribute is less important
- Zero weight ⇒ attribute doesn't matter
- Weights allow KNN to be effective with axis-parallel elliptical classes



- When do weights come from?

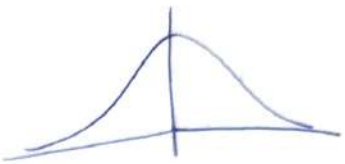
Distance-Weighted KNN

- tradeoff between small and large k can be difficult
- use large k, but more emphasis on nearer

neighbors?

$$\text{prediction}_{test} = \frac{\sum_{i=1}^{k} w_i * class_i}{\sum_{i=1}^{k} w_i} \quad \left(\text{or} \quad \frac{\sum_{i=1}^{k} w_i * value_i}{\sum_{i=1}^{k} w_i} \right)$$

$$w_k = \frac{1}{Dist(c_k, c_{test})}$$



Locally Weighted Averaging

- Let k = no of training points,
- Let weight fall-off rapidly with distance

$$\text{prediction}_{test} = \frac{\sum_{i=1}^{t} w_i * class_i}{\sum_{i=1}^{k} w_i} \quad \left(\text{or} \quad \frac{\sum_{i=1}^{k} w_i * value}{\sum_{i=1}^{k} w_i} \right)$$

$$w_k = \frac{1}{e^{Kernel\ Width\ Dist(c_k, c_{test})}}$$

- kernel width control size of neighborhood that has large effect on value (analogous to k)

chapter 11 feature selection

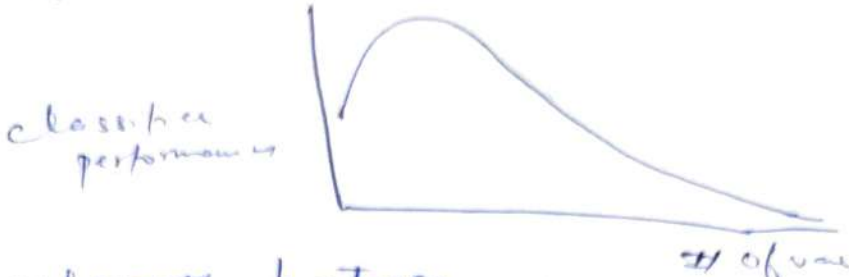
Feature Reduction

features contain information about target

More features $\Rightarrow$ { More information; Better discriminative power }

Curse of Dimensionality

- no. of training eg. is fixed
⇒ the classifier's performance usually will degrade for a large no of features!



Irrelevant features — Limited training eg
Redundant features — Limited computational resources

- Feature selection
- feature reduction

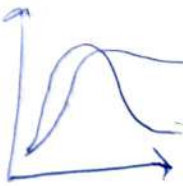
$$f = \{x_1, x_2, \ldots x_n\}$$

$$f' \subset f = \{x_1', x_2', \ldots x_m'\}$$

- Feature Extraction

Projection to $m < n$ dimension

optimize ——— either improve or maintain accuracy
↘ simplify classifier complexity



2^n possible subsets

- optimum method
- Heuristic method
- Randomized

Evaluation

- unsupervised – filters
- supervised – wrapper

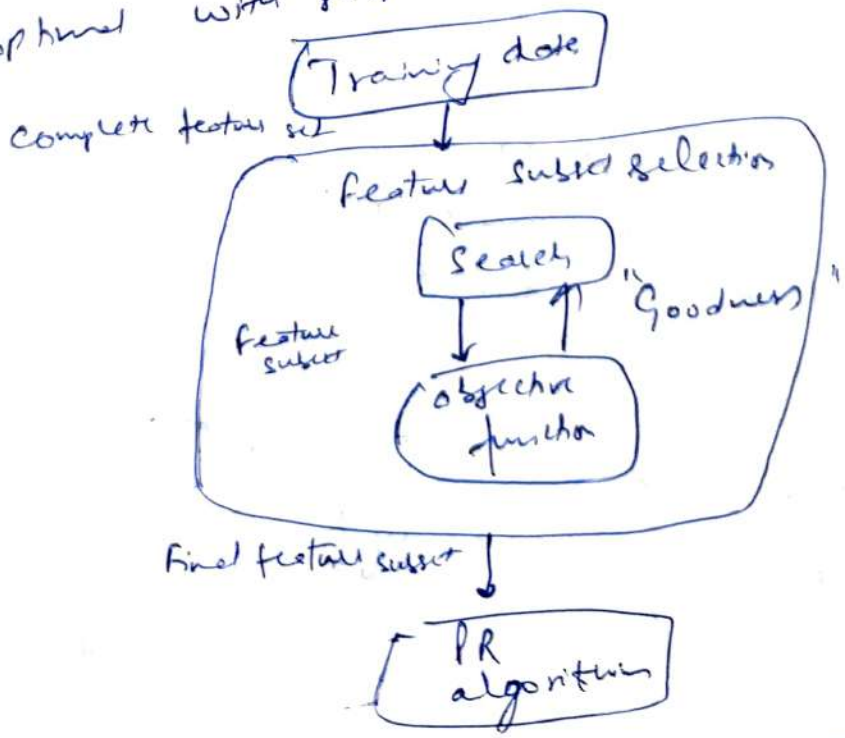
Subset selection

- d initial features
- there are 2^d possible subsets
- Criteria to decide which subset is the best
  - classifier based on these m features, has the lowest probability of error of all such classifiers.
- can't go over all 2^d possibilities
- need some heuristics.

Feature selection steps

Feature selection is an optimization problem

- step 1: search the space of possible feature subsets
- step 2: Pick the subset that is optimal or near optimal with respect to some objective function



Forward selection

- start with empty feature set
- Try each remaining features
- Estimate classification/reg. error for adding each feature
- select feature that gives max. improvement
- stop when there is no significant improvement

Backward search

- start with full feature set
- Try removing features
- Drop feature with smallest impact on error

Feature selection

Univariate (look at each feature independently of others)

- Pearson correlation coefficient
- f-score
- chi-square
- Signal to noise ratio
- mutual information
- Etc

Univariate methods measure some type of correlation s/w two random variables - the label (y_i) and fixed feature (x_{ij} for fixed j)

Rank features by importance

Ranking cut-off is determined by user

Pearson correlation coefficient

- Measures the correlation b/w two variables
- formula for Pearson correlation:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

- The correlation r is b/w +1 and −1
  - +1 means perfect +ve correlation
  - −1 ~~means~~ in other direction.

Signal to noise ratio

- Difference in means divided by difference in Standard deviation b/w the two classes

$$SNR(x,y) = (\mu_x - \mu_y)/(\sigma_x - \sigma_y)$$

- Large values indicate a strong correlation

~~Multivariate~~ Multivariate feature selection

- Multivariate (considers all features simultaneously)
- consider the vector w for any linear classifier
- ~~classifier~~ classification of pt b is given by $w^T x + b$
- Small entries of w will have little effect on the dot product and therefore those features are less relevant
- for example if $w = (10, 0.01, -9)$ then feature 0 and 2 all contributing more to the dot product than feature 1.
  - A ranking of features given by this w is 0, 2, 1

- The w can be obtained by any of linear classifiers
- A variant of the approach is called recursive feature elimination:
  - Compute w on all features
  - Remove feature with smallest w_i
  - Recompute w on reduced data
  - If stopping criterion not met then go to step 2

Chapter 12 feature extraction

$$\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}}_{x} \longrightarrow \underbrace{\begin{bmatrix} z_1 \\ z_2 \\ \\ z_m \end{bmatrix}}_{z} = f\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

$M < N$

Find a projection matrix w

$$z = w^T x$$

- Uncorrelated, cannot be reduced further
- Large variance

Geometrical picture of Principal components

Algebraic definition of PCs

Given a sample of p observations on vector of N variables

$\{x_1, x_2, \dots x_p\} \in \mathbb{R}^n$

define the principal component of the sample by the linear transformation

$$Z_1 = w_1^T x_j = \sum_{i=1}^{n} w_{i1} x_{ij}, \quad j = 1, 2, \dots, p$$

where the vector $w_1 = (w_{11}, w_{21}, \dots w_{n1})$

$x_j = (x_{1j}, x_{2j}, \dots, x_{nj})$

is chosen such that $\text{var}[Z_1]$ is maximum

$x_1 - - [\quad]$

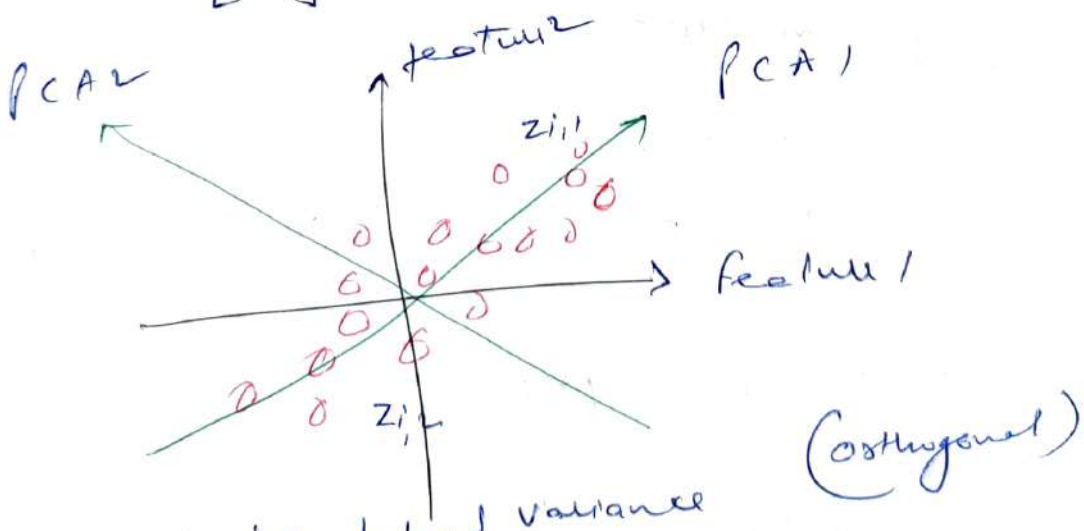
$x_2 - - [\quad]$

$x_3 - - [\quad]$

$\vdots$

$x_p - - [\quad]$

$\text{Var}[Z_1]$ is max...



(orthogonal)

① Maximize total variance

② Minimize correlation

Choose $M \leq N$ orthogonal directions

which maximize total variance

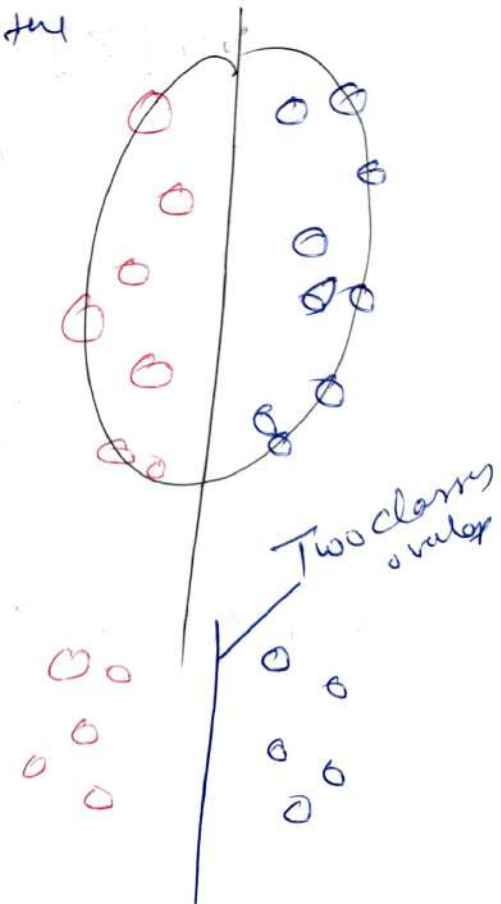
Trang data $Cov(X) = \Sigma$

select M largest eigenvector

- N - dimensional feature space
- $N \times N$ symmetric covariance matrix estimated from sample $Cov(x) = \Sigma$
- select M largest eigenvalue of the covariance matrix and associated M eigenvectors
- first eigenvector will be direction with largest variance

Is PCA a good criterion for classification?

- Data variation determine the projection direction.
- what's missing
  - class information



what is a good projection?

- similar, what is a good criterion?
  - separating different classes

What class information may be useful?

- B/w-class distance
  - Distance b/w the centroids of different classes

within-class distance

- Accumulated distance of an instance to the centroid of its class

- Linear discriminant analysis (LDA) find most discriminant projection by
  - maximizing b/w-class distance
  - and minimizing within-class distance

Linear Discriminant Analysis

- find a low-dimensional space such that when x is projected, classes are well separated

Good projection

- means are as far away as possible
- scatter is small as possible
- Fisher Linear Discriminant

$$J(w) = \frac{(m_1 - m_2)^2}{s_1^2 + s_2^2}$$

Chapter 13 collaborative filtering

Recommended system

(1) Item Recommended

(2) Rating prediction system

U = set of ~~users~~ user ρ : Users $\longrightarrow$ R

S = set of items - Learn ρ from data

- use ρ to predict the utility value of each item to each user

(1) Content Based

- based on Content Similarity

(2) - collaborative filtering system

- Similar user

	S_1	S_2	S_3	S_4	S_5	S_M
u_1	5	-	-	-	2	3
u_2	3	2	?			4
u_3	2	2	4	-	-	
u_4						
u_5						
u_M						

User ~~based~~ nearest method

Item based method

using

Collaborative filtering for Rating Prediction

- User-based Nearest Neighbour
- Neighbour = Similar user
- Generate a prediction for an item i by analyzing rating for i from user in u's neighbourhood

Neighborhood formation phase

- Let the record of the target user be u, and the record of another user be v ($v \in T$)
- The similarity b/w the target user, u, and a neighbor, v, can be calculated using Pearson's correlation coefficient

$$sim(u,v) = \frac{\sum_{i \in c} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in c} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in c} (r_{v,i} - \bar{r}_v)^2}}$$

Recommendation Phase

- Use the following formula to compute the rating prediction of item i for ~~target~~ user u

$$p(u,i) = \bar{r}_u + \frac{\sum_{v \in V} sim(u,v) \times (r_{v,i} - \bar{r}_v)}{\sum_{v \in V} |sim(u,v)|}$$

where V is the set of k similar user, $r_{v,i}$ is the rating of user v given to item i,

Issue with the user based KNN CF

- The problem with the user-based formulation of collaborative filtering is the issue of scalability:
  - it requires the real time comparison of the target user to all user record in order to

generate predictions.

- A variation of this approach that remedies this problem is called item based CF

- Lack of scalability

- item based CF

by comparing item based on their pattern of rating across user. The similarity of item i and j is computed as follows

$$sim(i,j) = \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_u)(r_{u,j} - \bar{r}_u)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{u \in U} (r_{u,j} - \bar{r}_u)^2}}$$

Chapter 14 Python exercise PCA and k-means

Chapter 15

Chapter 16 Bayesian learning

Probability : modeling concepts

Bayesian Probability : Partial ~~beliefs~~ beliefs

Bayesian Estimation: Calculate validity of a proposition

- Prior estimate
- New relevant evidence

Bayes Theorm

$$P(h/D) = \frac{P(D|h) \cdot P(h)}{P(D)} \rightarrow \text{Prior}$$

$$P(h \cap D) = P(h) \cdot P(D|h)$$

$$P(D \cap h) = P(D) \cdot P(h/D)$$

Bayes theorm!

Goal: to determine the most probable hypothesis given the data D plus any initial knowledge about the prior probality of various hypotheses in H

An example: Does patient have cancer or not?

A patient takes a lab test and the result comes back positive. The test returns a correct +ve result in only 98% of the case in which the disease is actually present, and a correct -ve result in only 97% of the cases in which the disease is not present. Furthermore, 0.008 of the entire population have this cancer

$$P(\text{cancer}) = 0.008, \quad P(-\text{cancer}) = 0.992$$

$$P(+|\text{cancer}) = 0.98, \quad P(-|\text{cancer}) = 0.02$$

$$P(+|-\text{cancer}) = 0.03, \quad P(-|-\text{cancer}) = 0.97$$

$$P(\text{cancer}|+) = \frac{P(+|\text{cancer})^{0.98} P(\text{cancer})^{0.02}}{P(+)}$$

$$P(-,\text{cancer}|+) = \frac{P(+|-\text{cancer})^{0.03} P(-\text{cancer})^{0.992}}{P(+)}$$

Maximum A Posteriori (MAP) hypothesis

$$P(h|D) = \frac{P(D|h) P(h)}{P(D)}$$

The Goal of Bayesian learning: the most probable hypothesis given the training data (Maximum A Posteriori hypothesis)

$$h_{MAP} = \arg\max_{h \in H} P(h|D)$$

$$= \arg\max_{h \in H} \frac{P(D|h) P(h)}{P(D)}$$

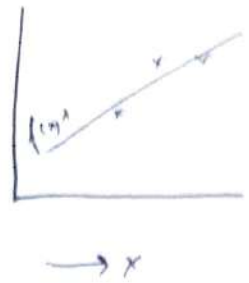
$$= \arg\max_{h \in H} P(D|h) P(h)$$

Learning a real valued function

f: Target function

$$(x_i, d_i) \quad d_i = f(x_i) + (\varepsilon_i)$$

$$d_i \sim N(f(x_i), \sigma^2)$$



$$h_{ML} = \underset{h}{\operatorname{argmax}}\ P(D|h)$$

$$= \underset{h}{\operatorname{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(d-h(x))^2}{\sigma}}$$

$$= \underset{h}{\operatorname{argmax}} \sum_{i=1}^{m} -\frac{1}{2}\ln(2\pi\sigma^2) - \frac{1}{2}\left(\frac{d-h(x)}{\sigma}\right)^2$$

$$\underset{h}{\operatorname{argmin}} \sum_{i=1}^{m} (d_i - h(x_i))^2$$

Bayes Optimal Classifier

Training data

h_{MAP}

h_1, h_2, h_3

$P(h_1|D) = .4$

$P(h_2|D) = .3$

$P(h_3|D) = .3$

$\overset{X}{=}$

$h_1(x) = +$

$h_2(x) = -$

$h_3(x) = -$

$$\text{argmax}_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i) \cdot P(h_i|D)$$

Bayes Optimal classifier

Q:- Given new instance x, what is the most probable classification

- $h_{MAP}(x)$ is ~~not~~ most probable classification

Gibbs Sampling

Choose a hypothesis randomly acc to $P(h|D)$

Use it to classify the new instance

Chpt 17 Naive Bayes

$$P(h|D) = \frac{P(D|h) P(h)}{P(D)}$$

$$P(Y|\bar{X}) \propto P(\bar{x}|Y)\, P(Y)$$

$$P(x_1, x_2 \ldots x_n|Y) \cdot P(Y)$$

Naive Bayes Assumption

$$= P(x_1|Y) \cdot P(x_2|x_1, Y) \; P(x_n|x_1 \ldots x_{n-1}) \; P(Y)$$

$$= P(x_1|Y) \cdot P(x_2|Y) \; P(x_n|Y) \cdot P(Y)$$

Bayes rule:

$$P(Y=y_k \mid X_1 \dots X_n) = \frac{P(Y=y_k)\, P(X_1 \dots X_n \mid Y=y_k)}{\sum_j P(Y=y_j)\, P(X_1 \dots X_n \mid Y=y_j)}$$

Assuming conditional independence among X_i's:

$$P(Y=y_k \mid X_1 \dots X_n) = \frac{P(Y=y_k) \prod_i P(X_i \mid Y=y_k)}{\sum_j P(Y=y_j) \prod_i P(X_i \mid Y=y_j)}$$

