SQL:-

USE <DB_NAME> → Use a database

SHOW TABLES; → A list of all tables in the database

DESCRIBE <table-name> → Shows a summary of the table's columns
Includes datatypes, length & names

```
mysql> DESCRIBE directors_genres;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| director_id | int(11)      | NO   | PRI | NULL    |       |
| genre       | varchar(100) | NO   | PRI | NULL    |       |
| prob        | float        | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

→ Two primary keys. Genre is not unique but combination of Genre & dir_id gives a unique value (probability). This type of primary key is called composite primary key.

→ Apps/Websites all of them use a database

```
Terminal - yash@yash-xub:~
| 412312 | "nica noche, La"          | 1996 |      NULL |
| 412313 | "nica Verdade, A"         | 1985 |      NULL |
| 412314 | "nica Verdade, A"         | 1958 |      NULL |
| 412315 | "pa el nimo"              | 1962 |      NULL |
| 412316 | "zem blch krlu"           | 2002 |      NULL |
| 412317 | "rgammR"                  | 1991 |      NULL |
| 412318 | "zgnm Leyla"              | 1995 |      NULL |
| 412319 | " Istanbul"               | 2002 |      NULL |
| 412320 | "sterreich"               | 1983 |      NULL |
| 412321 | "Ostreich"                | 1958 |      NULL |
+--------+---------------------------+------+-----------+
388269 rows in set (0.90 sec)

mysql>
```

SELECT * from movies;

Keyword

columns

* = all columns

table name

→ IRL tables have 100s of columns. So * may not be needed all the time.
In that case.

SELECT name, year FROM movies;

will display only these two columns

```
mysql> describe movies;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| id       | int          | NO   | PRI | 0       |       |
| name     | varchar(100) | YES  | MUL | NULL    |       |
| year     | int          | YES  |     | NULL    |       |
| rankscore | float       | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

→ We don't tell SQL how to print it. SQL is a declarative language. Not a procedural language.

→ The tabular output generated after running a command is called a 'result-set'.

→ SELECT * is always going to be slower than SELECT __ __ specific columns.

↳ this order can be anything. Doesn't necessarily have to be the same order as the original table.
But the order in which the rows appear in the result-set will be the same.
This is called row order preservation.
** There is no proper guarantee that the order will be the same unless ORDER BY is used **

→ Using backticks (`) allows us to add special characters in column names.

LIMIT :- when we don't want all rows to be displayed at once.

OFFSET :-

ex :- SELECT name, year FROM movies LIMIT 20; → displays only 20 values.

→ SELECT name, year FROM movies LIMIT 20 OFFSET 20; → ignore/offset first 20 value & print the next 20 values.

→ SELECT name, year FROM movies LIMIT 20 OFFSET 40; → ignore/offset the first 40 values & print the next 20 values.

Similar to page 1, page 2 etc, in google search results

ORDER BY :-

column names table name First 10 results.

```
mysql> SELECT name,rankscore,year FROM movies ORDER BY year DESC LIMIT 10;
+-----------------------------------------+-----------+------+
| name                                    | rankscore | year |
+-----------------------------------------+-----------+------+
| Harry Potter and the Half-Blood Prince  |      NULL | 2008 |
| Tripoli                                 |      NULL | 2007 |
| War of the Red Cliff, The               |      NULL | 2007 |
| Rapunzel Unbraided                      |      NULL | 2007 |
| Spider-Man 3                            |      NULL | 2007 |
| Untitled Star Trek Prequel              |      NULL | 2007 |
| DragonBall Z                            |      NULL | 2007 |
| Harry Potter and the Order of the Phoenix |      NULL | 2007 |
| Andrew Henry's Meadow                   |      NULL | 2006 |
| American Rain                           |      NULL | 2006 |
+-----------------------------------------+-----------+------+
10 rows in set (0.15 sec)
```

order descending

Similar to 'sort by' in websites.

Default is Ascending order.

Sorting by multiple columns :- SELECT ___ FROM ___ ORDER BY column_1 ASC, column_2 DESC;

↳ First sorts by column_1 in Ascending order & then sorts by column_2 in descending order.

DISTINCT :- Get all unique values in the column

ex :- SELECT DISTINCT genre FROM movies;

↑ All genres will be pointed.

Distinct can also be used on multiple columns

SELECT DISTINCT first_name, last_name FROM directors;

↑ Will list all unique name combinations.

(Apply distinct on one column but display multiple columns ?)

WHERE :- Apply condition to the query.

ex :- SELECT name, rankscore FROM movies WHERE rankscore > 9;

↑ Only movies with rating > 9

SELECT * FROM MOVIES WHERE rankscore > 9 ORDER BY rankscore;

↑ Same query but ordered by rankscore in Ascending order.

→ The conditions output can be (i) TRUE (ii) FALSE (iii) NULL

→ != & <> both imply not equal to in SQL.

SELECT * FROM movies WHERE rating <> 1.0;

↑ all movies except those whose rating is equal to 1.

NULL → unknown / missing / does not exist

↳ keyword.

'=' does not work with NULL.

ex:- Select * from movies where rankscore = NULL;

↳ will return empty set.

IS NULL & IS NOT NULL should be used instead.

SELECT * FROM movies WHERE rankscore IS NOT NULL;

IS NULL;

→ SELECT * FROM movies WHERE rankscore LIKE 9.8;

LIKE tries to match that pattern whereas = 9.8 tries to find the exact same value. Sometimes LIKE performs better.

Logical Operators:-

AND, OR, NOT, ALL, ANY, BETWEEN, EXISTS, IN, LIKE, SOME

AND :- Both cond 1 & cond 2

NOT :- SELECT * from movies WHERE NOT year <= 2000;

↳ years > 2000 only.

OR :- only one cond has to be true.

BETWEEN :- SELECT * FROM movies WHERE year BETWEEN 1999 AND 2000;

↳ same as (year ≥ 1999 AND year ≤ 2000;)

Inclusive range ie., 1999 & 2000 are included.

BETWEEN a and b;

a should always be ≤ b otherwise it will result in an empty set.

IN :- SELECT * FROM movies where genre IN ('Comedy', 'Horror');

↳ Same as (genre = 'Comedy' OR genre = 'Horror';)

LIKE :- If we want names like "Bat...."

SELECT * FROM movies where name LIKE 'Bat%';

→ Indicates 0 or more characters (Regular Expression)

→ It's called a wildcard character.

_ :- Atmost one character.

'_atman'

↳ Batman ✓ Bratman ✗ (2)

Catman ✓

Ratman ✓

⋮

→ Backslash ie. the escape character.

ex:- if working with percentages.

'\%9%'

↳ Shows all %9_ marks.

Aggregate functions :- They compute a single value over a set of rows.

COUNT MIN MAX SUM AVG → All work similarly.

→ SELECT MIN(year) FROM movies;

↑ Prints the minimum year from the column

COUNT :-

```
mysql> SELECT COUNT(*) FROM movies;
+----------+
| COUNT(*) |
+----------+
|   388269 |
+----------+
1 row in set (0.05 sec)
```

→ overall number of rows.

→ Took all the columns.

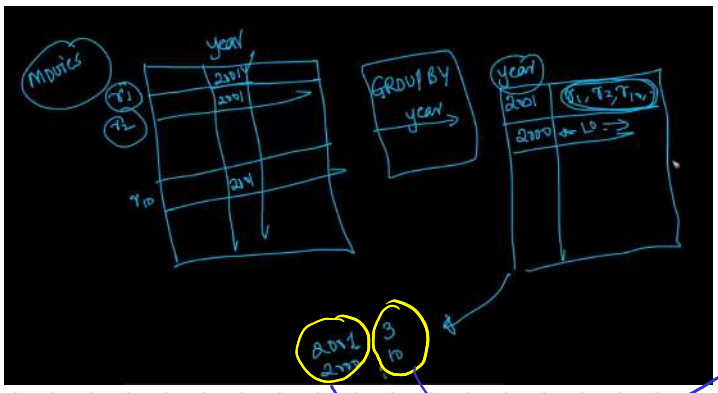
count(column_name) does not include NULL values.
COUNT(*) does.

SELECT COUNT(*) FROM movies WHERE year > 2000;

↳ Returns number of movies released after 2000.

→ COUNT(1) is better than COUNT(*) because it only iterates over one column instead of all.

GROUP-BY :-



SELECT year, COUNT(year) FROM movies GROUP BY year;

Using alias :- SELECT year, COUNT(year) year_count FROM movies GROUP BY year ORDER BY year_count;

Calculates this & stores it in

Sorted based on that.

- GROUP BY groups all the null values into one row.

Grouping odd year numbers :- SELECT year, COUNT(year) year_count FROM movies GROUP BY year WHERE year % 2 != 0 ORDER BY year count;

HAVING :-

SELECT year, COUNT(year) year_count FROM movies GROUP BY year HAVING year_count > 1000;

Order of execution :- ① ② ③

→ HAVING is often used with GROUP BY, but it's not mandatory

SELECT name, movies FROM movies HAVING year > 2000;

Same as

SELECT name, movies FROM movies WHERE year > 2000;

→ HAVING is applied on groups, WHERE is applied on rows. HAVING is applied after grouping, WHERE is applied before grouping.

→ HAVING will do tableSpace scan & WHERE will do table space scan. WHERE is better when grouping is not required.