So classification rule for $X^{new} = \langle X_1, \dots, X_n \rangle$ is:

$$Y^{new} \leftarrow \arg\max_{y_k} P(Y=y_k) \prod_i P(X_i^{new} \mid Y=y_k)$$

Naive Bayes Algorithm – discrete X_i

- Train Naive Bayes (examples)

  for each value y_k

    estimate $\pi_k \equiv P(Y=y_k)$

  for each value x_{ij} of each attribute X_i

    estimate $\theta_{ijk} \equiv P(X_i = x_{ij} \mid Y=y_k)$

- classify (X^{new})

  $$Y^{new} \leftarrow \arg\max_{y_k} P(Y=y_k) \prod_i P(X_i^{new} \mid Y=y_k)$$

  $$Y^{new} \leftarrow \arg\max_{y_k} \pi_k \prod_i \theta_{ijk}$$

Kusum

- Probability must sum to 1, so need estimate only $n-1$ parameters

Estimating Parameters: Y, X_i discrete-valued

Maximum likelihood estimates (MLE's)

$$\hat{\pi}_k = \hat{P}(Y = y_k) = \frac{\#D\{Y = y_k\}}{|D|}$$

~~$\hat{\theta}_{ijk} = \hat{P}(X_i = x_{ij} \wedge Y = y_k)$~~

$$\hat{\theta}_{ijk} = \hat{P}(X_i = x_{ij} \mid Y = y_k) = \frac{\#D\{X_i = x_{ij} \wedge Y = y_k\}}{\#D\{Y = y_k\}}$$

no. of items in set D for which $Y = y_k$

Estimating Parameters: Y, X_i discrete-valued

If unlucky, our MLE estimate for $P(X_i \mid Y)$ may be zero

$$\hat{\pi}_k = \hat{P}(Y = y_k) = \frac{\#D\{Y = y_k\}}{|D|}$$

$$\hat{\theta}_{ijk} = \hat{P}(X_i = x_{ij} \mid Y = y_k) = \frac{\#D\{X_i = x_{ij} \wedge Y = y_k\}}{\#D\{Y = y_k\}}$$

MAP estimates

$$\hat{\pi}_k = \hat{P}(Y = y_k) = \frac{\#D\{Y = y_k\} + 1}{|D| + 1R}$$

only different "imaginary" examples

$$\hat{\theta}_{ijk} = \hat{P}(X_i = x_{ij} \mid Y = y_k) = \frac{\#D\{X_i = x_{ij} \wedge Y = y_k\} + 1}{\#D\{Y = y_k\} + 1m}$$

Assumption of conditional Independence

often the X_i are not really conditionally independent

- We can use Naive Bayes in many cases anyway
  - often the right classification, even when not the right probability

Gaussian Naive Bayes (Continuous)

→ Algorithm: continuous-valued features

- Conditional probability often modeled with the normal distribution

$$P(X_i = x \mid Y = y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} e^{\frac{-(x - \mu_{ik})^2}{2\sigma_{ik}^2}}$$

Sometime assume variance

- is independent of Y (i.e., σ_i)
- or independent of X_i (i.e. σ_k)
- or both (i.e., σ)

Train Naive Bayes (examples)

for each value y_k

estimate $\pi_k = P(y = y_k)$

for each attribute x_i estimate

class conditional mean μ_{ik} variance σ_{ik}

- classify (x^{new})

$$y^{new} \leftarrow \arg\max_{y_k} P(Y = y_k) \prod_i P(x_i^{new} | Y = y_k)$$

$$y^{new} \leftarrow \arg\max_{y_k} \pi_k \prod_i \text{Normal}(x_i^{new}, \mu_{ik}, \sigma_{ik})$$

Estimating Parameters: Y discrete, X_i continuous

Maximum likelihood estimate:

$$\bar{\mu}_{ik} = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j X_i^j \delta(Y^j = y_k)$$

$\bar{\mu}_{ik}$: i^{th} feature, k^{th} class; X_i^j: j^{th} training example; $\delta(z) = 1$ if z true else 0

$$\bar{\sigma}_{ik}^2 = \frac{1}{\sum_j \delta(Y^j = y_k)} \sum_j (X_i^j - \bar{\mu}_{ik})^2 \delta(Y^j = y_k)$$

Chapt 18 Bayesian Networks

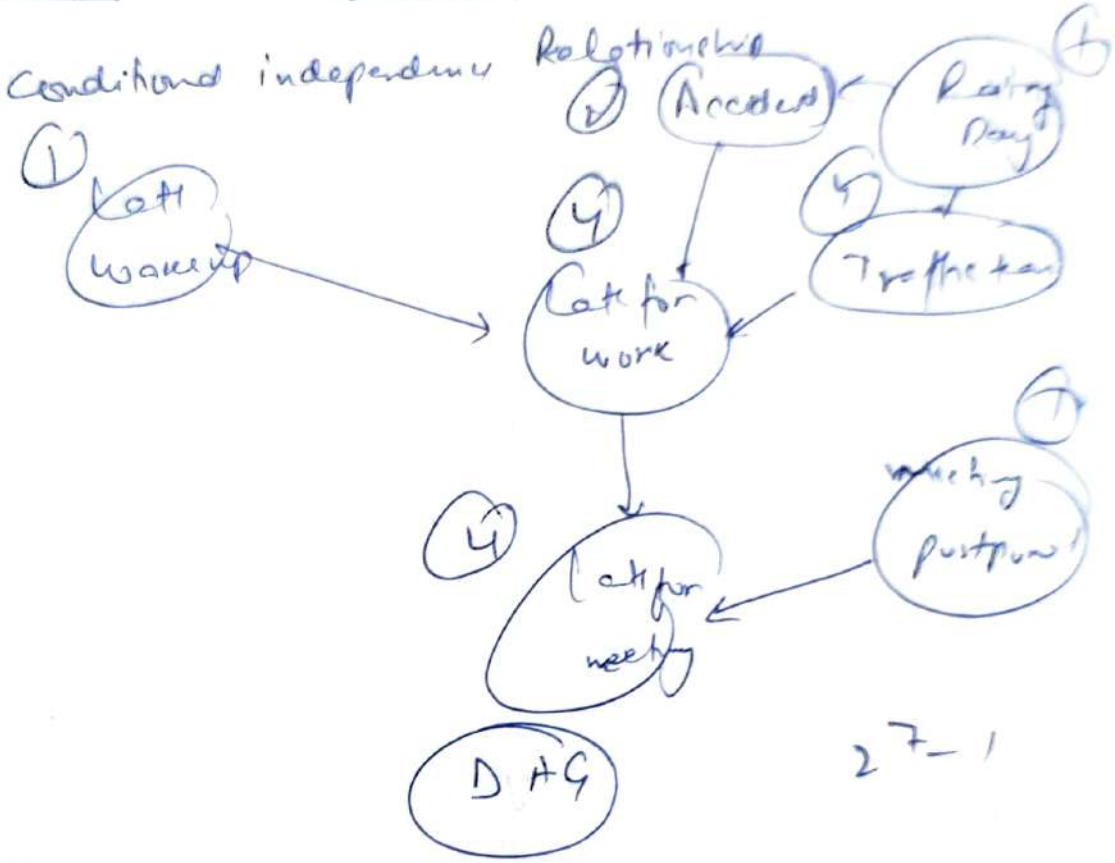
- is a type of graphical model

$X = \{X_1, X_2, X_n\}$

Nodes

Arcs Causality

Conditional independence Relationship



$2^7 - 1$

Represents efficiently the joint probability distribution of the variables

Arcs representation probabilistic

- Arcs represent probabilistic dependencies among variables
- DAG
- Each node local probability distribution (CPT)

Conditional probability tables associated w/ its each node specifies the the conditional distribution for the variable given its immediate Parent in the graph.

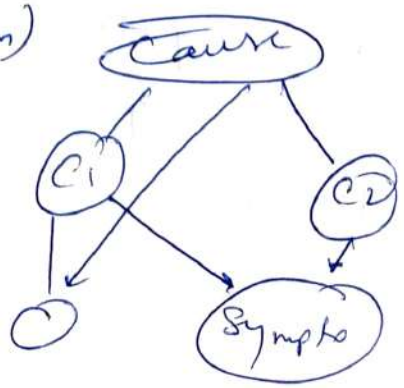
Each node is asserted to a conditionally independent of its non-descendant | given its immediate parents.

Inference captures posterior probabilities given evidence about some nodes

- Exact inference is intractable for arbitrary BN
- Approximate techniques eg Monte carlo methods can also be used
- Efficient algorithm that leverage the structure of graph

Application of Bayesian Networks

- Diagnosis: P(cause | symptom) = ?
- Prediction: P(symptom | cause) = ?
- classification: P(class | data)
- Decision-making (given a cost function)



- Structure of the graph ↔ conditional independence
- dence relations

In general

$$P(X_1, X_2, \dots X_n) = \prod_i P(X_i \mid \text{parents}(X_i))$$

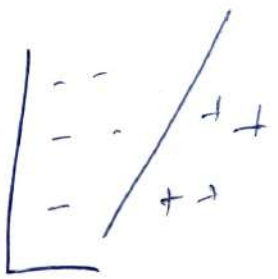
↑ The full joint distribution

↑ The graph-structure approximates

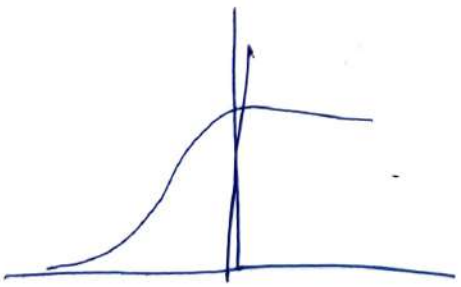
- Requires that graph is acyclic
- 2 components to a Bayesian network
  - the graph structure
  - The numerical probabilities

Chpt ~~lecture~~ 3 Logistic regression

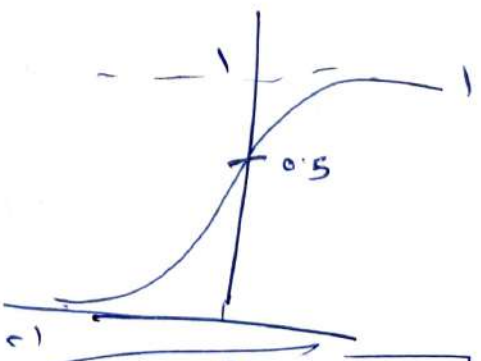
linear regression $h(x) = \sum_{i=0}^{n} \beta_i x_i$



Logistic Regression



$$g(z) = \frac{1}{1+e^{-z}}$$



$$h(x) = g(\beta^T x)$$

$$g(z) = \frac{1}{1+e^{-z}}$$

$\sigma(z)$ sigmoid function

$$= \frac{1}{1+e^{-\beta^T x}}$$

$g(z) \to 1$ as $z \to \infty$

$g(z) \to 0$ as $z \to -\infty$

$$g'(z) = \frac{1}{(1+e^{-z})^2} * e^{-z}$$

$$= \frac{1}{1+e^{-z}} \left(1 - \frac{1}{1+e^{-z}}\right)$$

$$= g(z).(1-g(z))$$

$$\boxed{p(y|x)} = h(x)^y (1-h(x))^{1-y}$$

$$h(x) = \frac{1}{1+e^{-\beta^T x}}$$

$p_y(x;\beta)$ is our estimate of $p(y|x)$

Learn optimal value of β

$$L(\beta) = p(y|x;\beta)$$

$$= \prod_{i=1}^{m} p(y_i|x_i;\beta)$$

$$= \prod_{i=1}^{m} (h(x_i)^{y_i} - (1-h(x_i))^{1-y}$$

$$l(\beta) = \sum_{i=1}^{m} y_i \log(h(x_i)) + (1-y_i)\log(1-h(x_i))$$

$$\beta = \beta + \alpha \nabla_\beta l(\beta)$$

x, y

$$\frac{\partial}{\partial \beta}(l(\beta)) = \left(y \cdot \frac{1}{g(\beta^T x)}\right) - (1-y)\frac{1}{1-g(\beta^T x)} \frac{\partial}{\partial \beta_j} g(\beta^T x)$$

$$= y \frac{1}{g(\beta^T x)} - (1-y)\frac{1}{1-g(\beta^T x)}(1-g(\beta^T x))\frac{\partial}{\partial \beta} \beta^T x$$

$$= (y(1-g(\beta^T x)) - (1-y)g(\beta^T x))\, x_j$$

$$= (y - h_\beta(x))\, x_j$$

$$= (y - h_\beta(x))\, x_j$$

Put above value in this eqn

$$\beta = \beta + \alpha \nabla_\beta l(\beta)$$

$$\beta_j = \beta_j + \alpha (y - h_\beta(x)) x_j$$

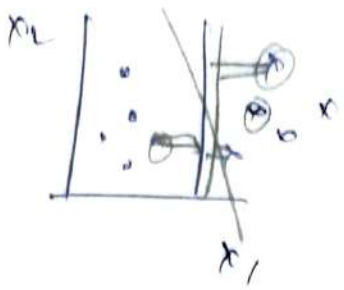
Lecture 20 Introduction to support vector machine

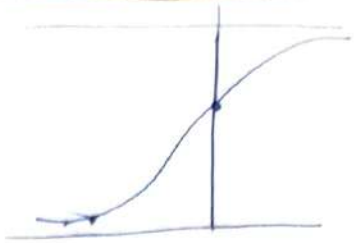
Logistic regression

$$P(y=1|x) = h(x) = \sigma(\beta^T x)$$

Predict 1 when $h(x) > 0.5$

0 when $h(x) < 0.5$

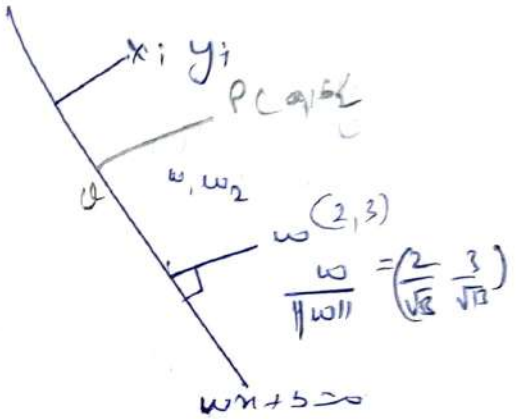




Functional Margin distance of (x_i, y_i) from (w, b)

(x_i, y_i) wrt (w, b)

$$\gamma_i' = y_i (w^T x_i + b)$$



$$2x + 3y + 1 = 0$$

$$4x + 6y + 2 = 0$$

$$20x + 30y + 10 = 0$$

of $S = \{(x_1, y_1) \ldots (x_m, y_m)\}$

$$\gamma = \min_{i=1}^{m} \gamma^i$$

Geometrical Margin

$$2x + 3y + 1 = 0$$

$$P = O + \gamma \frac{w}{\|w\|}$$

$$(a_1, a_2) = (b_1, b_2) + \gamma \frac{w}{\|w\|}$$

$$w^T \left((a_1, a_2) - \gamma \frac{w}{\|w\|} \right) + b = 0$$

$$\Rightarrow \gamma = \frac{w^T (a_1, a_2) + b}{\|w\|}$$

$$= \frac{w^T}{\|w\|} (a_1, a_2) + \frac{b}{\|w\|}$$

$$\gamma = y_i \left(\frac{w^T}{\|w\|} (a_1, a_2) + \frac{b}{\|w\|} \right)$$

$$\gamma = \min \gamma_i$$

Maximize margin width

- maximize $\frac{\gamma}{\|w\|}$ subject to
- $y_i (w^T x_i + b) \geq$ for $i = 1, 2, \ldots n$
- scale so that $\gamma = 1$
- maximize $\frac{1}{\|w\|}$ is the same as minimize $\|w\|^2$
- minimize $w.w$ subject to the constraints
- for all (x_i, y_i), $i = 1, \ldots, m$:
  $w^T x_i + b \geq 1$ if $y_i = 1$

$w^T x_i + b \leq -1$ if $y_i = -1$

Large Margin Linear Classifier

formulation

minimize $\frac{1}{2}\|w\|^2$

such that

$y_i(w^T x_i + b) \geq 1$

Solving the optimization problem

minimize $\frac{1}{2}\|w\|^2$

s.t $y_i(w^T x_i + b) \geq 1$

Slide 21 The Dual formulation

minimize $\frac{1}{2}\|w\|^2$

s.t $y_i(w^T x_i + b) \geq 1$

- optimization problem with convex quadratic objective and linear constraint.
- can be solved using QP
- Lagrange duality to get the optimization problem in dual form;
  - Allow us to use kernel to get optimal margin classifiers to work efficiently in very high dimensional spaces
  - Allow us to derive an efficient algorithm for solving the above optimization problem that will

typically do much better than generic QP software

The primal Problem

$$\min_w f(w)$$

$$\text{s.t.} \quad g_i(w) \le 0, \quad i = 1, \dots k$$

$$h_i(w) = 0, \quad i = 1, \dots l.$$

the generalized Lagrangian

$$L(w, \alpha, \beta) = f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{l} \beta_i h_i(w)$$

this α's ($\alpha_i \ge 0$) and β's are called lagrange multipliers

Lemma:

$$\max_{\alpha, \beta, \alpha_i \ge 0} L(w, \alpha, \beta) = \begin{cases} f(w) & \text{if } w \text{ satisfies primal constraints} \\ \infty & \text{otherwise} \end{cases}$$

A re-written Primal

$$\min_w \max_{\alpha, \beta, \alpha_i \ge 0} L(w, \alpha, \beta)$$

Lagrangian Duality, cont.

The Primal Problem $p^* = \min_w \max_{\alpha, \beta, \alpha \ge 0} L(w, \alpha, \beta)$

The Dual Problem: $d^* = \max_{\alpha, \beta, \alpha_i \ge 0} \min_w L(w, \alpha, \beta)$

Theorem (weak duality)

$$d^* = \max_{\alpha, \beta, \alpha_i \ge 0} \min_w L(w, \alpha, \beta) \le \min_w \max_{\alpha, \beta, \alpha_i \ge 0} L(w, \alpha, \beta) = p^*$$

Theorem (strong duality)

Iff there exist a saddle pt of $L(w, \alpha, \beta)$

we have $d^* = p^*$

The KKT conditions

If there exists some saddle pt of L then it satisfies the following "Karush-Kuhn-Tucker" (KKT) conditions:

$$\frac{\partial}{\partial w_i} L(w, \alpha, \beta) = 0, \quad i = 1, \ldots n$$

$$\frac{\partial}{\partial \beta_i} L(w, \alpha, \beta) = 0, \quad i = 1, \ldots, l$$

$$\alpha_i g_i(w) = 0, \quad i = 1, \ldots, m$$

$$g_i(w) \leq 0, \quad i = 1, \ldots m$$

$$\alpha_i \geq 0, \quad i$$

Theorem: If w^*, α^*, and s^* satisfy the KKT condition, then it is also a soln. to the primal and the dual problem.

Support vector

- Only a few α_i's can be nonzero
- Call the training data pts whose α_i's are nonzero the support vectors

$\alpha_i g_i(w) = 0 \quad i = 1, \ldots m$

If $\alpha_i > 0$ then $g(w) = 0$

Solving the optimization problem

Quadratic Programming with linear constraints

$$\text{minimize } \frac{1}{2}\|w\|^2$$
$$\text{s.t } y_i(w^T x_i + b) \geq 1$$

Lagrangian function

$$\text{minimize } L_p(w, b, \alpha_i) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i (y_i(w^T x_i + b) - 1)$$

$$\text{s.t } \alpha_i \geq 0$$

minimize with respect to w and b

$$\frac{\partial L_p}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

$$\frac{\partial L_p}{\partial b} = 0 \Rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0$$

$$L_p(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i^T x_j) - b \sum_{i=1}^{m} \alpha_i y_i$$

$$L_p(w, b, \alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i^T x_j)$$

The Dual Problem

Now we have the following dual opt problem

$$\text{max}_{\alpha} J(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i^T x_j)$$

$$\text{s.t } \alpha_i \geq 0, \quad i = 1, \ldots k$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0$$

Primal

This is a quadratic programming problem
- A global maximum of α_i can always be found

SVM

- Once we have Lagrange multipliers $\{\alpha_i\}$ we can reconstruct the parameter vector w as a weighted combination of training examples.

$$w = \sum_{i=1}^{m} \alpha_i y_i x_i \qquad w = \sum_{i \in SV} \alpha_i y_i x_i$$

- For testing with a new data z
  - compute

$$w^T z + b = \sum_{i \in SV} \alpha_i y_i (x_i^T z) + b$$

and classify z as class 1 if the sum is +ve, and class 2 otherwise

Note: w need not be formed explicitly

Solving the optimization Problem

- The discriminant function is:

$$g(x) = w^T x + b = \sum_{i \in SV} \alpha_i x_i^T x + b$$

- It relies on a dot product b/w the test pt. x and the support vector x_i

- Solving the optimization problem involved computing the dot product $x_i^T x_j$ b/w all pairs of training pts
- The optimal w is a linear combination of small no. of data pts

Chapter 2 Maximum margin with noise

Linear SVM formulation

Find w and b such that

$$\frac{2}{\|w\|} \text{ is maximized}$$

For each training pt.

$$y(wx_i + b) \geq 1$$

$\frac{1}{2}\sqrt{w \cdot w}$ is minimized

$\|w\|^2 = w \cdot w$ minimized

Limitation of previous SVM formulation

- what if the data is not linearly separable
- also noisy data pts.?