Order of Keywords :-

```
SELECT
    ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      SQL_NO_CACHE [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
      [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
    [HAVING where_condition]
    [WINDOW window_name AS (window_spec)
        [, window_name AS (window_spec)] ...]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR {UPDATE | SHARE} [OF tbl_name [, tbl_name] ...] [NOWAIT | SKIP LOCK
      | LOCK IN SHARE MODE]]
```

MySQL

JOINS :-

→ Combine data from multiple tables

movies (m)

id	name
⋮	⋮

movie_genre (g)

movie_id	genre
⋮	⋮

SELECT m.name, g.genre FROM movies m JOIN movie_genres g ON m.id=g.movie_id LIMIT 20;

alias

'id' is the common row in both tables.

The above query is called an inner join.

Natural Join :- When the column names are same in both tables, ON can be skipped.

T1

C1	C2	C3

T2

C1	C4

SELECT * FROM T1 JOIN T2;

is same as

SELECT * FROM T1 JOIN T2 ON T1.C1 = T2.C1;

is same as

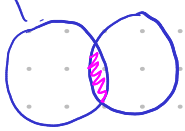
SELECT * FROM T1 JOIN T2 USING(C1);

Outer Join :- Three types. (i) Left Outer Join (ii) Right outer join (iii) Full Outer Join.

Left :-

SELECT * FROM T1 LEFT OUTER JOIN T2 ON T1.C1 = T2.C2;

Inner Join



↳ If values from C1 are missing in C2, they will be replaced with NULL.

Right :- SELECT * FROM T1 RIGHT OUTER JOIN ON T1.C1 = T2.C2;

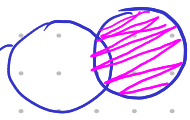
↳ At the end of result-set, leftover rows from T2 are added with NULL T1 values.

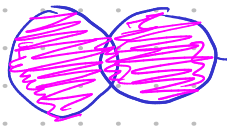
Left Outer Join

Full Outer Join :- SELECT * FROM T1 FULL JOIN T2 ON T1.C1 = T2.C2;

All rows from both tables are added with NULL.

Right Outer Join





→ Full Outer Join.

→ INNER JOIN = JOIN | FULL OUTER JOIN = FULL JOIN | LEFT OUTER JOIN = LEFT JOIN | RIGHT OUTER JOIN = RIGHT JOIN

K-Way Join :-

```
# 3-way joins and k-way joins
SELECT a.first_name, a.last_name FROM actors a JOIN roles r ON a.id=r.actor_id JOIN movies m on
m.id=r.movie_id AND m.name='Officer 444';
```

• → 1 • → 2

→ Joins can be computationally expensive (Time consuming)

SubQueries :-

```
| id          | int(11)      | NO   | PRI | 0       |       |
| first_name  | varchar(100) | YES  | MUL | NULL    |       |
| last_name   | varchar(100) | YES  | MUL | NULL    |       |
| gender      | char(1)      | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
4 rows in set (0.15 sec)

mysql> describe roles;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| actor_id | int(11)      | NO   | PRI | NULL    |       |
| movie_id | int(11)      | NO   | PRI | NULL    |       |
| role     | varchar(100) | NO   | PRI | NULL    |       |
+----------+--------------+------+-----+---------+-------+
3 rows in set (0.01 sec)

mysql> describe movies;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| id        | int(11)      | NO   | PRI | 0       |       |
| name      | varchar(100) | YES  | MUL | NULL    |       |
| year      | int(11)      | YES  |     | NULL    |       |
| rankscore | float        | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

actors

roles

movies

```
SELECT first_name, last_name from actors WHERE id IN
    ( SELECT actor_id from roles WHERE movie_id IN
        (SELECT id FROM movies where name='Schindler's List')
);
```

③ ② ①

→ Order just like loops in prog lange.

General Syntax :-

```
# Syntax:
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
    (SELECT column_name [, column_name ]
    FROM table1 [, table2 ]
    [WHERE])
```

Operators that can be used :-

IN, NOT IN,

EXISTS (returns true if query returns one or more records or NULL)

ANY, ALL

ANY: if any of the subqueries meet the condition.

ALL: iff all the subqueries meet the condition.

```
SELECT * FROM movies where rankscore >= ALL (SELECT MAX(rankscore) from movies);
# get all movies whose rankscore is same as the maximum rankscore.
```

9.9

example.

→ Nested Subqueries are much more readable than joins.

correlated Subquery :-

```
SELECT employee_number, name
  FROM employees emp
  WHERE salary > (
    SELECT AVG(salary)
      FROM employees
      WHERE department = emp.department);
```

(wikipedia)

→ inner query.

→ outer query.

→ For every employee, the inner query is run.

As a result, the inner query is run multiple times.

→ The query evaluation time of JOIN is very high compared to that of subquery. But after JOIN is performed, RDBMS performs indexing & it becomes faster.

→ Exists is intended to be used as a way to avoid counting.

```
--this statement needs to check the entire table
select count(*) from [table] where ...

--this statement is true as soon as one match is found
exists ( select * from [table] where ... )
```

Data Manipulation :-

SQL → Data Manipulation Language
→ Data Definition Language.
→ Data Control Language.

→ SELECT , INSERT , UPDATE , DELETE.

→ INSERT INTO table_name(col_1, col_2, col_3) VALUES (val_1, val_2, val_3), (val_1_1, val_2_1, val_3_1), (val_1_2, val_2_2, val_3_2);

↳ multiple rows being inserted.
→ if col_1 is a primary key, we can't insert a new row with an already existing primary key value. It'll throw an error.

We can use nested subqueries to insert data from another table.

```
INSERT INTO phone_book2
SELECT *
FROM   phone_book
WHERE  name IN ('John Doe', 'Peter Doe')
```

Sub-Q

→ UPDATE table_name SET col1 = val1, col2 = val2 WHERE <CONDITION>;
can be used with subqueries just like INSERT.
Multiple rows can be updated as well.

→ DELETE FROM table_name WHERE CONDITION;

→ TRUNCATE TABLE table_name;
↳ Delete all rows from table
Same as DELETE without WHERE clause.
Delete was row lock whereas Truncate locks the whole table.
Rollback is possible in both cases.

Data Definition Language :-

CREATE TABLE table_name (col_1 INT PRIMARY col_2 VARCHAR(50) NOT NULL);

INT

column names.

col_1 is the primary key.

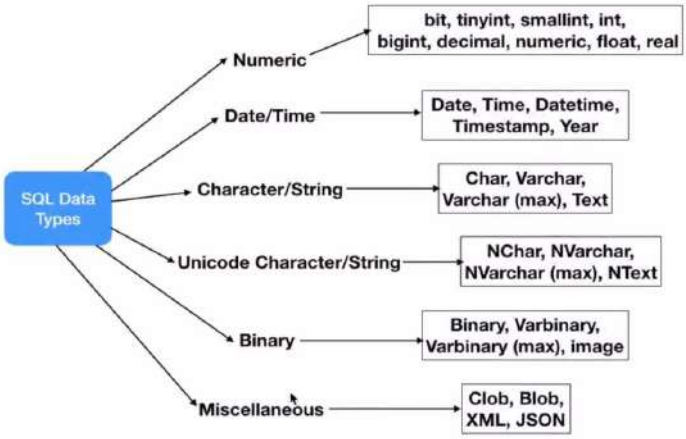
If is a char sequence of size ≤ 50.

column value can't be NULL

Constraints :-

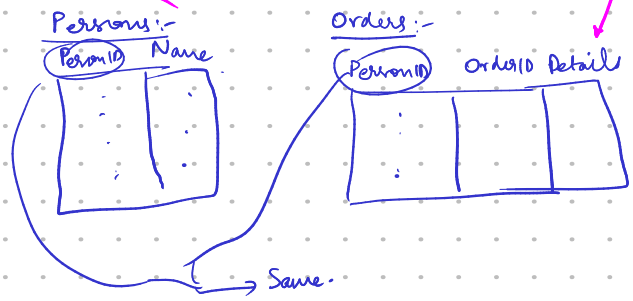
→ NOT NULL : column value can't be NULL
→ UNIQUE :- Column values have to be unique. Can contain NULL values.
→ PRIMARY :- combination of NOT NULL & UNIQUE.
→ FOREIGN :- Uniquely identifies a row in another table.
→ CHECK :- Ensures that all values follow a specific condition.
→ DEFAULT :- Sets a default value if a value isn't specified.
→ INDEX :- Used to create & retreive data from the database very quickly.
Indexes a column & allows us to access fast & perform ops fast.

SQL Datatypes :-



Foreign Key :-

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```



Modify Existing table :-

(ALTER) :- ADD , MODIFY , DROP