Extend the definition of maximum margin to allow non-separating planes.

$w \cdot w + C$ # training errors.

Objective to be minimized

- Minimize

  $w \cdot w + C$ (distance of error pts. the correct one)

- Add slack variables ξ_i

$$M = \frac{2}{\sqrt{w \cdot w}}$$

15456

Minimize

$$w \cdot w + c \sum_{k=1}^{m} \xi_k$$

m constraints

$$\begin{cases} w \cdot x_k + b \geq 1 - \xi_k \text{ if } y_k = 1 \\ w \cdot x_k + b \leq -1 + \xi_k \text{ if } y_k = -1 \end{cases}$$

$\equiv$

$$y_k (w \cdot x_k + b) \geq 1 - \xi_k, \quad k = 1, \ldots, m$$

$$\xi_k \geq 0, \quad k = 1, \ldots, m$$

Lagrangian

$$L(w, b, \xi, \alpha, \beta) = \frac{1}{2} w \cdot w + C \sum_{i=1}^{m} \xi_i + \sum_{i=1}^{m} \alpha_i [y_i (x \cdot w + b) - 1 + \xi_i] - \sum_{i=1}^{m} \beta_i \xi_i$$

α_i's and β_i's are lagrange multipliers (≥ 0)

$$\xi_i \geq 0 \text{ for } i = 1, \ldots, m$$

Dual formulation

Find $\alpha_1, \alpha_2, \ldots \alpha_m$ s.t

$$\max J(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i^T x_j)$$

Linear svm

s.t. $\alpha_i \geq 0 \quad i = 1, \ldots m$

$$\sum_{i=1}^{m} \alpha_i y_i = 1$$

s.t $0 \leq \alpha_i \leq C \quad i = 1, \ldots m$

$$\sum_{i=1}^{m} \alpha_i y_i = 0.$$

Solution to soft Margin classification

- x_i with non zero α_i will be support vectors

Soln to the dual problem is:

$$w = \sum_{i=1}^{m} \alpha_i y_i x_i$$

$$b = y_k (1 - \xi_k) - \sum_{i=1}^{m} \alpha_i y_i x_i^T x_k$$

for any k s.t $a_k > 0$

for classification

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i x_i x + b$$

(we need to compute w explicitly)

Chapter 23 Non Linear SVM and kernel function

$x \longrightarrow \phi(x)$

Map to high dim feature space computable costs?

$$\sum \alpha_i y_j \quad y_i y_j \quad \underline{x_i x_j}$$

$O(d^2)$ $\qquad$ $O(D^2)$

$D >> d$.

Kernel functions

$x_a \quad x_b$

$\phi(x_a) \quad \phi(x_b) \qquad x_a \cdot x_b \qquad \phi(x_a) \cdot \phi(x_b)$

$$k(x_a, x_b) = \phi(x_a) \cdot \phi(x_b)$$

↑

The kernel trick

$$g(x) = w^T \phi(x) + b = \sum_{i \in SV} \alpha_i \boxed{\phi(x_i)^T \phi(x)} + b$$

- We only use the dot product of feature vector in both the training and test.
- A kernel function is defined as a function that corresponds to a dot product of two feature vectors in some expanded feature space

$$K(x_a, x_b) = \phi(x_a) \cdot \phi(x_b)$$

$$x = [x_1, x_2]$$

$$K = (x_i, x_j) = (1 + x_i x_j)^2$$

(x_{i1}, x_{i2}) (x_{j1}, x_{j2})

$$= 1 + x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{j1} x_{i2} x_{j2} + x_{i2}^2 x_{j2}^2 + 2 x_{i1} x_{j1} + 2 x_{i2} x_{j2}$$

$$= [1 \; x_{i1}^2 \; \sqrt{2} x_{i1} x_{i2} \; x_{i2}^2 \; \sqrt{2} x_{i1} \; \sqrt{2} x_{i2}] \cdot [x_{j1}^2 \; \sqrt{2} x_{j1} x_{j2} \; x_{j2}^2 \; \sqrt{2} x_{j1} \; \sqrt{2} x_{j2}]$$

$$\phi(x_i) \cdot \phi(x_j)$$

Commonly used kernel functions

- Linear kernel: $K(x_i, x_j) = x_i \cdot x_j$
- Polynomial of power p:

$$K(x_i, x_j) = (1 + x_i \cdot x_j)^p$$

- Gaussian (radial-basis function)

$$K(x_i, x_j) = e^{\left[\frac{\|x_i - x_j\|^2}{2\sigma^2}\right]}$$

Sigmoid

$$k(x_i, x_j) = \tanh(\beta_0 x_i \cdot x_j + \beta_1)$$

In general, functions that satisfy Mercer's condition can be kernel function.

- kernel function can be thought of similarity measure b/w the input objects
- Not all similarity measure can be used as kernel function.
- Mercer's condition states that any +ve semi-definite kernel $K(x, y)$, i.e

$$\sum_{i,j} K(x_i, x_j) c_i c_j \geq 0$$

- Can be expressed as a dot product in a high dimensional space.

Nonlinear SVM: optimization

- formulation (Lagrangian Dual Problem)

maximize $\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$

such that $0 \leq \alpha_i \leq C$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

The sign of the discriminant function is

$$g(x) = \sum_{i \in SV} \alpha_i K(x_i, x_j) + b$$

Performance

→ SVM work very well in practice
- the user must choose the kernel function and its parameters

→ They can be expensive in time and space for big data set
- The computation of the maximum-margin hyper-plane depends on the square of the number of training ~~cases~~ cases
- we need to store all the support vectors

→ The kernel trick can also be used to do PCA in a much higher-dimensional space, thus giving a non-linear version of PCA in the original space

Multiclass classification

- SVMs can only handle two class outputs
- learn N SVM's
  - SVM 1 learns class 1 vs REST
  - SVM 2 learns class 2 vs REST
  - ⋮
  - SVM N learns class N vs REST

- Then to predict the output for a new input, just predict with each SVM and find out which one puts the prediction the furthest into the reg

Sigmoid

$$k(x_i, x_j) = \tanh(\beta_0 x_i \cdot x_j + \beta_1)$$

In general functions that satisfy Mercer's condition can be kernel function

- kernel function can be thought of similarity measure b/w the input objects
- Not all similarity measure can be used as kernel function.
- Mercer's condition states that any +ve semi-definite kernel $K(x, y)$, i.e

$$\sum_{i,j} K(x_i, x_j) c_i c_j \geq 0$$

- Can be expressed as a dot product in a high dimensional space.

Nonlinear SVM: optimization

- formulation (Lagrangian Dual Problem)

maximize $\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$

such that $0 \leq \alpha_i \leq C$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

The form of the discriminant function is

$$g(x) = \sum_{i \in SV} \alpha_i K(x_i, x_j) + b$$

Performance

→ SVM work very well in practice
- the user must choose the kernel function and its parameters

→ They can be expensive in time and space for big data set
- The computation of the maximum-margin hyper plane depends on the square of the number of training ~~cases~~ cases
- we need to store all the support vectors

→ The kernel trick can also be used to do PCA in a much higher-dimensional space, thus giving a non-linear version of PCA in the original space

Multiclass classification

- SVMs can only handle two class outputs
- learn N SVM's
  - SVM 1 learns class 1 vs REST
  - SVM 2 learns class 2 vs REST
  - ⋮
  - SVM N learns class N vs REST
- Then to predict the output for a new input, just predict with each SVM and find out which one puts the prediction the furthest into the region

Chpt 2.4 Solution to the Dual Problem

SMO (sequential optimization)

Coordinate Ascent

$$\max_{\alpha} \; w(\alpha_1, \alpha_1 \ldots \alpha_n) \quad \text{constraints}$$

Loop until convergence {

for $i = 1$ to n {

$$\alpha_i = \arg\max_{\hat{\alpha}_i} w(\alpha_1, \hat{\alpha} \ldots \alpha_n)$$

- Coordinate ascent
  - ellipses are the contours of the function
  - At each step, the path is // to one of the axes.

- Sequential minimal optimization

constrained optimization:

$$\max_{\alpha} J(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (x_i x_j)$$

$$s.t. \quad 0 \le a_i \le C, \quad i = 1, \ldots, m$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0$$

- Choose a set of α_i's satisfying the constraints
- α_i is exactly determined by the other α's
- We have to update at least two of them simultaneously to keep satisfying the constraints.

The SMO algorithm

Repeat till convergence {

1. Select some pair α_i and α_j to update next (using a heristic that tries to pick the two that will allow us to make the biggest progress toward the global maximum.
2. Re-optimize $W(\alpha)$ with respect to α_i and α_j while holding all the other α_k's ($k \neq i, j$) fixed }

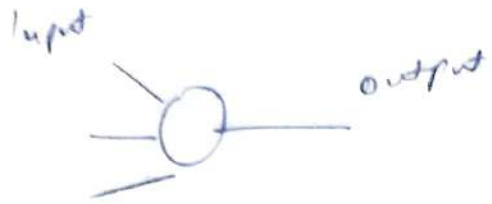
. The update to α_i and α_j can be computed very efficiently

Chp 25 Python Exercise on Svm

Chp 27 Neural Network Introduction

10's billion of neurons

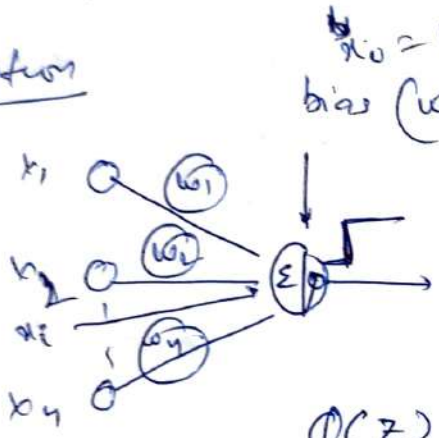
1. Massive Parallelism
2. Connectionism
3. Distributed Associative memory



Ann incorporate the two fundamental compo-
of biological neural net

① Nodes — Neuron
② weights — Synapses

Perceptron



$$\phi(z) = z$$

$$\phi(z) = \sqrt{\quad}$$

$$y = \sum_{i=0}^{n} w_i x_i$$

$$\phi_2(z) = \sqrt{\quad}$$

D

x_1	y_1
x_2	y_2
x_m	y_m

Perceptron training rule

$$w_i = w_i + \Delta w$$

$$\Delta w_i = \eta (y - \hat{y}) x$$

- Perceptron learning convergence if D is linearly separable.

Gradient descent can be used in the more general phase

$$E = \frac{1}{2} \sum_{d \in D} (y_d - \hat{y}_d)^2$$

$\phi_1(z) = z$

Stochastic gradient descent

$$E = \frac{1}{2} (y - \hat{y})^2$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\frac{\partial \epsilon}{\partial w_i} = x_{ij} (y_j - \hat{y}_j)$$

$\phi_3(z) = \sigma(z) = \frac{1}{1+e^{-z}}$ Sigmoid function.

$\phi_3'(z) = \phi_3(z) \cdot (1 - \phi_3(z))$

$E = \frac{1}{2} \sum_{d \in D} (y_d - \sigma(w \cdot x_d))^2$

$$\frac{\partial E}{\partial w_i} = \frac{1}{2} \sum_d \frac{\partial E}{\partial y_d} \cdot \frac{\partial y_d}{\partial w_i}$$

$\sigma'(w \cdot x_d) x_{i,d}$

$$\sum_d (y_d - \hat{y}_d) \cdot \frac{\partial}{\partial w_i} (y_d - \sigma(w \cdot x_d))$$

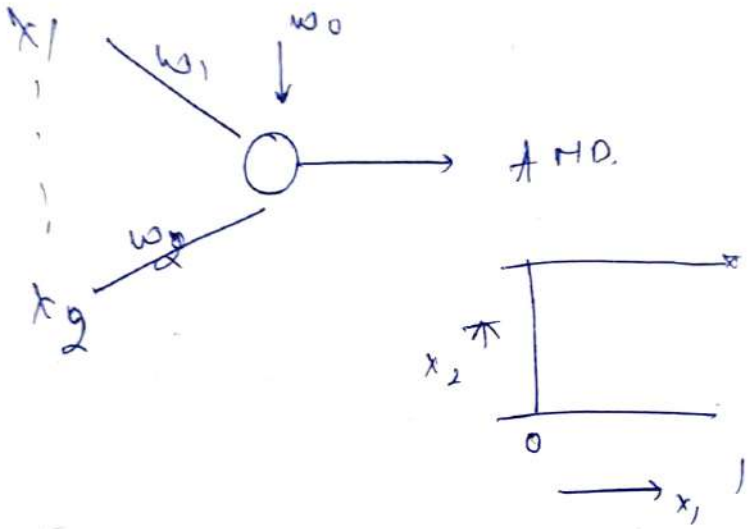
$$= \sum_d (y_d - \hat{y}_d) \, x_{i,d} \, \hat{y}_d (1 - \hat{y}_d)$$

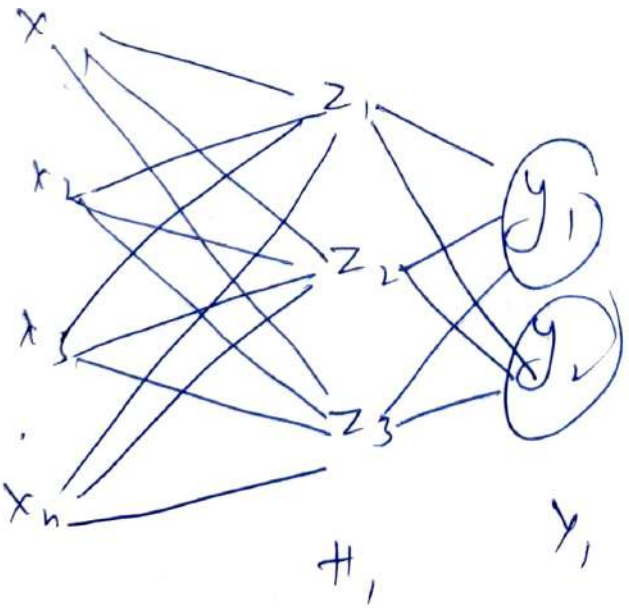
$$= \Delta w_i = \eta \sum_d (y_d - \hat{y}_d) \, \hat{y}_d (1 - \hat{y}_d) \cdot x_{i,d}$$

Chapter 28 Multilayer Neural Network

Limitation of Perceptrons

- Perceptrons have a monotonicity property:
  If a link has +ve weight, activation can only +ve as corresponding input value +ves. (irrespective of other input values)
- Can't represent when inputs interaction can cancel one another's effect (e.g. XOR)
- Can represent only linearly separable functions.

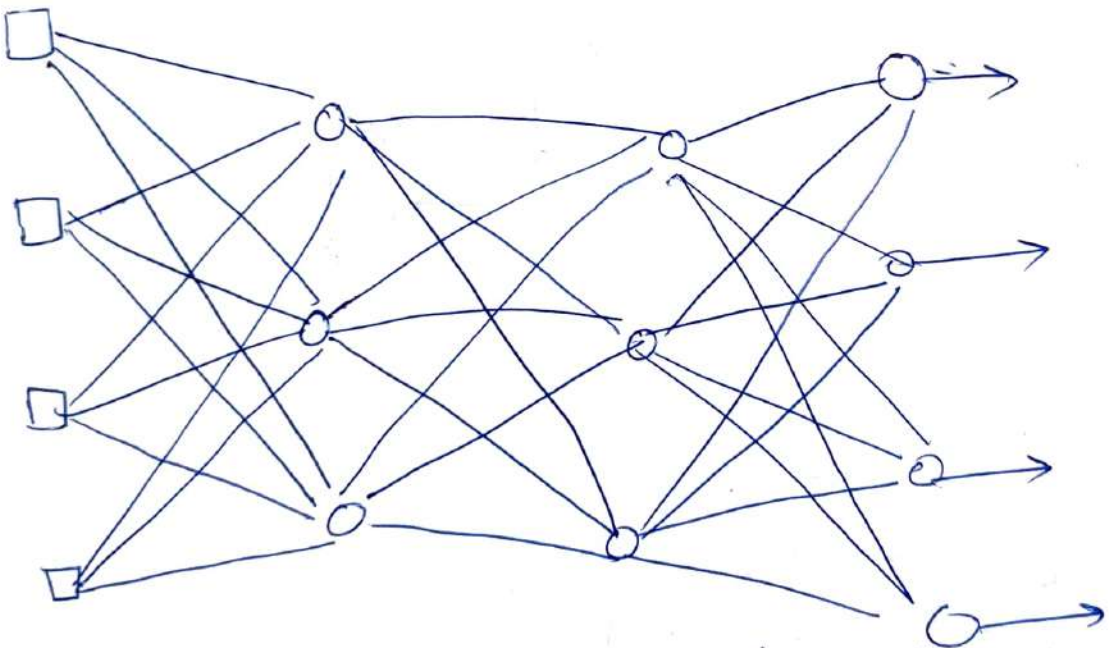




Power / Expressiveness of Multilayer Networks

- Can represent interactions among inputs
- Two layer networks can represent any Boolean functions and continuous functions (within a tolerance) as long as the no. of hidden units is sufficient and appropriate activation functions used
- Learning algorithms exist, but weaker guarantees than perceptron learning algorithm

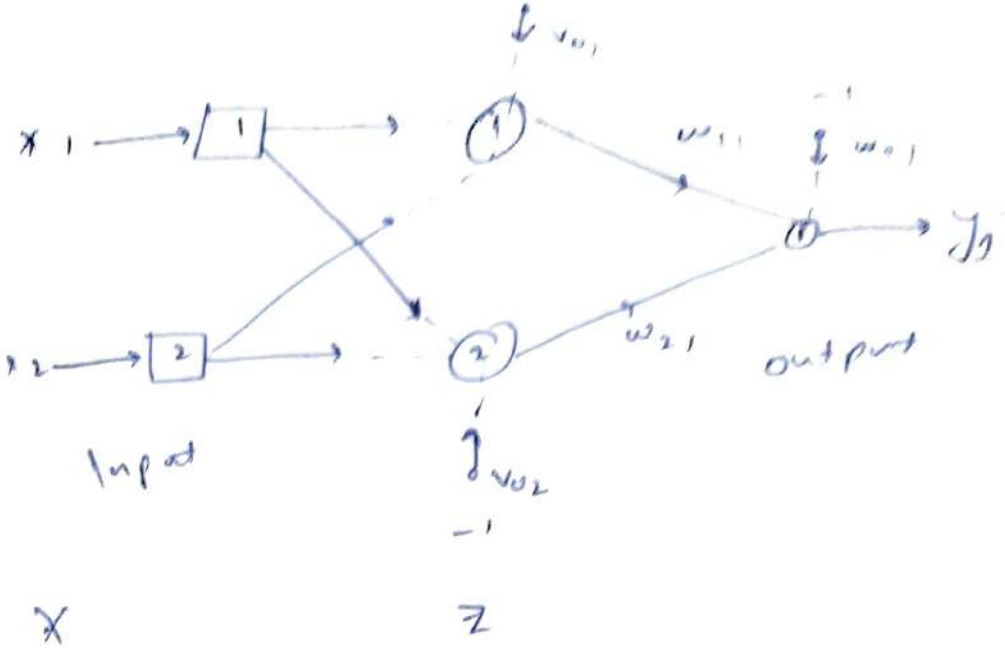
Two-Layer back-propagation



The back-propagation training algorithm

step 1 : Initialisation

set all the weights and thresholds levels of the network to random no. uniformly distributed inside a small range



Backprop

initialization

- forward computing
  - Apply an input vector x to input units
  - compute activation/output vector z on hidden layer

$$z_j = \varphi(\Sigma_i v_{ij} x_i)$$

- Compute the output vector y on output layer

$$y_k = \varphi(\Sigma_j w_{jk} z_j)$$

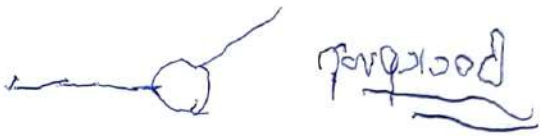
y is the result of the computation.