ALTER TABLE language ADD country VARCHAR(50);

ALTER TABLE language MODIFY country VARCHAR(60);

ALTER TABLE language DROP country;

Changing column name :- ALTER TABLE language CHANGE country origin VARCHAR(60);

old name ↑ ↑ new name.

Drop :-

DROP TABLE tablename;

↑ whole table is deleted. (including table structure)

DROP TABLE IF EXISTS tablename;

↑ drops table iff it exists.

TRUNCATE TABLE tablename; is same as DELETE FROM tablename;

↑ Deletes contents but not the table structure.

Data Control Language :-

→ for DB Admins.

Database

- → S/W Engineers — Need R & W
- → Data Analysts — Need only Read
- → Admins — R/W + Creating Table, Indexing a column etc.

* DCL control the access to a DB*

* Not all RDBMS have DCL. SQLite used in smartphones does not have DCL as it does not have multiple logins.

GRANT → Gives permissions

REVOKE → Revokes permissions.

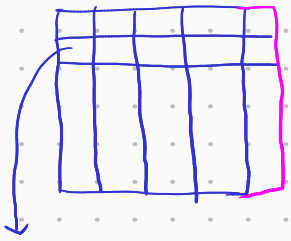
Plotting for Exploratory Data Analysis :-

→ Analyzing data using plotting tools, statistics, linear algebra etc.,

→ column to be predicted = class label / dependant variable

Data already there = data points / vectors → n dimensional numerical array.

↳ columns are called features / Input variable / independent variable



1 row = 1 array/vector.

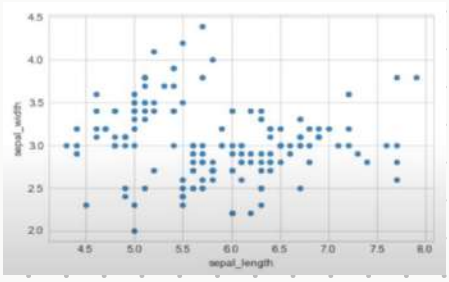
→ 1D vectors = Scalar.

[a], [1], [8.1]
⇓ ⇓ ⇓
'a' 1 8.1

→ Balanced Dataset → Each class has equal number of datapoints.

→ When reading a plot, always read axis labels & values. It doesn't always start at 0.

→ iris.plot(kind=scatter, x="sepal length", y="petal_width");



(sns.scatterplot() only plots one plot
sns.FacetGrid() can plot multiple plots.)

→ import seaborn as sns

sns.set_style("whitegrid") → white grid structure

sns.FacetGrid(iris, hue="species", size=4).map(plt.scatter, "sepal_length", "sepal_width").add_legend();

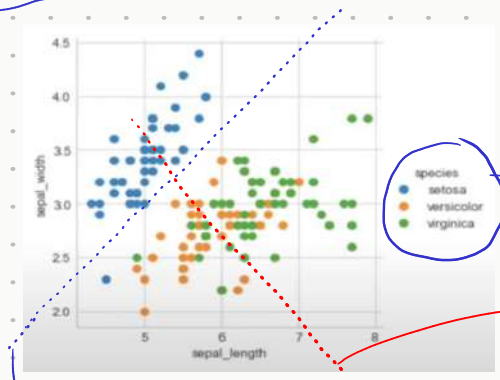
type of plot

x-Axis

y-Axis

dataset

by which column should the dataset be colored



→ legend.

Can't seprate as too many outliers.

↳ line could seprate setosa from versicolor & virginica (this is called linear separable)