Learning for BP nets

- update of weights in w (b/w output and hidden layers)
  - delta rule

- Not applicable to updating V (b/w input and hidden)
  - don't know the target values for hidden units $z^1, z^2, \ldots, z^p$
- Solution: Propagate errors at output units to hidden units to drive the update of weights in V (again by delta rule) (error propagation learning)
- Error back propagation can be continued

do



Derivation

- For one output neuron, the error function is

$$E = \frac{1}{2}\sum_d (y - \hat{y})^2$$

- For each unit j, the output O_j is defined as:

$$O_j = \varphi(net_j) = \varphi\left(\sum_{k=1}^{n} w_{kj} O_k\right)$$

The input net to a neuron is the weighted sum of output O_k of previous n neurons

- finding the derivative of the error

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}}$$

$$\frac{\partial net_j}{w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{k=1}^{n} w_{kj} o_k \right) = o_i$$

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial}{\partial net_j} \varphi(net_j) = \varphi(net_j)(1 - \varphi(net_j))$$

$$\frac{\partial E}{\partial o_j} = \sum_i$$

$$\frac{\partial E(o_j)}{\partial o_j} = \frac{\partial E(net_1, net_2 \dots net_l)}{\partial o_j}$$

$$\frac{\partial \epsilon(o_j)}{\partial o_j} = \frac{\partial E(1, 2, \dots l)}{\partial o_j} = \sum_l \left(\frac{\partial \epsilon}{\partial net_l} \frac{\partial net_l}{\partial o_j} \right)$$

$$= \sum_l \left(\frac{\partial \epsilon}{\partial o_l} \frac{\partial o_l}{\partial net_l} \cdot w_{jl} \right)$$

$$\delta_j = \delta_j = \frac{\partial G}{\partial o_j} \frac{\partial o_j}{\partial net_j}.$$

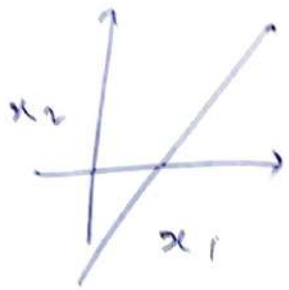
$(o_j - y_j) o_j (1 - o_j)$

$\sum_k (\delta_k w_{jk}) o_j (1 - o_j)$

Chapter 29 Neural Networks and Backpropagation Algorithm

Single layer Perceptron learn decision boundaries

$\phi\left(\sum_{i=0}^{n} w_i x_i\right)$



Chapter 30: Deep neural networks

Chapter 32 Introduction to Computation Learning theory

Goal of learning theory

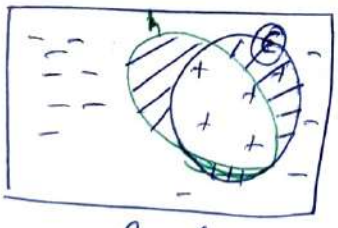
- To understand
  - what kind of tasks can learners learn
  - what kind of data is required for learnability
  - what are the requirements of learning algorithms?

- To develop and analyze models
- Develop algorithms that provable meet certain criteria
- Prove guarantees for successful algorithms

→ Learnable
  what tasks?
  what type of data
  Resource requirement - space, time
→ Develop algorithms
→ Prove guarantees

→ Design algorithms
→ Confidence → Generalization ability

PAC setting (models)
Probably
Approximately
correct



$c \oplus h$

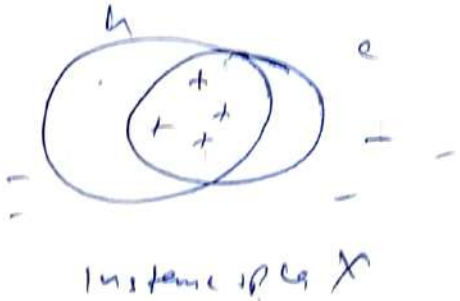
Probability $P^r(P(c \oplus h) \leq \varepsilon) \geq (1-\delta)$

$0 < \varepsilon < \frac{1}{2}$

$0 < \delta < \frac{1}{2}$

Prototypical Concept Learning Task

- Given:
  - Instances: e.g. $X = R^d$ or $X = \{0,1\}^d$
  - Distribution D over X
  - Target function c
  - Hypothesis space H
  - Training examples $S = \{(x_i, c(x_i))\}_{i=1}^{n}$ drawn from D.
- Determine
  - A hypothesis $h \in H$ s.t. $h(x) = c(x)$ for all x in S?
  - A hypothesis $h \in H$ s.t. $h(x) = c(x)$ for all x in X?
- An algorithm does optimization over S, finding hypothesis h.
- Goal: find h which has small error over D.



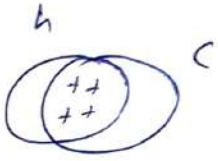
Instance Space X

Consistent hypothesis

Computational Learning theory

- Can we be ~~certai~~ certain about how the learning algorithm generalizes?
- We would have to see all the examples
- Inductive inference — generalizing beyond the training data is impossible unless we add more assumption (eg priors over H)

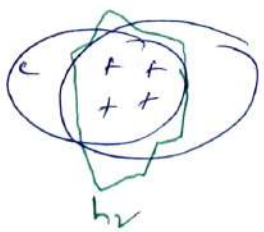
  we need a bias!



Instance space

function Approximation

- How many labeled examples in order to determine which of the 2^{2^n} hypotheses is correct one?
- All 2^n instances in X must be labeled!
- Inductive inference: generalizing beyond the training data is impossible unless we add more assumption (eg. bias)



$H = \{h : X \to Y\}$

$|H| = 2^{|X|} = 2^{2^n}$

N boolean features

2^N possible instance

h_i 2^{2^N} possible partitions of instance space

Error of a hypothesis

The true error of hypothesis, h, with respect to the target concept c and observation distribution D is the probability that h with misclassify an instance drawn according to D.

$$error_D(h) \Pr_{x \sim D}[c(x) \neq h(x)]$$

In a perfect world, we'd like the error to be 0

~~Bias~~ Bias: Fix hypothesis space H.

c may not be in H $\Rightarrow$ find h close to c

A hypothesis h is approximately correct if

$$error_D(h) \leq \varepsilon$$

h measure the error of h on S

$error_S(h)$ = True error of h

$error_S(h) = \Pr_{x \sim D}[c(x) \neq h(x)]$

$h \in H$. $c \notin H$

h close to c $P(c \oplus h)$

$error_D(h) \leq \epsilon$

↓

approximately correct

PAC model

- Goal: h has small error over D
- True error: $error_D(h) = P_r(h(x) \neq c^*(x))$
- How often $h(x) \neq c^*(x)$ over future instances drawn at random from D.
- But, can only measure:

  Training error: $error_s(h) = \frac{1}{m} \sum_i I(h(x_i) \neq c^*(x))$

  How often $h(x) \neq c^*(x)$ over training instances
- Sample complexity: bound $error_D(h)$ in terms of $error_s(h)$

- PAC learning concern efficient learning
- We would like to prove that
  - with high probability an efficient learning algorithm will find a hypothesis that is approximately identical to the hidden target concept
- We specify two parameters, ϵ and δ and require that with probability at least $(1-\delta)$, system learns a concept with error at most ϵ

Sample complexity for supervised learning

Theorem

$$m \geq \frac{1}{\epsilon}\left[\ln(|H|) + \ln\left(\frac{1}{\delta}\right)\right]$$

Labeled examples are sufficient so that with prob. $1-\delta$, all $h \in H$ with $error_D(h) \geq \epsilon$ have $error_s(h) > 0$

- Inversely linear in ϵ
- ~~Logarithmic~~ in $|H|$
- ϵ error parameter : D might place low weight on certain parts of the space
- δ confidence parameter : there is small chance the examples we get do not represent the distribution.

Theorem: $m \geq \frac{1}{\epsilon}\left[\ln|H| + \ln\left(\frac{1}{\delta}\right)\right]$ Labeled eg are sufficient so that with prob. $1-\delta$, all $h \in H$ with $error_D(h) \geq \epsilon$ have $error_s(h) > 0$

Proof: Assum k bad hypotheses $H_{bad} = \{h_1, h_2, \ldots, h_k\}$ with $err_D(h_i) \geq \epsilon$

- Fix h_i. Prob. h_i consistent with first training example is $\leq 1-\epsilon$. Prob. h_i consistent with first m training examples is $\leq (1-\epsilon)^m$.

- Prob. that at least one h_i consistent with m training examples

$$\leq k(1-\epsilon)^m \leq |H|(1-\epsilon)^m$$

- calculated value of m so that $|H|(1-\epsilon)^m \leq \delta$

- Use the fact that $1-x \leq e^{-x}$ sufficient to set

$$|H| e^{-\epsilon m} \leq \delta$$

Sample complexity — Finite hypothesis space Realizable case

PAC: How many examples suffice to guarantee small error whp.

Theorem

$$m \geq \frac{1}{\epsilon}\left[\ln(|H|) + \ln\left(\frac{1}{\delta}\right)\right]$$

Labeled examples are sufficient so that with prob. $1-\delta$, all $h \in H$ with $error_D(h) \geq \epsilon$ have $err_S(h) > 0$

Statistical learning way!

with probability at least $1-\delta$, all $h \in H$ s.t. $err_S(h) = 0$ we have

$$err_D(h) \leq \frac{1}{m}\left[\ln(|H|) + \ln\left(\frac{1}{\delta}\right)\right]$$

$$P(\text{Consist}(H_{bad}, D)) \leq |H| e^{-\epsilon m} \leq \delta$$

$$e^{-\epsilon m} \leq \frac{\delta}{|H|}$$

$$-\epsilon m \leq \ln\left(\frac{\delta}{|H|}\right)$$

$$m \geq \left[-\ln\frac{\delta}{|H|}\right]/\epsilon \quad \text{Chip inequality}$$

$$m \geq \left\lceil \frac{1}{\varepsilon} \ln \frac{|H|}{\delta} \right\rceil$$

$$m \geq \left(\ln|H| + \ln\frac{1}{\delta}\right)/\varepsilon$$

Chapter 33 Sample Complexity : Finite Hypothesis space

- For a single hypothesis to have misleading training error

$$Pr[error_D(f) \leq \varepsilon + error_D(f)] \leq e^{-2m\varepsilon^2}$$

- we want to ensure that the best hypothesis has error bounded in this way
  - so consider that any one of them could have a large error

$$Pr[(\exists f \in H)\ error_D(f) \leq \varepsilon + error_D(f)] \leq |H| e^{-2m\varepsilon^2}$$

- from this we can derive the bound for the no. of samples needed

$$m \geq \frac{1}{2\varepsilon^2}\left(\ln|H| + \ln\left(\frac{1}{\delta}\right)\right)$$

Sample complexity: finite hypothesis spaces

Consistent case

Theorem

$$m \geq \frac{1}{\epsilon}\left[\ln(|H|) + \ln\left(\frac{1}{\delta}\right)\right]$$

Labeled examples are sufficient so that with prob $1-\delta$, all $h \in H$ with $error_D(h) \geq \epsilon$ have $error_S(h) > 0$

Inconsistent case

What if there is no perfect h?

Theorem: After m examples, with probability $\geq 1-\delta$, all $h \in H$ have $|error_D(H) - error_S(h)| < \epsilon$ for

$$m \geq \frac{2}{2\epsilon^2}\left[\ln(|H|) + \ln\left(\frac{2}{\delta}\right)\right]$$

Sample complexity: example

C: conjunction of n Boolean literals. In C for

- Example?

$$|H| = 3^n$$

$$m \geq \frac{1}{\epsilon}(n \ln 3 + \ln(\frac{1}{\delta}))$$

- Concrete examples
  - $\delta = \epsilon = 0.05$, $n = 10$ gives 280 examples
  - $\delta = 0.01$, $\epsilon = 0.05$, $n = 10$ gives 312 examples
  - $\delta \delta = \epsilon = 0.01$, $n = 10$, gives 1560 examples
  - $\delta = \epsilon = 0.01$, $n = 50$ gives 5954 examples
- Result holds for any consistent learner, such as finds

H = Conjunction of Boolean literals.

$x_1, x_1, \ldots x_n$

$x_1 \wedge \bar{x}_1 \wedge x_3$

$x_1 \wedge x_1 \wedge \bar{x}_3$

$\bar{x}_1 \wedge x_3$

Concept learning

h = (?, cold, high, ?, ?, ?)

Indicates that Aldo enjoys his favorite sport on cold day with high humidity

most general hypothesis: (?, ?, ?, ?, ?, ?)

most specific hypothesis: (∅, ∅, ∅, ∅, ∅, ∅)

Find - S Algorithm

1. Initialize h to the most specific hypothesis in H
2. For each training instance x
   For each attribute constraint a_i in h
   If the constraint a_i in h is satisfied by x
   Then do nothing
   Else replace a_i in h by next more general
3. Output hypothesis h

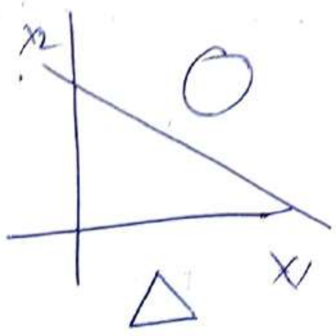
[illegible] Sample complexity of learning Arbitrary Boolean functions

$$m \geq \frac{1}{\epsilon}\left(\ln 2^{2^n} + \ln\left(\frac{1}{\delta}\right)\right) = \frac{1}{\epsilon}\left(2^n \ln 2 + \ln\left(\frac{1}{\delta}\right)\right)$$

Chapt 3? VC dimension

Sample complexity: Infinite hypothesis spaces

- Need some measure of the expressiveness of infinite hypothesis space
- The Vapnik-Chervonenkis (VC) dimension provides such a measure, denoted $VC(H)$
- Analogous to $\ln|H|$, there are bounds for sample complexity using $VC(H)$



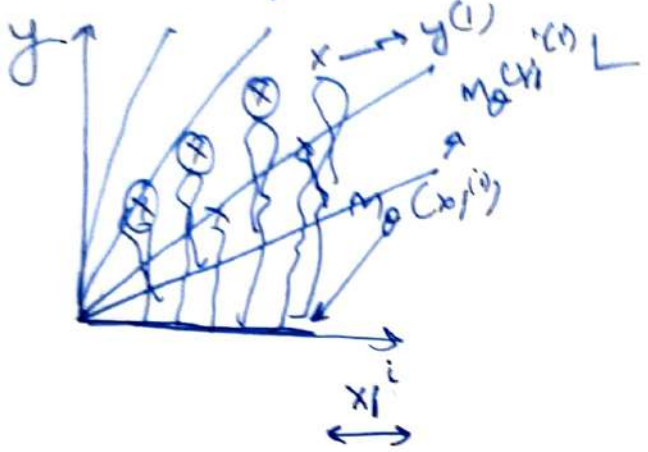
Loss L = $\underset{\theta_0, \theta_1}{\text{argmin}} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

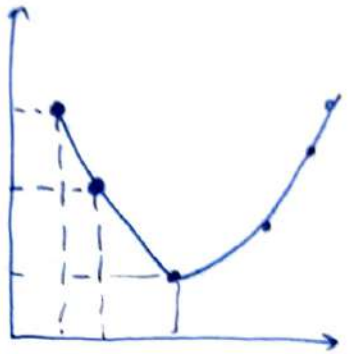
Least square cost function

Defining a Good model (selection of parameters)

How to select optimal θ_0, θ_1 ?

Curvature of the Loss function





In this curve there is single minimum point

$y = \theta_1 x$

If intercept is not there then line pass through origin

$\sum_{\theta} (h_\theta(x_1^{(i)}) - y^{(i)})^2$ (θ) we are changing the value of

~~too~~ 2D print of 3D graph is called contour

Generalized loss function (LF)

Non convex surface comprises of local minima and global minima

LF
- convex LF
- non convex LF

two ways to obtain optimal of θ_0 & θ_1

(a) Numerical method

(b) gradient descent

Numerical methods

$$L = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$= \frac{1}{2}\sum_{i=1}^{m}(\theta_0 + \theta_1 x_1^{(i)} - y^{(i)})^2 \quad \text{---(1)}$$

$$= \frac{1}{2m}\sum_{i=1}^{m}(\theta_0 + \theta_1 x_1^{(i)})^2 + (y^{(i)})^2 - 2(\theta_0 + \theta_1 x_1^{(i)} y^{(i)}]$$

$$L = \frac{1}{2m}\sum_{i=1}^{m}\left[\theta_0^2 + \theta_1^2 (x_1^{(i)})^2 + 2\theta_0\theta_1 x_1^{(i)} + (y^{(i)})^2 - 2\theta_0 y^{(i)} - 2\theta_1 x_1^{(i)} y^{(i)}\right]$$

$$\frac{\partial L}{\partial \theta_0}, \frac{\partial L}{\partial \theta_1} = ?$$

$$\frac{\partial L}{\partial \theta_0} = \frac{1}{2m}\sum_{i=1}^{m}\left[2\theta_0 + 2\theta_1 x_1^{(i)} - 2y^{(i)}\right]$$

$$\frac{\partial L}{\partial \theta_1} = \frac{1}{2m}\sum_{i=1}^{m}\left[2\theta_1 (x_1^{(i)})^2 + 2\theta_0 x_1^{(i)} - 2x_1^{(i)} y^{(i)}\right]$$

Equating the derivatives to zero to obtain values of θ_0 & θ_1 that corresponds to minimum of the loss function L

$$\frac{\partial L}{\partial \theta_0} = \frac{1}{2m}\sum_{i=1}^{m}\left[2\theta_0 + 2\theta_1 x_1^{(i)} - 2y^{(i)}\right] = 0$$

$$\theta_0 + \frac{2}{2m}\sum_{i=1}^{m}\theta_1 x_1^{(i)} - \frac{1}{m}\sum_{i=1}^{m} y^{(i)} = 0$$

$$\theta_0 = \frac{1}{m}\sum_{i=1}^{m} y^{(i)} - \frac{\theta_1}{m}\sum_{i=1}^{m} x_1^{(i)}$$

$$\hat{\theta}_0 = \bar{y} - \theta_1 \bar{x}_1$$

To ensure minima we have to check the 2nd derivative

$$\frac{\partial^2 L}{\partial \theta_0^2} = \frac{(m+1)}{2} > 0 \quad (\because \text{minima is guaranteed})$$

optimal value of $\hat{\theta}_1 = ?$

$$\frac{\partial L}{\partial \theta_1} = \frac{1}{2m} \sum_{i=1}^{m} \left[2\theta_1 (x_1^{(i)})^2 + 2\theta_0 x_1^{(i)} - 2 x_1^{(i)} y^{(i)} \right] = 0$$

$$\theta_1 \frac{1}{m} \sum_{i=1}^{m} (x^{(i)})^2 + \frac{1}{m} \sum_{i=1}^{m} (\bar{y} - \theta_1 \bar{x}_1)\, x_1^{(i)} - \frac{1}{m} \sum_{i=1}^{m} x_1^{(i)} y^{(i)} = 0$$

$$\theta_1 \frac{1}{m} \sum_{i=1}^{m} (x^{(i)})^2 + (\bar{y} - \theta_1 \bar{x}_1) \frac{1}{m} \sum_{i=1}^{m} x_1^{(i)} - \frac{1}{m} \sum_{i=1}^{m} x_1^{(i)} y^{(i)} = 0$$

$$\theta_1 \left[\frac{1}{m} \sum_{i=1}^{m} (x^{(i)})^2 - \bar{x}_1 \bar{x}_1 \right] = \frac{1}{m} \sum_{i=1}^{m} x_1^{(i)} y^{(i)} - \bar{y}\bar{x}_1$$

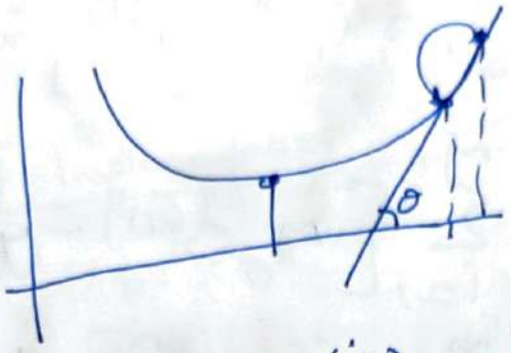
$$\hat{\theta}_1 = \frac{\frac{1}{m} \sum_{i=1}^{m} x_1^{(i)} y^{(i)} - \bar{y}\bar{x}}{\frac{1}{m} \sum_{i=1}^{m} (x^{(i)})^2 - \bar{x}_1 \bar{x}_1}$$

$\hat{\theta}_0 = \bar{y} - \theta_1 \bar{x}$

$\{\hat{\theta}_0, \hat{\theta}_1\} \rightarrow$ optimal values

Second Method

$$\theta_{new} = \theta_{old} - \alpha \frac{\partial L}{\partial \theta}$$



$$L = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$= \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x_1^{(i)} - y^{(i)})^2$$

2nd method.

$$\frac{\partial L}{\partial \theta_0} = \frac{1}{2m} \sum_{i=1}^{m} [2\theta_0 + 2\theta_1 x_1^{(i)} - 2y^{(i)}]$$

$$= \frac{1}{m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x_1^{(i)} - y^{(i)})$$

$$\frac{\partial L}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_0 := \theta_0 - \alpha \frac{\partial L}{\partial \theta_0} = \theta_0 - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

new old

$$\frac{\partial L}{\partial \theta_1} = \frac{1}{2m} \sum_{i=1}^{m} [2\theta_1 (x_1^{(i)})^2 + 2\theta_0 x_1^{(i)} - 2x_1^{(i)} y^{(i)}$$

$$\frac{\partial L}{\partial \theta_1} = \frac{1}{m}\left\{\sum_{i=1}^{m}\left[\theta_0 + \theta_1 x_1^{(i)}\right] x_1^{(i)} - x_1^{(i)} y^{(i)}\right\}$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left\{\left[\theta_0 + \theta_1 x_1^{(i)}\right] - y^{(i)}\right\} x_1^{(i)}$$

$$\frac{\partial L}{\partial \theta_1} = \frac{1}{m}\sum_{i=1}^{m}\left[h_\theta(x^{(i)}) - y^{(i)}\right] x_1^{(i)}$$

$$\theta_1 := \theta_1 - \frac{\alpha}{m}\sum_{i=1}^{m}\left[h_\theta(x^{(i)} - y^{(i)}\right] x_1^{(i)}$$

Gradient Descent Algorithm (single feature)

repeat till minimum is reached

$$\theta_0 := \theta_0 - \frac{\alpha}{m}\sum_{i=1}^{m}\left(h_\theta(x_0^{(i)} - y^{(i)}\right)$$

$$\theta_1 := \theta_1 - \frac{\alpha}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)} - y^{(i)}\right] x_1^{(i)}$$

Vectorization Techniques (Motivation)

Gradient descent Algorithm

repeat till maximum is reached

$$\theta_0 := \theta_0 - \frac{\alpha}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)} - y^{(i)}\right)$$

$$\theta_1 := \theta_1 - \frac{\alpha}{m}\sum_{i=1}^{m}\left[h_\theta(x^{(i)}) - y^{(i)}\right] x_1^{(i)}$$

input?

What is Vectorization Technique?

$$Z = \sum_{i=1}^{n} AB$$

$$Z = A^T B$$

$$A = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad B = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$$

MATLAB, Python → Library

Vectorized Implementation of Cost function

$$h_\theta(x^{(i)}) = \theta_0 + \theta_1 x_1^{(i)}$$

(Parameters)

Let $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} \in R^{2\times 1}$, $x^{(i)} = x_1^{(i)}$

$$X = \begin{bmatrix} -(x^{(1)})^T- \\ -(x^{(2)})^T- \\ -(x^{(m)})^T- \end{bmatrix} = \begin{bmatrix} x_1^{(1)} \\ x_1^{(2)} \\ \vdots \\ x_1^{(m)} \end{bmatrix} \in R^{m\times 1}$$

Design matrix

$$x_0 = [1 \; x] = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ 1 & x_1^{(3)} \\ \vdots & \\ 1 & x_1^{(m)} \end{bmatrix} \in R^{m\times 2} \rightarrow x^{0(2)} = [1 \; x_1^{(2)}]$$

$$\rightarrow x_2^{0(2)} = x_1^{(2)}$$

$$Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in R^{m\times 1}$$

$$L = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$L = \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x_1^{(i)} - y^{(i)})^2$$

$$L = \frac{1}{2m} \sum_{i=1}^{m} \left[\left[x^{o(i)} \theta - y^{(i)} \right]^2 \right]$$

(1×2) (2×1) (1×1)

Single vectorized

Double vectorized (No summation in eq. (A))

(A)

So we can write, $h_\theta(x) = \underset{1\times 2}{x^o} \underset{2\times 1}{\theta}$,

$$\text{Loss} = \frac{1}{n} (x^o\theta - y)^T (x^o\theta - y)$$

↳ similar to eqn. (A)

(1×m) (m×1)

$$= \frac{1}{m} \left[(x^o\theta)^T - y^T)(x^o\theta - y) \right]$$

$$= \frac{1}{m} \left[(\theta^T x^{oT} - y^T)(x^o\theta - y) \right]$$

$$= \frac{1}{m} \left[\theta^T x^{oT} x^o \theta - \theta^T x^{oT} y - y^T x^o \theta + y^T y \right]$$

(1×2)(2×m)(m×1) (1×m)(m×1) (1×1)

$$L = \frac{1}{m} \left[\theta^T x^{oT} x^o \theta - 2\theta^T x^{oT} y + y^T y \right]$$

$$\frac{\partial L}{\partial \theta} = \frac{2}{m} x^{oT} x^o \theta - \frac{2}{m} x^{oT} y = 0$$

$$x^{oT} x^o \theta = x^{oT} y$$

$$\hat{\theta} = (x^{oT} x^o)^{-1} x^{oT} y$$

Inverse takes lot of time

Disadvantage of normal method

If large no. of features are there, then $(x^{oT} x^o)$ will be a big square matrix,

$$X = \begin{bmatrix} -(x^{(1)})^T- \\ -(x^{(2)})^T- \\ -(x^{(3)})^T- \\ \vdots \\ -(x^{(m)})^T- \end{bmatrix} \Rightarrow \quad X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \cdots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \cdots & x_n^{(2)} \\ x_1^{(3)} & x_2^{(3)} & x_3^{(3)} & \cdots & x_n^{(3)} \\ \vdots & & & & \\ x_1^{(m)} & x_2^{(m)} & x_3^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}$$

Design matrix

$\mathbb{R}^{m \times n}$

$$X^0 = [1 \; X] = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \cdots & x_n^{(2)} \\ 1 & x_1^{(3)} & x_2^{(3)} & x_3^{(3)} & \cdots & x_n^{(3)} \\ \vdots & & & & & \\ 1 & x_1^{(m)} & x_2^{(m)} & x_3^{(m)} & \cdots & x_n^{(m)} \end{bmatrix} \in \mathbb{R}^{m \times (n+1)}$$

$$\text{Loss} = \frac{1}{m} (X^0 \theta - Y)^T (X^0 \theta - Y)$$

$m \times (n+1)$ $\quad (n+1) \times 1$ $\quad m \times (n+1)$ $\quad (n+1) \times 1$ $\quad m \times 1$

$1 \times m$ $\quad m \times 1$

→ multiple feature

Gradient Descend Algorithm (Multiple features)

Repeat till maximum is ~~fixed~~ achieved

$$\{ \quad \theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} \left[h_\theta(x^{(i)}) - y^{(i)} \right] x_j^{(i)}$$

$$\theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} \quad \{ j = 0 \text{ to } n \}$$

↓

features

j ...

Vectorized formulation

Using Normal update

$$\theta = (X^{0T} X^{0})^{-1} X^{0T} y$$

$(n+1) \times (n+1)$

↳ slow if n is large $O(n^3)$

Using gradient Descent

$$\theta := \theta - \alpha \frac{2}{M} X^{0T}(X^{0}\theta - y)$$

X^{0T}: $(n-1 \times m)$, X^{0}: $m \times (n+1)$, $(X^{0}\theta - y)$: $m \times 1$, result: $(n+1) \times 1$

Gradient Descent	Normal equation
– Required	– No α is requires
– Need iteration	– No iteration is requires
– Work well if features are large	– slow if feature are well

Any drawback with this costfunction?

$$\text{Loss}(L) = \underset{\theta_0, \theta_1}{\text{argmin}} \frac{1}{2m} \sum_{i=1}^{m} (M_\theta(x^{(i)}) - y^{(i)})^2$$

Least ~~fit~~ ~~square~~ overfit to outliers.

Solution Least ~~Absolute~~ Deviation

→ we use absolute error $|M_\theta(x^{(i)}) - y^{(i)}|$ instead of ~~square~~ version

– still treat negative & +ve error equally, but do not exaggerate the importance of large errors

Least ~~Absolute~~ Deviation cost

$$\text{Loss}(L) = \underset{\theta_0, \theta_1}{\text{argmin}} \frac{1}{2m} \sum_{i=1}^{m} |(M_\theta(x^{(i)}) - y^{(i)})|$$

$A - B \Rightarrow |A| + |B|$

Comparision b/w least square & Absolute Deviation

Weighted Regression (Dealing with Duplicates)

$$L = \frac{1}{2M} \sum_{i=1}^{M} (M_\theta(x^{(i)}) - y^{(i)})^2$$

$$= \frac{1}{2M} \left[(M_\theta(x^{(1)}) - y^{(1)})^2 + \underbrace{\cdots}_{\beta_1 \text{ term}} + (M_\theta(x^{(1)}) - y^{(1)})^2 \right]$$

$$+ \left[(M_\theta(x^{(2)}) - y^{(2)})^2 + \underbrace{\cdots}_{\beta_2 \text{ term}} + (M_\theta(x^{(2)}) - y^{(2)})^2 \right]$$

$$+ \left[(M_\theta(x^{(m)}) - y^{(m)})^2 + \underbrace{\cdots}_{\beta_m \text{ term}} + (M_\theta(x^{(m)}) - y^{(m)})^2 \right]$$

$$L = \frac{1}{\beta_1 + \beta_2 + \cdots + \beta_m} \sum_{i=1}^{M} \beta_i \left[(M_\theta(x^{(i)}) - y^{(i)})^2 \right]$$

$$= \frac{1}{(\beta_1 + \beta_2 + \cdots + \beta_m)} \sum_{i=1}^{M} \beta_i \left[(M_\theta(x^{(i)}) - y^{(i)})^2 \right]$$

$$= \frac{1}{\beta_1 + \beta_2 + \cdots + \beta_m} \sum_{i=1}^{M} \beta_i \left[x^{(i)}\theta - y^{(i)} \right]^2$$

$$20 \left[M_\theta(x^{(5)}) - y^{(5)} \right] + 10 \left[M_\theta(x^{(4)}) - y^{(4)} \right]$$

taking inverse will slow down the speed. see in next section

Also ill-posed

$$\frac{\partial L}{\partial \theta} = \frac{2}{m} x^T (x\theta - y)$$

update: $\theta := \theta - \alpha \frac{2}{m} x^T (x\theta - y)$ → faster speed compare to eqn. (5) and I is not there, → complete matrix multiplication.

Linear Regression with Multiple features

Feature = 4 $x_1\ x_2\ x_3\ x_4$ target (y)

sample = 24

$x_1^{(1)} = 2017$ | $x_3^{(5)} = 0$ there is no feature

$x_2^{(2)} = 1$ | $x_4^{(3)} = 5.3$

$$x^{(24)} = \begin{bmatrix} 2016 \\ 1 \\ 1.75 \\ 6.1 \end{bmatrix} \in R^{4\times 1}$$

Parameters

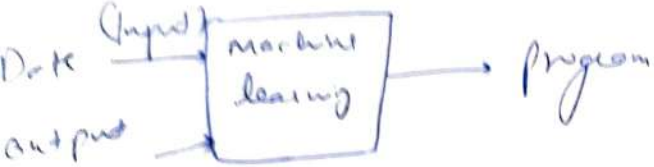
$$h_\theta(x^{(i)}) = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \theta_3 x_3^{(i)} + \cdots \theta_n x_n^{(i)}$$

Let $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \vdots \\ \theta_n \end{bmatrix} \in R^{(n+1)\times 1}$

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ x_3^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in R^{n\times 1}$$

Chp 1: Introduction to Machine Learning

It is the field of study that gives computer the capability to learn without being explicitly programmed



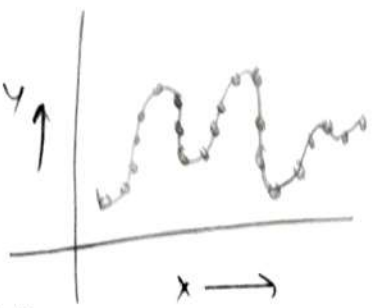
Error margin = Actual output − Ideal output

Chp 2: Classic & Adaptive machines

① Classic / Non adaptive system

Chp 3: Basic of training and testing

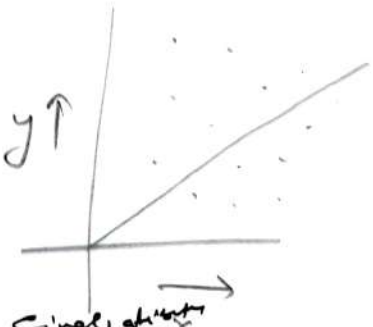
Chp 4: Overfitting and Underfitting



Four attributes
sphere
ploy
cot
Radius = 5cm

- Line try to cover every points.

Best fit



sphere
- Features are less
- No difference b/w ball and orange

Chp 5 - Feature Selection techniques

1. Filter methods
   - IG (Information Gain method) ; optimal features
   - Chi-Square test
   - correlation coeff

2. Wrapper methods
   - Recursive feature elimination
   - Genetic Algorithms

3. Embeded methods
   - Decision tree

Chp 6: One vs All — One vs One ($\frac{n(n-1)}{2}$)

Chp 7: Principle component Analysis

(It is in overfitting)

$$C = \begin{bmatrix} cov(x,x) & cov(x,y) \\ cov(y,x) & cov(y,y) \end{bmatrix}$$

$$cov(x,y) = \sum_{i=1}^{n} \frac{(x_i - \bar{x})(x - \bar{x})}{n-1}$$

x	$x-\bar{x}$	$(x-\bar{x})(x-\bar{x})$	y	$y-\bar{y}$	$(x-\bar{y})(y-\bar{x})$

x	y	$x-\bar{x}$	$y-\bar{y}$	$(x-\bar{x})(y-\bar{y})$

Chp 8:

$$C - \lambda I = 0$$

$$\begin{bmatrix} 0.6165 & 0.6154 \\ 0.6154 & 0.7165 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = 0$$

$$\begin{bmatrix} 0.6165-\lambda & 0.7161 \\ 0.6154 & 0.7161-\lambda \end{bmatrix}$$

$$= \lambda^2 - 1.3778\lambda + 0.0670 = 0$$

$$\lambda_1 = 0.0490$$

$$\lambda_2 = 1.3288$$

$$AV = \lambda V$$

$$\begin{bmatrix} 0.6165 & 0.7161 \\ 0.6154 & 0.7161 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = 0.0490 \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

$$0.6165\,x_1 + 0.7161\,y_1 = 0.0490\,x_1$$

$$0.6154\,x_1 + 0.7161\,y_1 = 0.0490\,y_1$$

$$= 0.5674\,x_1 = -0.6671\,y_1$$

$$x_1 = -1.0845\,y_1$$

$$y_1 = 1$$

$$\begin{bmatrix} -1.0845 \\ 1 \end{bmatrix} = \sqrt{2} \cdot 1.4142$$

$$\frac{1}{1.4142} \begin{bmatrix} -1.0845 \\ 1 \end{bmatrix} = \frac{-1.0845}{1.4142} = -0.7671$$

$$\frac{1}{1.4142} \begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix}$$

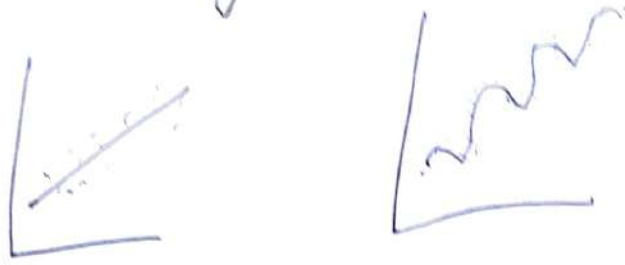
$$\lambda_2 = 1.3288$$

$$\frac{1}{1.4142} = 0.7071$$

$$\begin{bmatrix} 0.7071 \\ 0.7071 \end{bmatrix} = 1.0000$$

Ch 9 Regression Analysis

- Dependent & Independent variables
- Outliers
- Multicollinearity
- Underfitting and overfitting



Ch 10 :- Linear and Logistic Regression

Dependent variable all continuous in nature

→ Linear Relationship

→ 1 → I, & I → D

→ I → D & D → I

multiple LinaR

→ $y = \beta_1 + \beta_1 x_2 + \dots \beta_k \beta_k + \epsilon$

- Dependent variable is binary
- 1 (True) 0 (False)
- Goal is to find the best fitting model for I & D variable Relationship
- Independent variable can be continuous & binary

→ Also called Logit R

→ Used in Machine learning

→ Deals with probability to measure the relation b/w dependent & Independent variable

Ch 11 : Confusion matrix

Actual \ Predicted	No	Yes	
No	TN 50	FP 10	60
Yes	FN 5	TP 100	105
	55	110	

Recall = $\frac{TP}{\text{actual yes}}$

$= \frac{100}{105} = 0.91$

Accuracy $= \frac{TP + TN}{Total} = \frac{(100 + 50)}{165}$

$= 0.91$

Error rate = 1 − accuracy

or

$\frac{FP + FN}{Total} = 0.09$

Precision $= \frac{TP}{\text{Predicted yes}} = \frac{100}{110} = 0.63$

Ch13 Curse of dimensionality

Ch14 Managing missing values (features)

one hot coding — Dummy coding — Effect coding

after threshold — encoding

Ch15 Regression

Ch16 Logistic regression

$y = \frac{1}{1 + e^{-x}}$ sigmoid function

30 – 50 data points

chp 17 Polynomial Regression

$$y = \alpha_0 + \alpha_1 x_1$$

$$y = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 + \alpha_4 x_4$$

$$\boxed{y = \alpha_0 + \sum_{i=1}^{m} \alpha_i x_i}$$

0 degree poly : y = constant

1 degree poly : $y = mx + c$

2 degree poly : $y = ax^2 + bx + c$

$$y = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3 + \dots + \alpha_n x^n$$

$$\boxed{y = \alpha_0 + \sum_{i=1}^{m} \alpha_i x_i + F_p}$$

chp 18 Ridge Regression

Regularisation
- Ridge
- Lasso

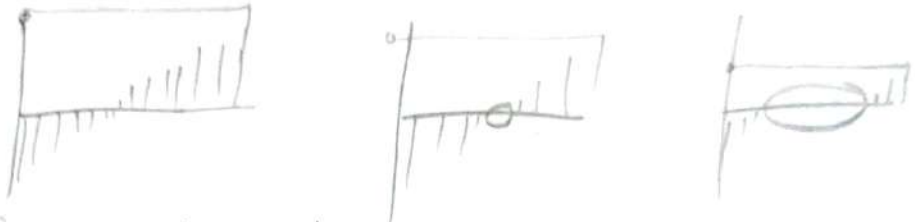
Ridge R = Loss + $\alpha \|w\|^2$ (penalty)

Ridge R = Loss + $\alpha \|w\|^2$ penalty

Lasso Regression

Lasso = Loss + α ||w||
Penalty

||w|| = |w1| + |w2| + |w3| + ... + |wn|



magnitude of coefficient exactly zero
— feature selection

Chp 20: Elastic Net regression

hybrid version of Ridge and Lasso

Ridge R = Loss + $\alpha ||w||^2$

Lasso R = Loss + $\alpha ||w||$

Elastic Net R = Loss + $\alpha_1 ||w||^2 + \alpha_2 ||w||$

Chp 21: Bayes theorem

$$P(A/B) = \frac{P(B/A) \cdot P(A)}{P(B)}$$

P(A/B): Position
P(B/A): Likelihood
P(A): Prior
P(B): marginal

$$P(A/B) = \frac{P(B|A)\,(A)}{P(B)}$$

Chp 22 Naive Bayes Classifier Algorithm

$$P(A/B) = \frac{P(B/A) \cdot P(A)}{P(B)}$$

37

Chp23 Back propagation of Error

Stanford lecture 4.