Observations :- ① S_length & S_width can seprate setosa flowers from others.

② Seprating versicolor & virginica is harder.

QUICK SEABORN INTRODUCTION :-

→ simpler way to plot attractive plots.

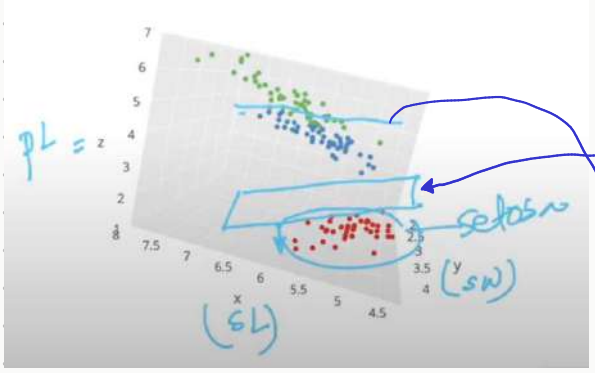
→ high level interface to matplotlib.

→ some features include:

- default aesthetic themes
- custom color palettes
- ★ Visualizing info from matrices & dataframes.
- ★ attractive statistical plots.
- ★ easily displaying distributions

→ Seaborn is a complement, not a substitute of matplotlib

3D Scatter Plot :-



when 3D plot is plotted, we use a plane to seperate instead of a line.

→ Plotting petal length helps notice that plotting a line can seperate them which we were not able to do in a 2D plot.

Drawbacks :- → while being interactive is a pro, it can't be viewed on 2D surfaces like paper or in an ipynb.

→ Since we can't visualize 4D datasets, we use pair plots.

↓

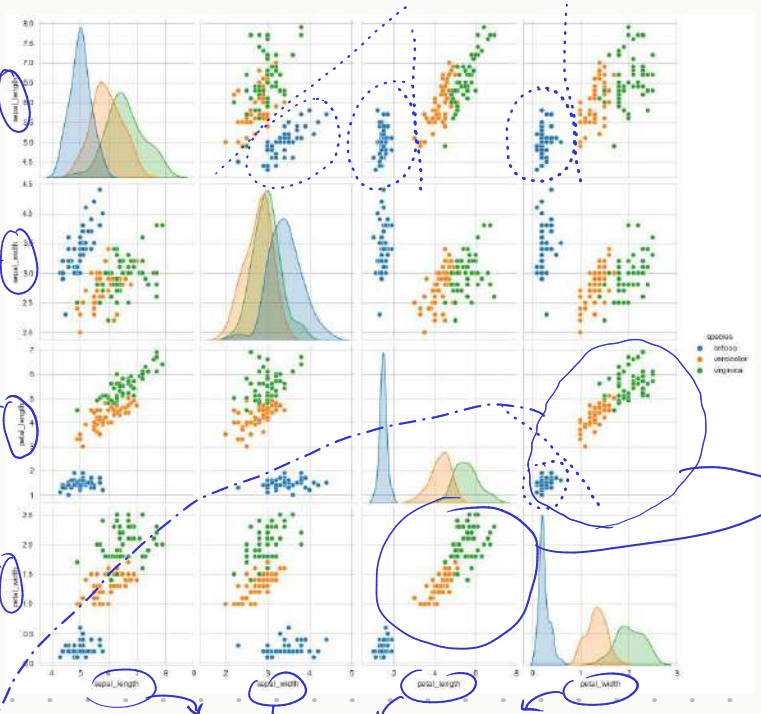
there are 4 variable ⇒ $^4C_2$ pairs are possible.

↓

6 → { (SL, SW), (SW, PL) / (SL, PL) (PW, PL) / (SL, PW) (SW, PW) }

Since we can't visualize 4D, we try & visualize these 6.

< sns.pairplot(iris, hue="species", size=3) >



(The diagonal plots are PDFs of each feature)

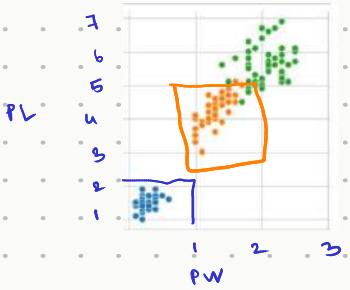
4 axis for the white row

x axis for the white column.

mirror. As both are b/w PL, PW.

Similarly there are 3 pairs in total.

So we focus only on 6 plots.



⇒ if PL ≤ 2 & PW ≤ 1 then flower type = setosa

if (PW ≤ 2 & PW ≥ 1) & (PL ≤ 5 & PL ≥ 2.5) then versicolor

→ There will be some error but that's OK.

Drawback of pairplots :-

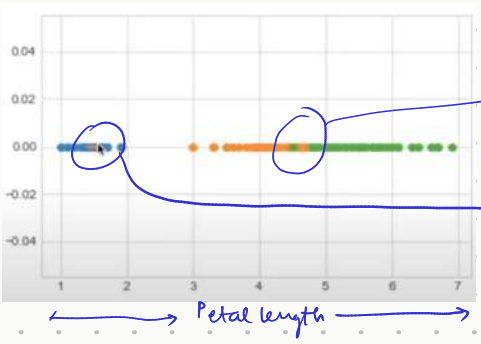
→ When dimensionality is 4, we get $^4C_2$ plots

If the dimensionality is n, we will get $^nC_2$ plots which is not that useful.

So we use techniques like PCA, t-SNE that will help with this.

In real world, we limit pairplots to only important features & not apply it on all features.

1D Scatter Plots :-



→ harder to read.

→ green overlaps orange.

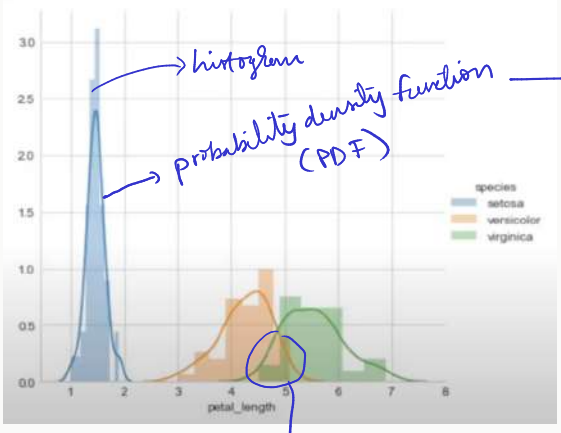
→ Number of points here is not clear.

To overcome this, instead of using a scatterplot, we can use a histogram

```
sns.FacetGrid(iris, hue="species", size=5).
    map(sns.distplot, "petal_length").add_legend()
```

distribution plot aka histogram aka density plot

y-axis: count.



→ histogram

→ probability density function (PDF) → smoothed histogram. Smoothed using kernel Density Estimation. (KDE)

overlapping region

if petal length < 2 then setosa.

But there will be errors

if petal length > 2 & petal length < 4.7 then versicolor

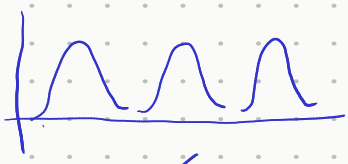
if petal length > 4.7 then virginica

≈

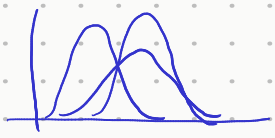
Univariate Analysis :-

→ Analysis of one variable (uni + variate)

→ The farther the distributions are, the better.

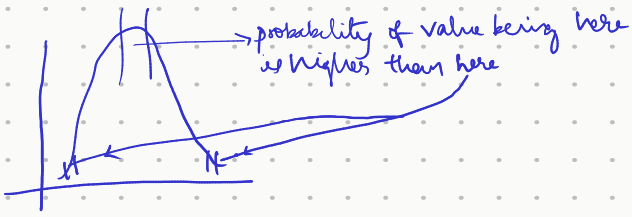


✓



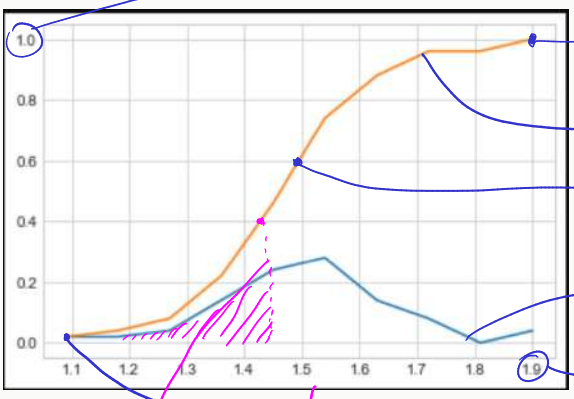
✗

PDF → probability of variable to be in the range of values. Ex:-



Cumulative Data Function :- (CDF)

→ What % of values are ≤ the value at that point.



→ 100% of the values are ≤ 1.9

→ CDF

→ 60% of the values are ≤ 1.5

→ PDF

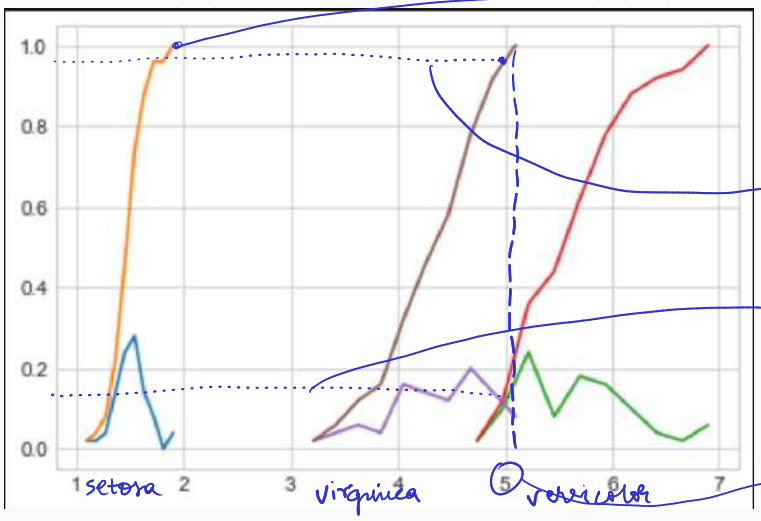
→ 0% of the values are ≤ 1.1

→ Differentiation of CDF = PDF

integration of PDF = CDF

The CDF value at this point = sum(values) till that point = cumulative sum.

→ Using CDF, we can make assumptions such as



→ 100% of setosa flowers petal length ≤ 2.

→ 95% of virginica have petal length ≤ 5

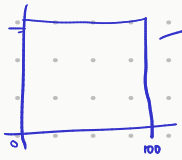
⇒ error % = 5%

→ 15% of versicolor have petal length ≤ 5

⇒ 85% are ≥ 5.

⇒ error % = 15%

bin = number of bars in a region in a histogram.



→ 1 bin

→ np.histogram returns counts & bin_edges.

How to compute CDF ? :-