$s = f(x, w) = w_x$ score function

$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$ SVM loss

$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k w_k^2$ data loss + regularization

want $\nabla_w L$

$$\frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

$\sigma(x) = \frac{1}{1 + e^{-x}}$ sigmoid function

$\frac{e^{-x}}{(1 + e^{-x})^2}$

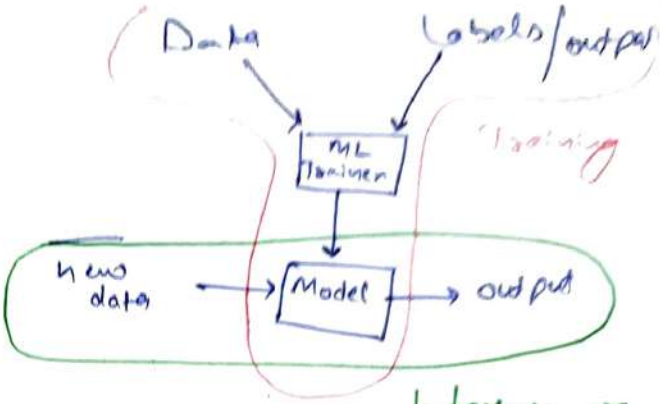
Always check the gradient with respect to data should have the same shape

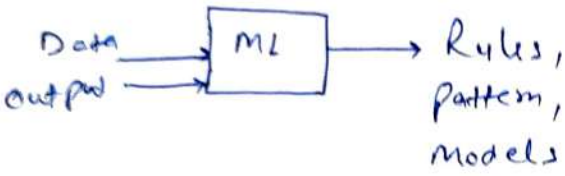
Lecture 2.

Introduction to tensorflow
Dr. Ashish Tendulkar (Google)

Add two no.
a & b

func f(a, b)
return (a + b)

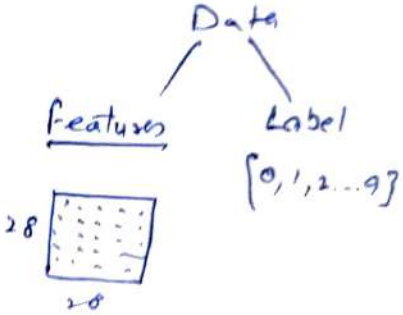




1. Data :

eg Handwritten image; $x^{(i)}$

$D = \{ x^{(1)}, x^{(2)}, \ldots x^{(m)} \}$



Features → House prize prediction
- # bedrooms $x_1^{(i)}$
- Area (sq. feet) $x_2^{(i)}$
- Distance from school $x_3^{(i)}$

$x^{(i)} : x_1^{(i)}, x_2^{(i)} \ldots x_n^{(i)}$

Label → Prize of House $y^{(i)}$

$Data = \{ (x^{(i)}, y^{(i)}) \}_{i=1}^{n}$

Label

Present → supervised Learning

absent → unsupervised Learning

Label

Discrete → Classification

continuous → Regression

Feature : # bedrooms, area

(i) Numeric feature

(ii) categorical

city : ["Mumbai", "Delhi", "chennai"]

	f_ Mumbai	f_ Delhi	f_ chennai
Mumbai	1	0	0
Delhi	0	1	0
chennai	0	0	1

One-hot-encoding

Lecture 3 :- Dr. Ashish tendulkar (Google)

(1) Data preprocessing

x_j : mean : μ_j

std. dev. : σ_j

$\frac{x_j - \mu_j}{\sigma_j}$ Z-score : -3 to +3

min_j, max_j

$\frac{x_j - min_j}{max_j - min_j}$ 0 to 1

log transformation

Model

(1) Linear regression

m- features : $x_1, x_2, \ldots, x_m$

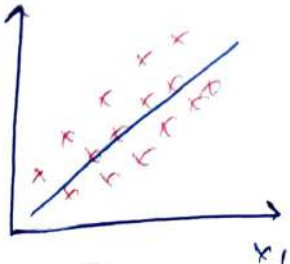
$y = b + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n$

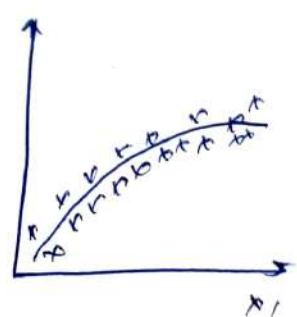
$(b, w_1, w_2, \ldots w_n)$ ← Parameters

$y = mx + c$

$y = b + w_1 x_1 + w_2 x_2^2 + w_3 x_1^3$

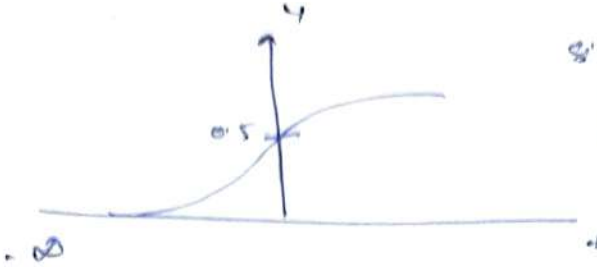
$y = (b) + (w_1) x_1$





(2) Logistic regression:

$z = b + w_1 x_1 + w_2 x_2 + \dots + w_m x_m$



Sigmoid function

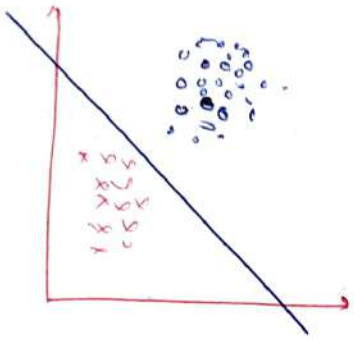
$$\sigma(z) = \frac{1}{1+\exp(-z)}$$

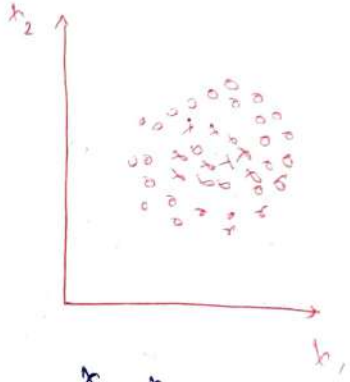
$z = 0, \ \sigma(z) = 0.5$

$z = \infty \rightarrow \sigma(z) \rightarrow 1$

$z = -\infty \leftarrow \sigma(z) \rightarrow 0$

$$P_\theta(y^{(i)} = 1 \mid x^{(i)}) = \frac{1}{1+\exp[-[b + w_1 x_1 + \dots + w_m x_m]]}$$

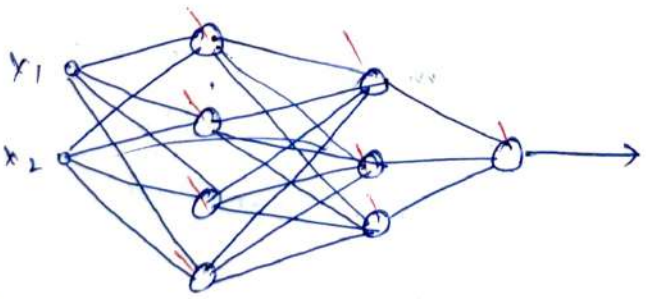




x_1, x_2

$x_1^2, x_2^2, x_1 x_2$

Feedforward neural network

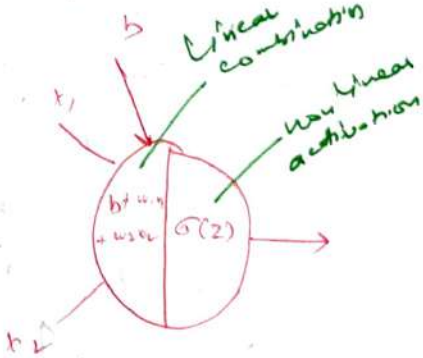


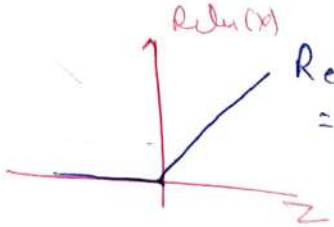
Input Layer

Hidden Layers

Output Layer

features =





$= \begin{cases} z \text{ if } z > 0 \\ 0, \text{ otherwise} \end{cases}$

Lecture 4 Dr. Ashish tendulkar Loss function.

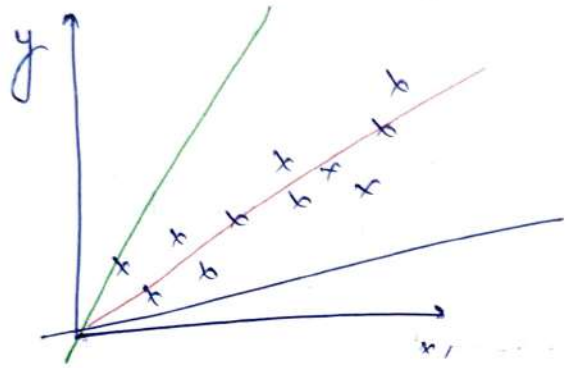
Linear regression $b + w_1 x_1 + w_2 x_2 + \dots + w_m x_m$

$$h_{w,b}(x) = b + \sum_{i=1}^{m} w_i x_i$$

Logistic regression

$$P_\theta(y = 1/x) = \frac{1}{1 + \exp(-z)}$$

$$z = b + w_1 x_1 + w_2 x_2 + \dots + w_n x_n$$



$$y = b + w_1 x_1$$

$$y = w_1 x_1$$

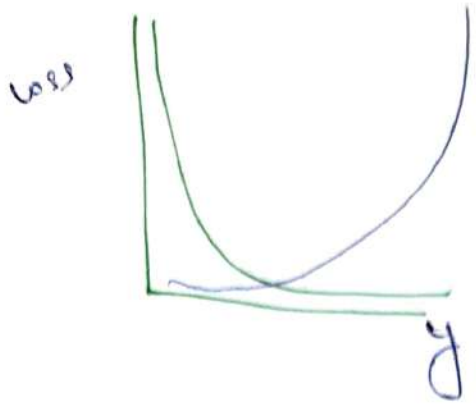
Loss function

$$J(w,b) = \frac{1}{2} \sum_{i=1}^{n} \left(\underbrace{h_{w,b}(x^{(i)})}_{\text{actual}} - y^{(i)} \right)^2$$

(predicted)

$$= \frac{1}{2} \sum_{i=1}^{n} \left([b + w_1 x_1^{(i)} - y^{(i)} \right)^2$$

actual (y)	Predicted ($\hat{y}$)	
1	0	error
0	1	error
1	1	ok (no-error)



if $y=1$ $-y\log(p)$

$y=0$ $-(1-y)\log(1-p)$

cross entropy loss:

$= -y\log(p) - (1-y)\log(1-p)$

for $y=1$

$= -\log(p)$

for $y=0$

$-\log(1-p)$

Lecture 5 : Gradient Descent

Loss function

$$J(w,b) = \frac{1}{2}\sum_{i=1}^{n}\left(h_{w,b}(x^{(i)}) - y^{(i)}\right)^2$$

(predicted value, actual value)

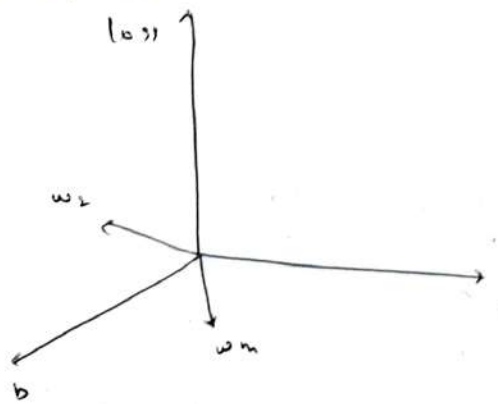
$y = b + w_1x_1 + w_2x_2 + \ldots + w_mx_m$

m - features

(m+1) parameters

$(b, w_1, w_2, \ldots, w_m)$

$(b, \vec{w})$



Data

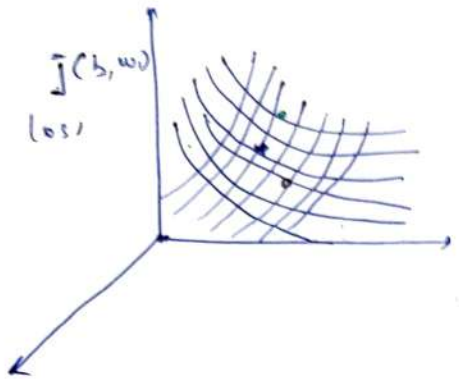
$D = \{(x^{(i)}, y^{(i)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(n)}, y^{(n)})\}$

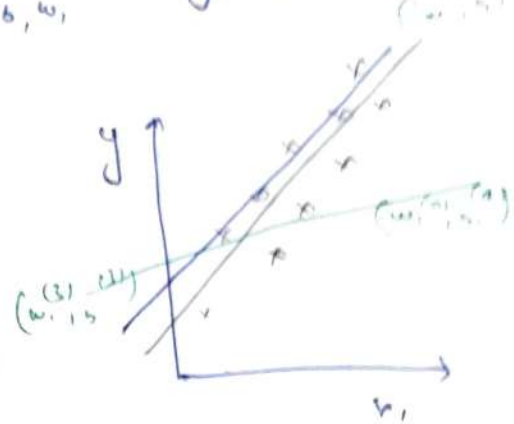
$= \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$

$x^{(i)} : x_1$, $y^{(i)} \in \mathbb{R}$

Model: $y = b + w_1 x_1$

loss : $J(b, w_1) = \frac{1}{2} \sum_{i=1}^{n} (h_{b,w_1}(x^{(i)}) - y^{(i)})^2$





Learning problem

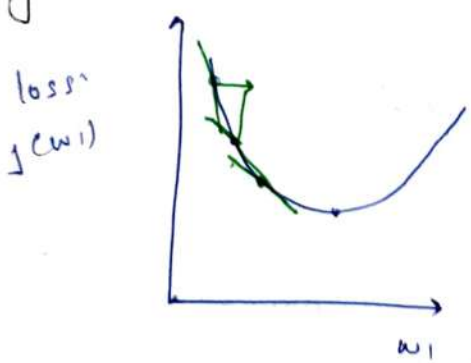
Find w, b such that loss is minimized

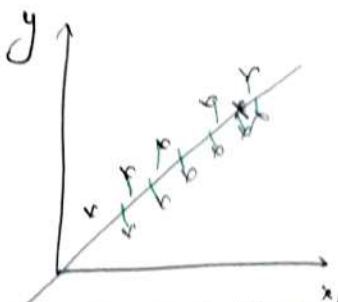
$J(w, b)$

$y = b + w_1 x_1$; $b = 0$

$y = w_1 x_1$; Find out w_1 s.t. $J(w_1, b)$ is minimized

"Gradient Descent"





(0) Randomly initialize w_1

(1) Loss $J(w_1, b)$ calculation - Predictions

(2) Gradient calculation

Learning rate α

(iv) $w_1^{(new)} = w_1^{(old)} - \alpha \cdot \text{gradient}$ | Steps (shortcut steps)

Gradient Descent $b, w_1, w_2, \ldots w_n$

(i) Randomly initialize $y = b + \sum_{i=1}^{m} w_i x_i$

$b, w_1, w_2, \ldots w_m$ $J(b, w) = \frac{1}{2} \sum_{i=1}^{n} (h_{w,b}(x^{(i)}) - y^{(i)})^2$

(ii) Repeat until convergence

(iii) Predict $\hat{y}^{(i)}$ for each data point in training

(iv) Calculate loss $J(b, w)$

(v) Calculate gradient of $J(b, w)$

(vi) $b^{(new)} = b^{(old)} - \alpha \cdot \text{gradient } b$

$w_1^{(new)} = w_1^{(old)} - \alpha \cdot \text{gradient } w_1$

$w_n^{(new)} = w_n^{(old)} - \alpha \cdot \text{gradient } w_n$

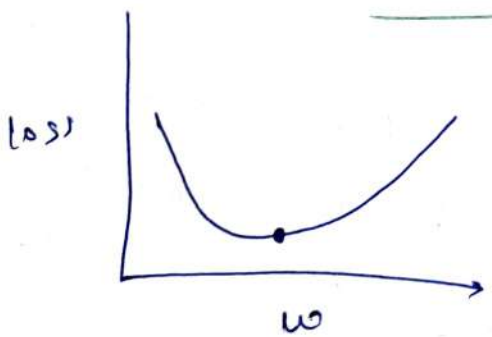
(vii) Update

$b, w_1, w_2, \ldots, w_n$

simultaneously

go to step (ii)

Lecture 6: Gradient descent Variation



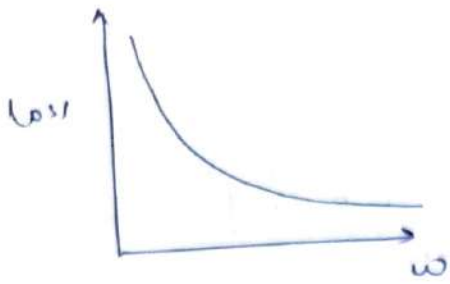
(i) No change in parameter value

gradient = 0

$w_1^{(new)} = w_1^{(old)} - \alpha \cdot \text{gradient}$

(ii) No. of steps

Gradient Derivation of Loss function.



$$J(w,b) = \frac{1}{2}\sum_{i=0}^{n} (h_{w,b}(x^{(i)} - y^{(i)})^2$$

$w \rightarrow w_1, w_n \cdots w_m$

$$\frac{\partial}{\partial w_1} J(w,b) = \frac{\partial}{\partial w_1} \frac{1}{2} \sum_{i=0}^{n} (h_{w,b}(x^{(i)} - y^{(i)})^2$$

$$= \frac{1}{2} \cdot \sum_{i=1}^{n} h_{w,b}(x^{(i)} - y^{(i)}) \frac{\partial (h_{w,b}(x^{(i)} - y^{(i)})}{\partial w_1}$$

$$b + \sum_{j=1}^{n} w_j \cdot x_j^{(i)}$$

$$= \sum_{i=1}^{n} (h_{w,b}(x^{(i)} - y^{(i)})$$

$$\frac{\partial}{\partial w_1} (b + w_1 b_1 + w_2 x_2 \cdots w_n x_n)$$

$$= \sum_{i=1}^{n} (h_{w,b}(x^{(i)} - y^{(i)}) \cdot x_i$$

$$\frac{\partial}{\partial w_j} J(w,b) = \sum_{i=1}^{n} (h_{w,b}(x^{(i)} - y^{(i)}) \cdot x_j$$

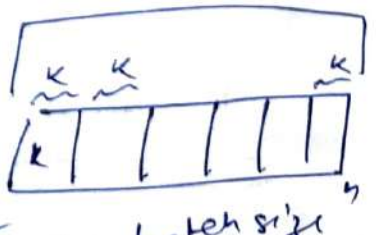
$$\frac{\partial}{\partial b} J(w,b) = \sum_{i=1}^{n} (h_{w,b}(x^{(i)} - y^{(i)}) \cdot 1$$

$$\frac{\partial}{\partial w_j} J(w,b) = \sum_{i=1}^{n} (h_{w,b}(x^{(i)} - y^{(i)}) \cdot x_j^{(i)}$$

Batch gradient descent "n" data points

"k" points

$$\sum_{i=1}^{k} (h_{w,b}(x^{(i)} - y^{(i)}) x_j^{(i)}$$



k: batch size
mini-batch grad.

$$1 \text{ epoch} = \frac{n}{k} \text{ batch iterations}$$

$k = 1$: stochastic G.D

SGD — G.D

1 — n

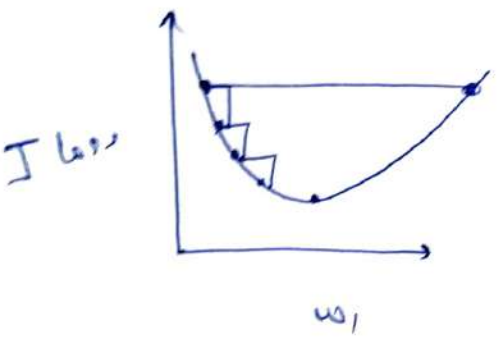
(i) Draw a batch of k examples $(x^{(i)}, y^{(i)})$

(ii) Randomly initialize parameters

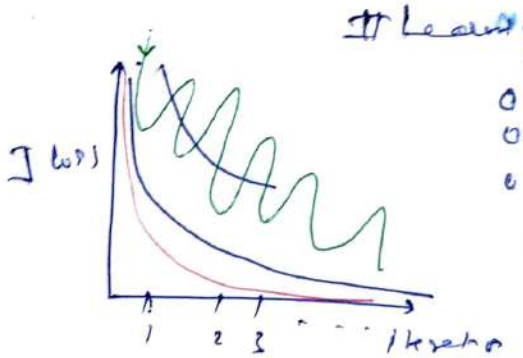
(iii) Repeat until convergence

- obtain prediction from model
- calculate loss on the batch
- calculate gradient of loss w.r.t parameters
- $w^{(new)} = w^{(old)} - \alpha \cdot \text{gradient}$
- update simultaneously

How do I know if my model is learning?



Learning curve



0.001
0.01
0.1
1

Learning rate:

(1) Small

(2) Larger

(3) Too Large

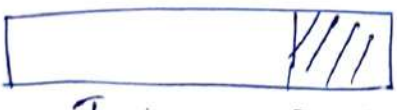
$w^{(new)} := w^{(old)} - \alpha \cdot \text{gradient}$

Lecture 7: Model selection and evaluation

Regression

MSE

MAE

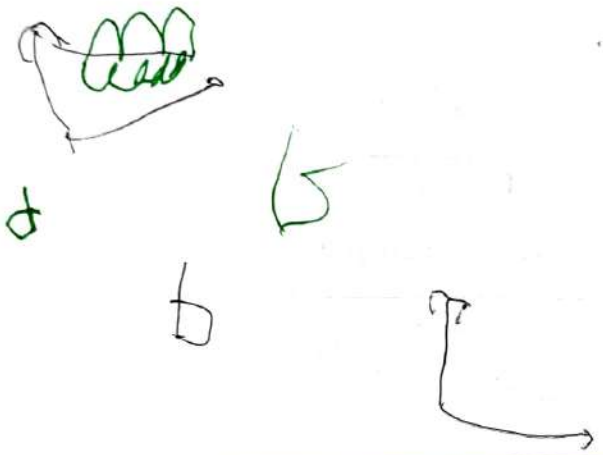


Classification

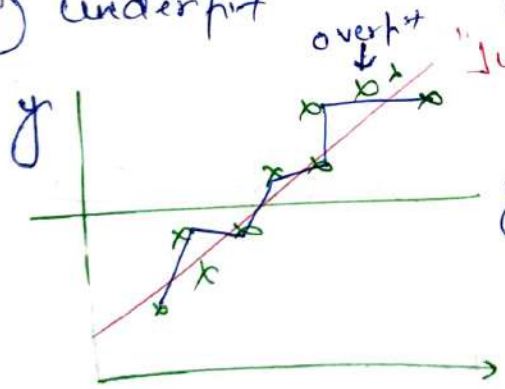
Precision, recall - f-1 ~~score~~ score

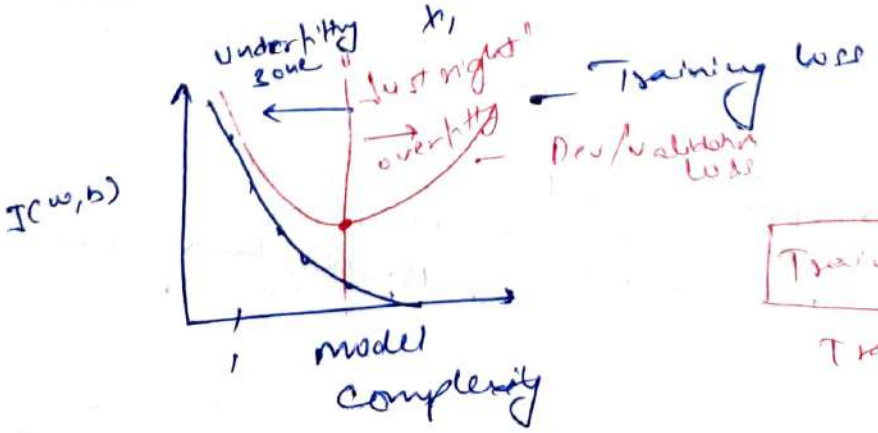
Accuracy

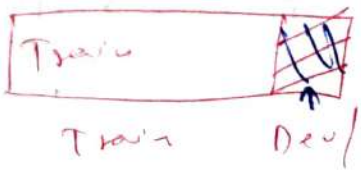
ROC
PR-curves



(i) underfit (ii) just right (iii) overfit







Fixing underfitting

(1) x_1, x_2

$x_1^2, x_2^2, x_1 x_2$

(2) Reduce λ to fix underfitting

Fixing overfitting

(1) Get more data

(2) Reduce model complexity

"Regularization"

$J(w,b) + \lambda$ model complexity

L_2 - regularization $\sum_{i=1}^{m} \|w_i\|_2 = \sum_{i=1}^{m} w_i^2$

L_1 - regularization $\sum_{i=1}^{m} \|w_i\|_1$

$= \sum_{i=1}^{m} |w_i|$

	More Data	More complexity
underfit	✗	Decrease more complex
overfit	✓	Increase less complexity

Confusion matrix

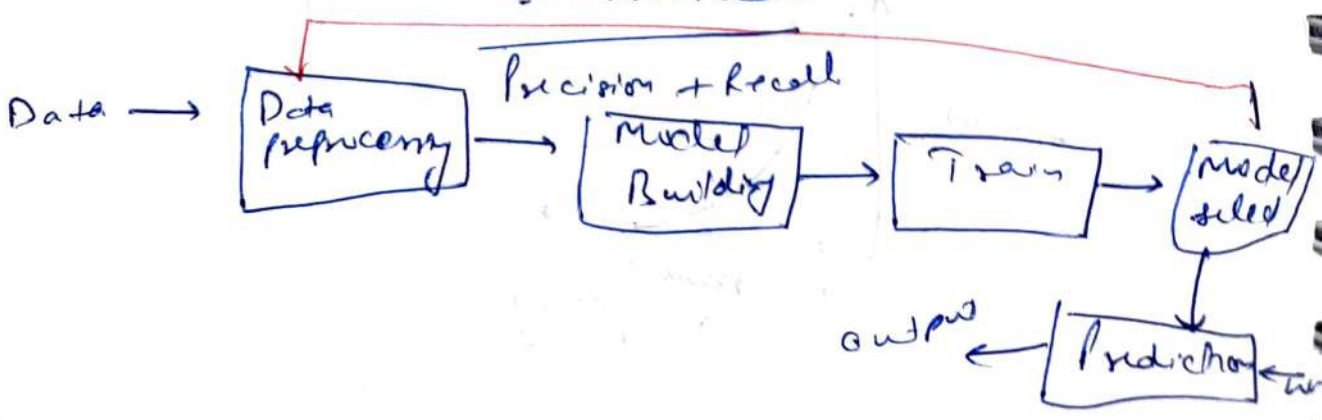
Actual \ Predicted	+1	-1
+1	TP	FN
-1	FP	TN

Accuracy $= \frac{TP+TN}{TP+FN+FP+TN}$

Precision $= \frac{TP}{TP+FP}$

Recall $= \frac{TP}{TP+FN}$

F_1-score $= 2 \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$



(0) Training data: $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$

(1) Model —— $\hat{y} = b + \sum_{j=1}^{m} w_j x_j$

(2) Cost ←—— $\frac{1}{n}\sum_{i=1}^{n}$ (pred − actual)^2

(3) Train —— G.D → Batch
→ Mini batch

(4) Evaluate —— MSE / MAE
Linear regression

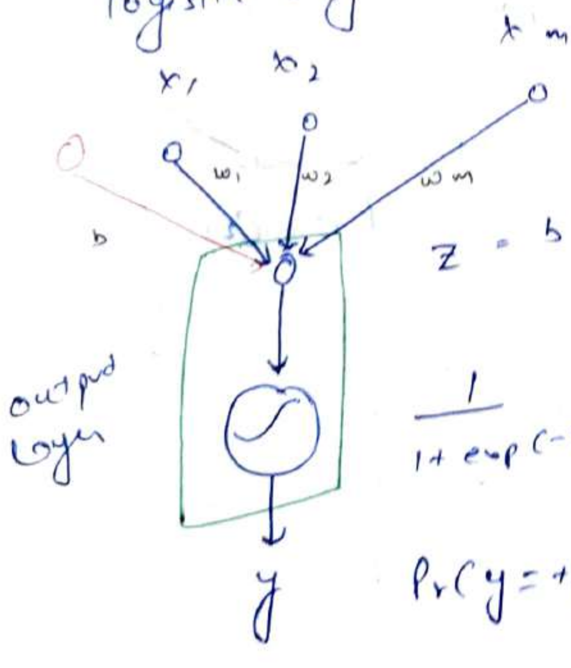
Lecture 8: Machine learning visualization

"A Neural Network (playground)" google / search

Tinker With a Neural Network

Lecture 9: Deep Learning Refresher

logistic regression



$Z = b + w_1 x_1 + w_2 x_2 + \ldots + w_m x_m$

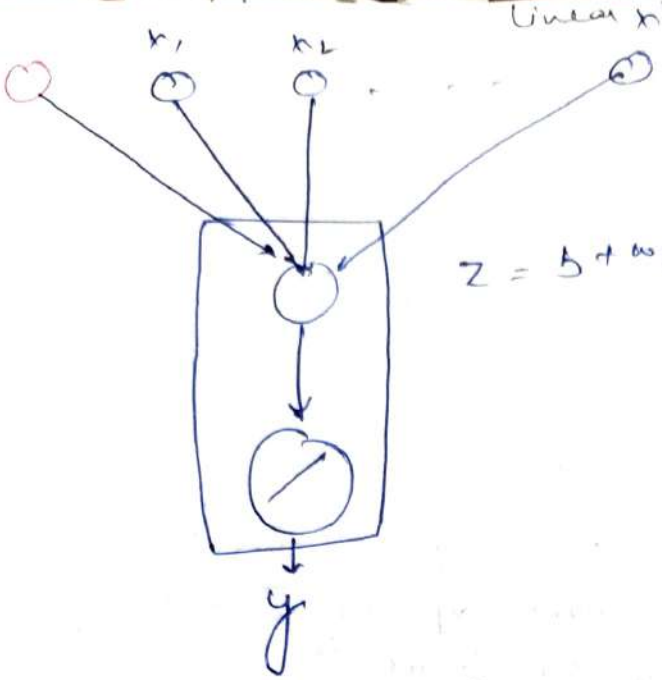
$\frac{1}{1 + \exp(-z)}$

$Pr(y = +1 \mid x)$

$x^{(i)}$

m: # of features

m+1 nodes (Input)



Linear regression

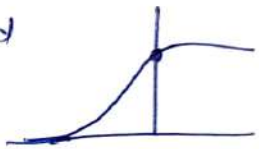
Input layer

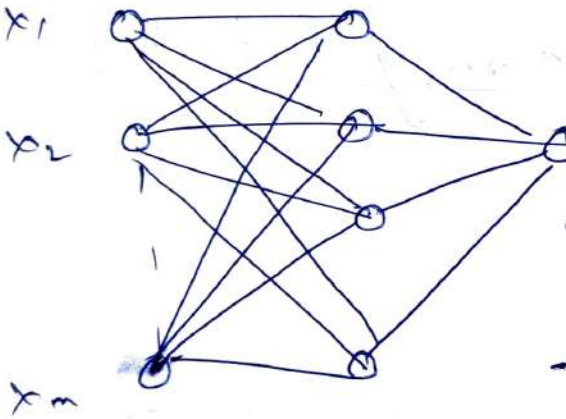
$z = b + w_1 x_1 + w_2 b_2 + \dots + w_m b_m$

output layer

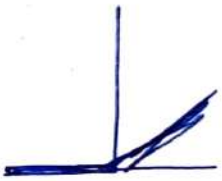
Building blocks of DL

- sigmoid





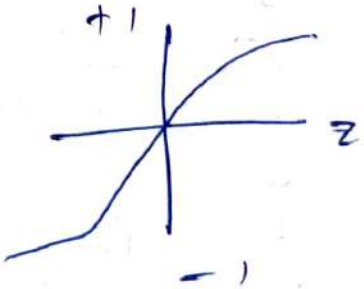
- Relu

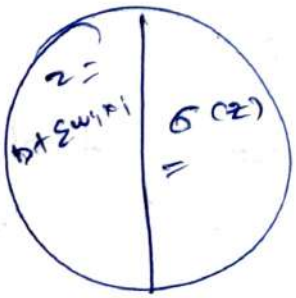


$\text{Relu}(z) = \begin{cases} z & z > 0 \\ 0 & \text{otherwise} \end{cases}$

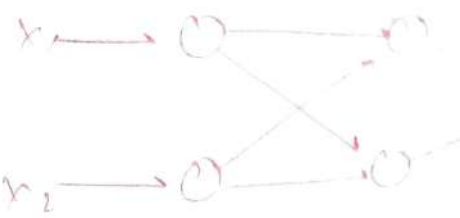
Input Hidden

- Tanh





Relu

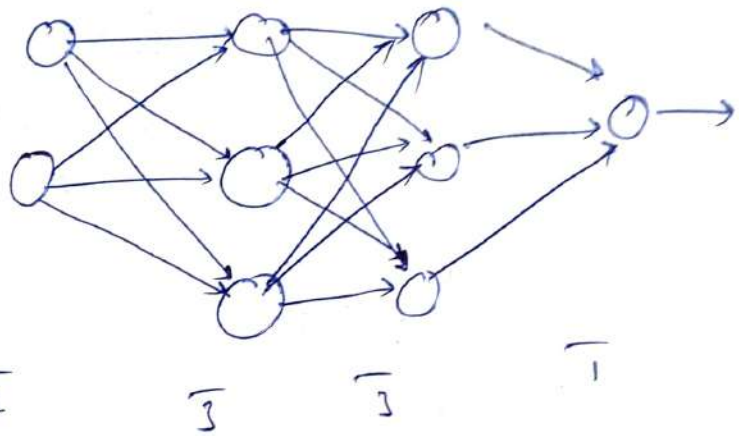


$$z_1 = relu(b + w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2)$$

$$z_2 = relu(b + w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2)$$

Feed forward NN

Model :-



Loss : MSE for regression.

Lecture 10 Introduction to Tensors

Scalar

```
x = np.array(10)
```

Vector 1D tensor

```
b = np.array([4, 4, 5, 9])
x.shape
```

Matrix (2D tensor) [[1,2,3],
[4,5,6],
[7,8,9]]

x

3D tensor x = np.array([[[1,2,3],
[4,5,6],
[7,8,9]]
[[1,2,3],
[4,5,6],
[7,8,9]]
[[1,2,3],
[4,5,6],
[7,8,9]]])

x.shape:
(3,3,3)

x.type:

Tensor in MNIST datasets

from __future__ import absolute_import, division, print_function, unicode_literals

! pip install tensorflow==2.0.0-beta1
import tensorflow as tf

[ ] mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

- selecting a single datapoint
x = x_train[0]
x_1

Select multiple

Select data point from 10 to 100 (100 is not included)

x_train_slice = x_train[10:100]

print(x_train_slice)

Here we explicitly select the first axis

x_train_slice = x_train[10:100, :, :]

Here we explicitly select the remaining two axes to get a 3-D slice

x_train_slice = x_train[10:100, 0:28, 0:28]

x_train_bs_slice = x_train[:, 14:, 14:]

Data batches

first batch - first 128 examples. Each batch has 128 examples

batch_1 = x_train[:128]

Next batch - next 128 examples

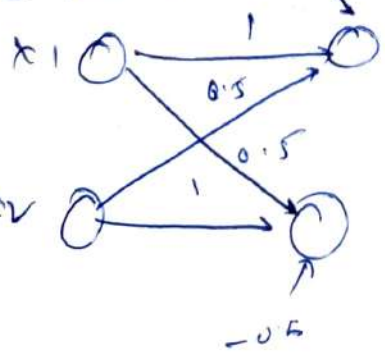
batch_1 = x_train[128:256]

Lecture 11 Mathematical foundation of deep learning

Key operation in Neural Networks

tf.keras.layers.Dense(128, activation='relu')

output = relu(dot(W, input) + b)



[1, 0.5], [0.5, 1]

2×2

2 d tensor with shape (2,2)

Input tensor

$[x_1, x_2]$

$[[x_1^{(1)}, x_2^{(1)}], [x_1^{(2)}, x_2^{(2)}]]$

2-d tensor (2,2)

Bias vector $[2, 0.5]$

1-d tensor with shape

output = np.maximum(0., z)

Element wise addition

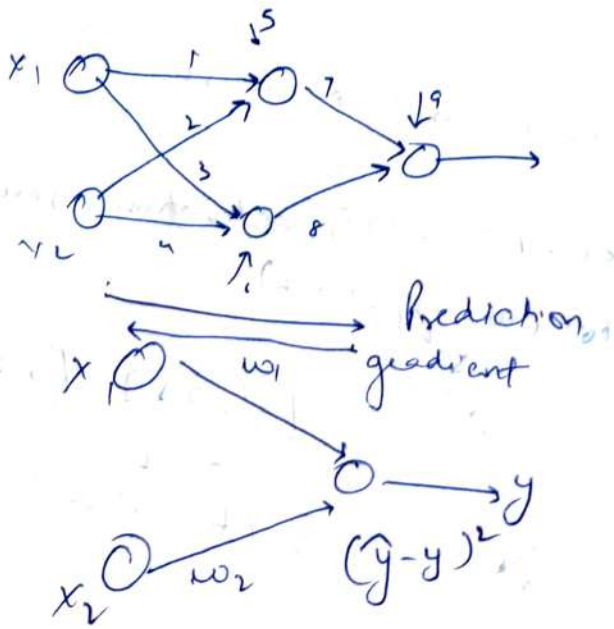
Broadcasting

x_1 = np.random.rand(32,10)

x_2 = np.random.rand(10,)

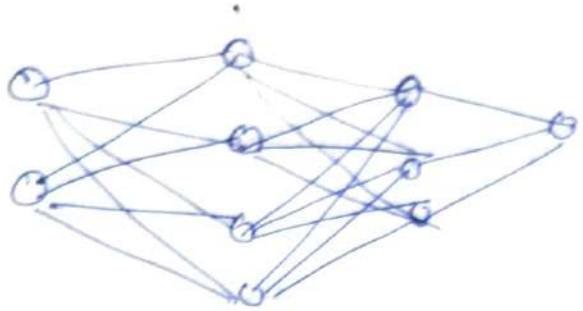
Reshaping

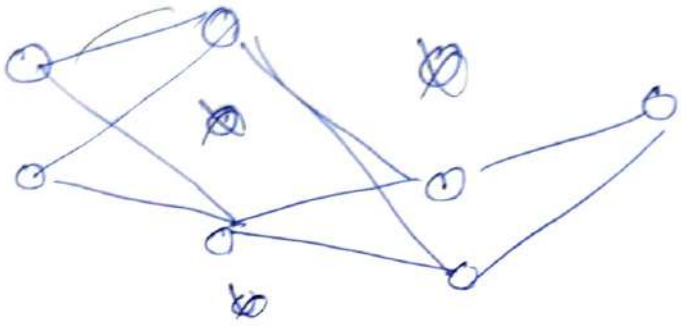
Transpose x = np.transpose



Dropout 0.2 or 20%.

0.1 or 10%





0.5 2.

Lecture 12 A Building data pipelines for tensorflow

Text

structure data

Image

Time series

sparse tensor

shuffel Dataset.shuffle()

Preprocessing

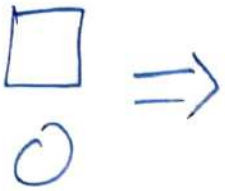
Tf. Keras tf. estimator

Lecture 12 B

Load csv with tf. data

Support vector Machine (SVM)

- Supervised learning used here



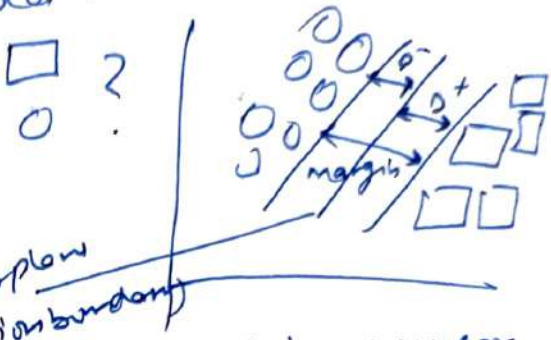
⇒ Model training

↓

Prediction ← new data

↓

output



- find the nearest point in one class which is very near to other class, then draw line which touch the one point and ⊥ to the hyperplane
- support vector :- the point from which two line ⊥ to hyperplane drawn, that point called support vector

why hyperplane drawn

- condition is that we need maximum width of the margin so in (b) which is angled and (a) is perpendicular / angled margin distance width is more that why this called maximal margin hyperplane should get selected

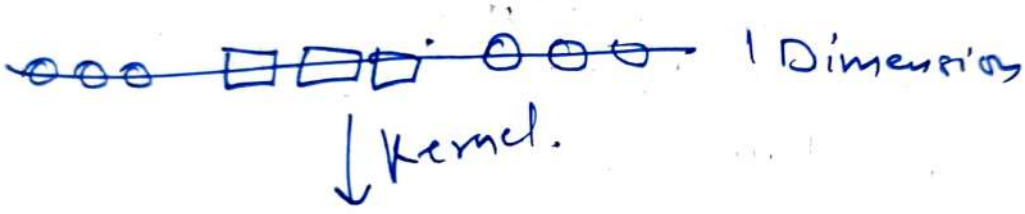
— Miss classification minimum
—— Maximal margin hyperplane

Non Linear SVM and Kernel function

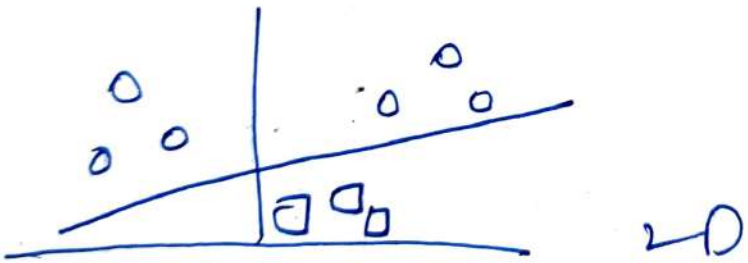
Low dimension feature space

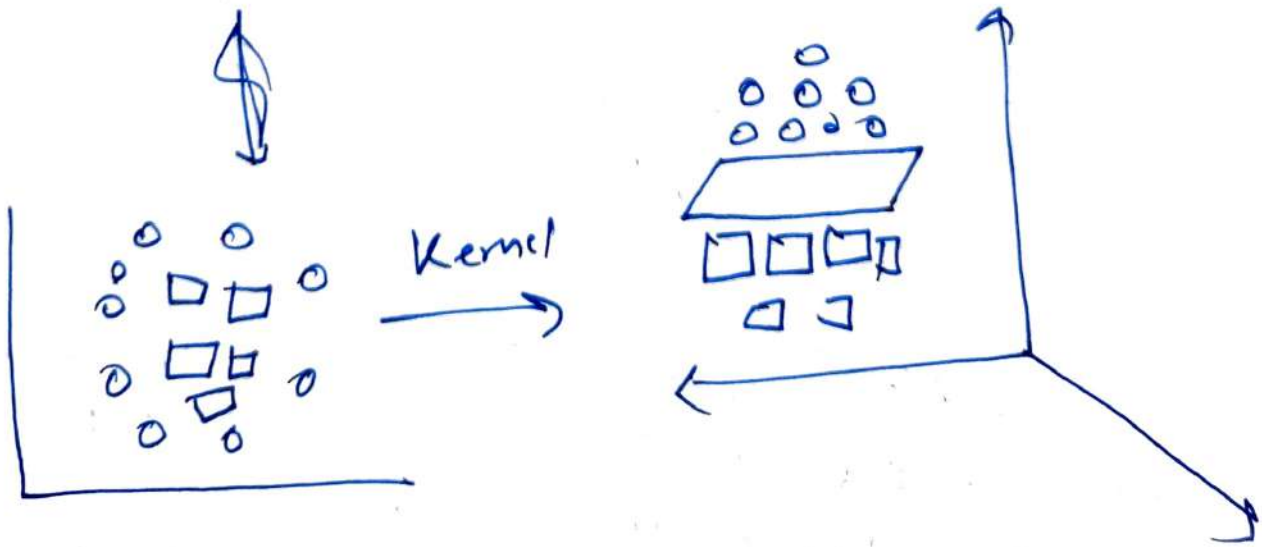
↓ Kernel

↓ high dimension feature space



↓ Kernel.





Decision tree Algorithm

- select one attribute which is known as target attribute
- then detect the information gain of that target attribute

$$I.G = \frac{-P}{P+N} \log_2\left(\frac{P}{P+N}\right) - \frac{N}{P+N} \log_2\left(\frac{N}{P+N}\right)$$

$$E(A) = \sum_{i=1}^{v} \frac{P_i + N_i}{P+N} I(P_i, N_i)$$

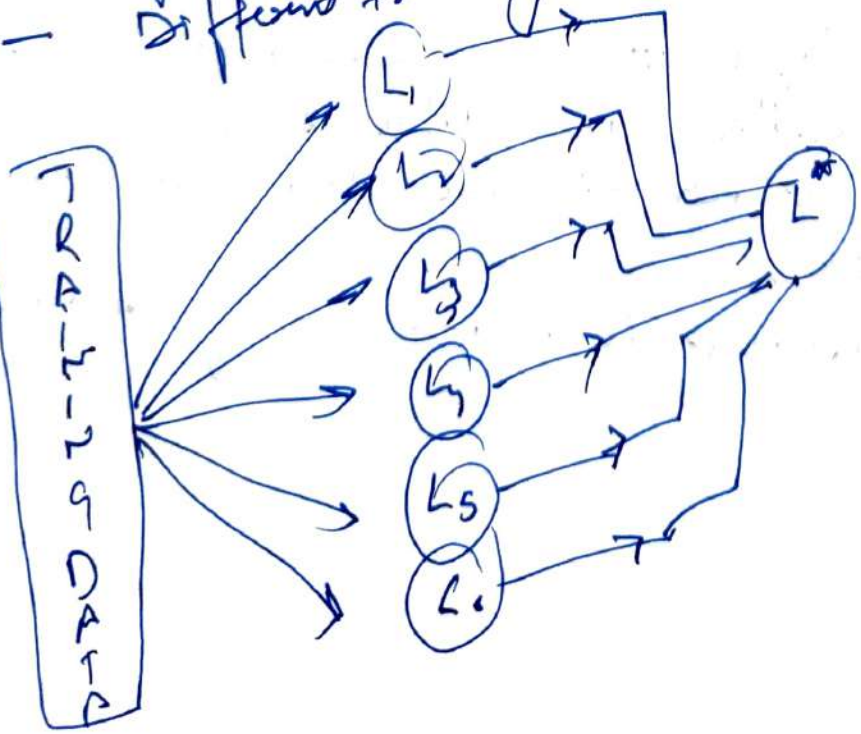
$$Gain = I.G - E(A)$$

$$\log_2 x = \frac{\log_{10} x}{\log_{10} 2}$$

Root node which has maximum gain and each draw with decrease the value of Gain information

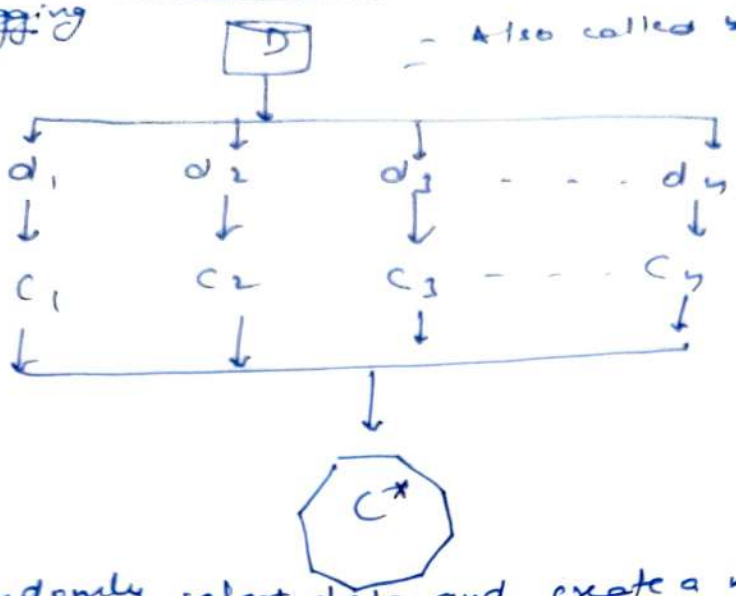
Ensemble learning

- Different Algorithm
- Different training data set



Ensemble Methods

Bagging



- Also called bootstrapping aggregation

- Randomly select data and create a new data set several time i.e $d_1, d_2, d_3 \ldots d_4$ with replacement
  i.e sampling with replacement
  $d_1, d_2, d_3 \ldots d_4$ called bootstrap sample
- these classifier are trained. $C_1, C_2, C_3 \ldots C_4$

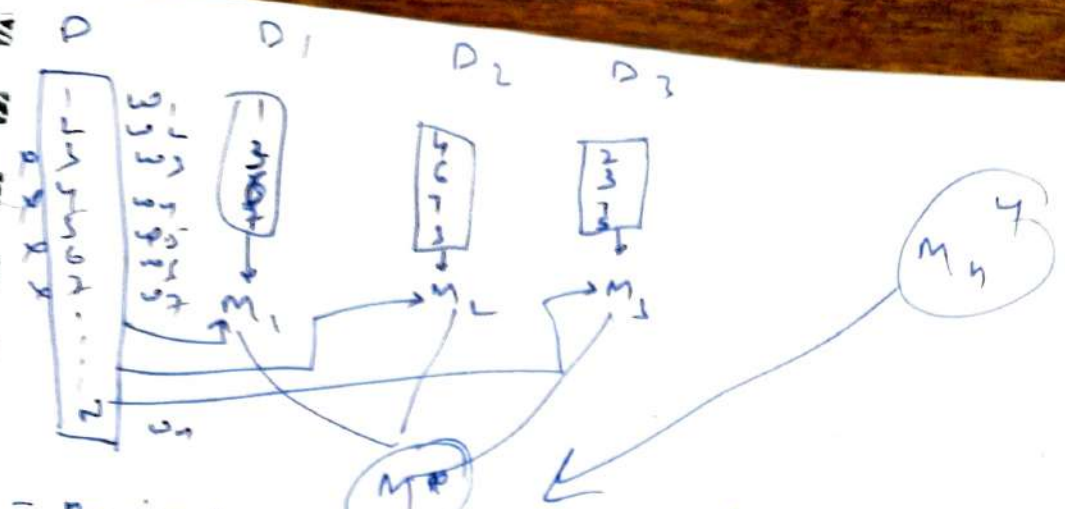
→ combine these classifier and generate ensemble classifier

→ error rate is low compare to individual classifier in ensemble classifier and accuracy is more in ensemble classifier

→ If we give a tuple to all classifier than individual classifier give different results than to decide which is the true answer we using voting.

→ Voting technique is used in case of classification problem

Boosting



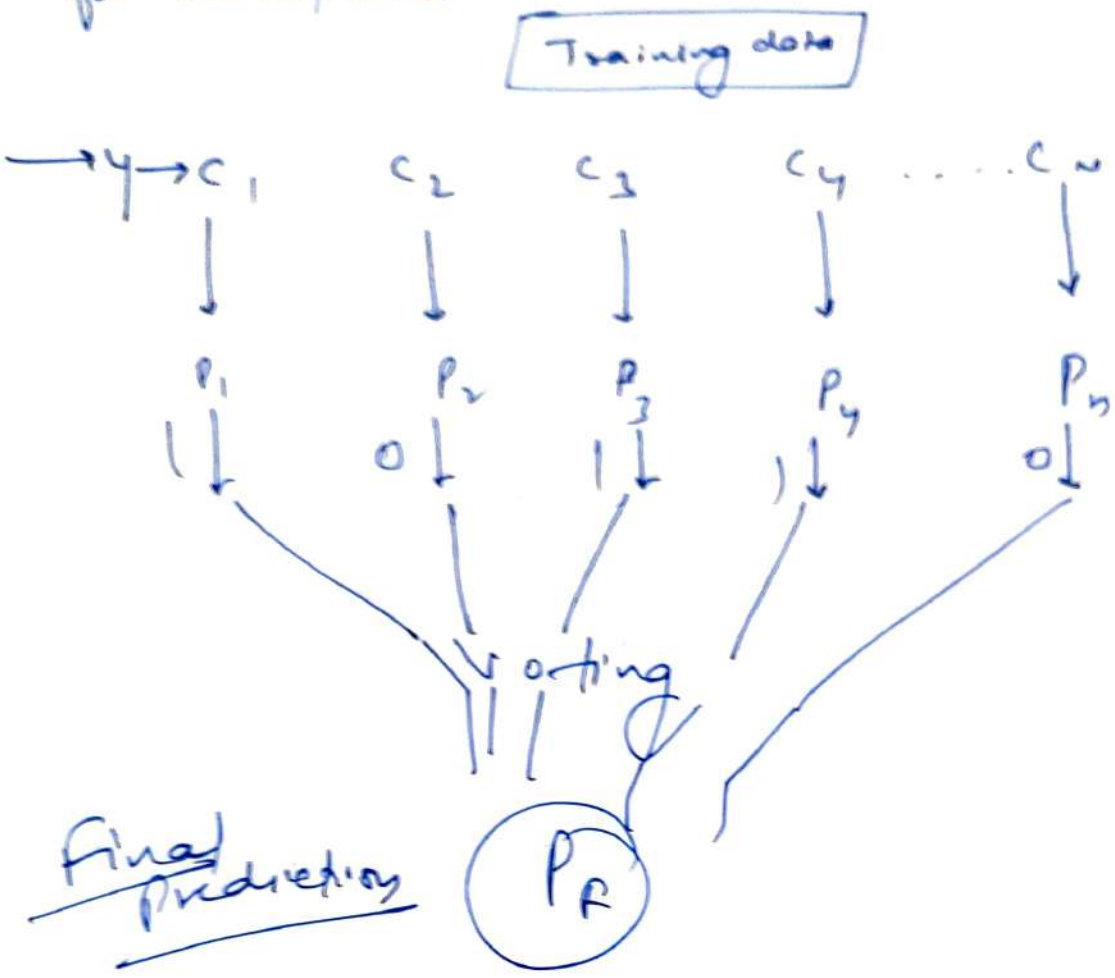
- n- instances and individual instances associated o
~~diff~~ weights and all weights are equal,
- Their will be some misclassification
- Before to random sampling update the weights particular those which are wrongly classified in previous model
- weights of wrongly classified now has higher degree than chance of selecting wrongly classified instance in the new model is more
- If all models are created then combine all models and create a strong model by combine all weak model
- Use voting method for test data

Voting classifier classification

(1) Hard voting also called majority voting

our Traing data we train n classifier then every classifier feed test data (4) all classifier give their predictions. after prediction voting is used for find prediction

- majority of voting decide which is the correct answer for classification



Soft voting depending on probability range from 0 to 1

C_1 0.9, 0.1

C_2 0.8, 0.2

C_3 0.4 0.6

(1,0) $= \frac{0.9 + 0.8 + 0.4}{3} = 0.7$

(1,1) $\frac{0.1 + 0.2 + 0.6}{3} = 0.3$ class

Random forest It is the ensemble classifier which use decision tree

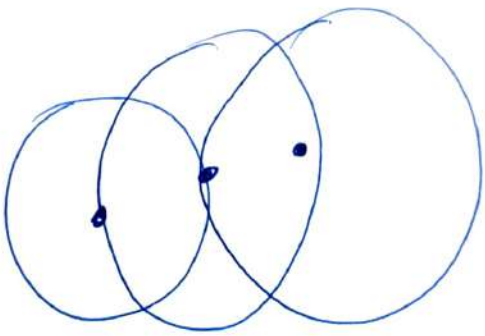
we have original data and from that create a Bootstrap data (BD)

- select subset of total variables
- Voting counting is performed, what with majority of votes attribute win

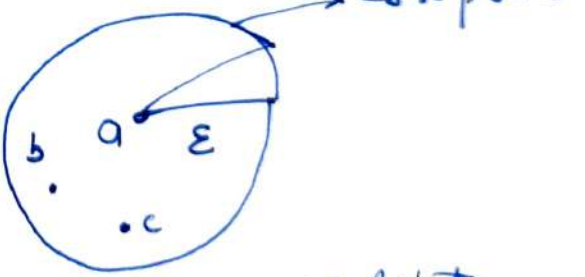
DBSCAN "Density Based spatial clustering of Application with noise"

A is better than B and C

DBSCAN



Noise point :



- Minimum points = 3 in a circle
- Boundary Point :- neighbour of core point
  Directly density reachable

K mean Algorithm

Euclidean distance

$$\sqrt{(x_0 - x_c)^2 + (y_0 - y_c)^2}$$

- Agglomerative Down to up
- Divisive up to Down

```
fig, ax = plt.subplots(figsize=(8,4))
sns.distplot(dataset.salary)
dataset['salary'].fillna(dataset['salary'].median(),
                         inplace=True)
salary_dataset = salary_dataset.dropna(axis=
                                       'rows')
```