```
counts, bin_edges = np.histogram(iris_setosa['petal_length'], bins=10,
                                 density = True)

pdf = counts/(sum(counts))
print(pdf);
print(bin_edges)

#compute CDF
cdf = np.cumsum(pdf)
plt.plot(bin_edges[1:],pdf)
plt.plot(bin_edges[1:], cdf)
```

→ number of bins

→ if false, it will just return the number of samples in each bin (aka normal hist). But if its true returns PDF

→ cumulative sum.

→ Another way to plot CDF :-

sns.Kdeplot(data, cumulative = True)

→ But can't adjust bins in this.

Mean, Variance & Standard deviation :-

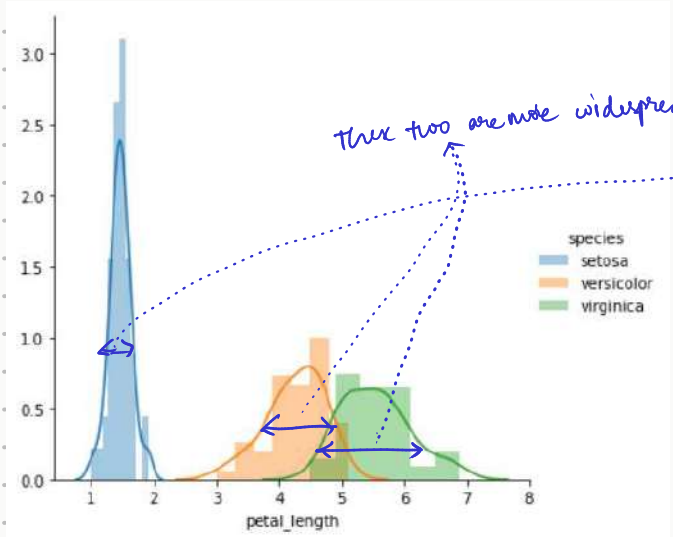
$$\text{Mean} = \frac{1}{n} \wedge \sum_{i=1}^{n} (x_i)$$

Gives an average value. But outliers affect this value. For ex; mean of setosa = 1.464.

Adding one extra value 50 will change the mean to 2.41.

How does mean help ?

Spread = how spreaded is the graph.



These two are more widespread than setosa.

we calculate standard deviation which gives us a range. Useful when we can't plot the data.

$$\text{std-dev} = \sqrt{\text{variance}} \longrightarrow \text{unit}$$

$$\text{variance} = \frac{1}{n} \sum_{i=1}^{n} (\mu_i - x_i)^2 \longrightarrow \text{unit}^2$$

μ_i → mean of dataset

Square because diff could be -ve.

np.mean() ⟶ mean

np.std() ⟶ standard deviations.

Now we can use this to estimate type of iris.

μ of setosa = 1.4

std-dev of setosa = 0.171

Now if flower is in the range $(1.4 - 0.171)$ to $(1.4 + 0.171)$ it could be a setosa.

Median :-

→ To overcome mean outlier problem, we use median.

→ Median = sort all elements & pick middle element.

if number of elements = even, mean of middle two elements = median.

→ Median is not corrupted as long as number of corrupted elements ≤ 50% of total elements.

np.median() to get median.

Percentile & Quantile :-

Let x be an array containing some values (100 values)

$x = [\ \ldots\ldots\ldots\ ]_{100}$

Let x_s be the sorted version of x.

$x_s = [\ \ldots\ldots\ldots\ ]$ → indices

0, 1, 2 … 50, 51 … 99, 100

The mean of elements at positions 50, 51 = median of x.

↳ This is also known as the 50th percentile.

↳ 50% of the values are ≤ this value.

Similarly $x[10]$ = 10th percentile.

↳ 10% of the values are ≤ this value.

→ 25th, 50th, 75th, 100th percentiles are called Quantiles.

How to get percentiles?

np.percentile(dataset, 90) ⟶ will give 90th percentile

np.percentile(dataset, np.arange(0,100,25)) ⟶ will give 0th percentile, 25th percentile, 50th percentile & 75th percentile.

↳ takes care of sorting. We don't have to pass a sorted array.

→ 90th & 95th percentile in e-commerce helps make sense out of data.

ex:- In Amazon, 90th percentile of delivery dates tell us 90% of orders are delivered in ≤ that days & they can work on improving that by adding more trucks, personnel etc.,

→ Duplicate value shift percentiles.

```
a1=[1,2,3,4,5]
a2=[1,2,2,2,2,2,2,2,2,3,4,5]
print(np.percentile(a1,90)) -- 4.6
print(np.percentile(a2,90)) -- 3.9
```

Median Absolute Deviation :-

→ Same notion as standard deviation.

$$\text{median}\Big(\underbrace{\underbrace{|x_i - \text{median}|}_{\text{deviation}}}_{\text{absolute deviation}}\Big)_{i=1}^{n}$$

Aman Sharma

I have seen many students are confused about why we are using SD rather than MAD
Don't worry I have an example

e.g
1,2,3,4,5
if you calculate the standard deviation of this number it will be 1.4142

now calculate MAD it will come around 1

now add or append 50
1,2,3,4,5,50
the standard deviation will be 17.1884
and MAD will be 1.5

so this is the power of MAD>SD.Try to explain all this concept with an example.

How to calculate?

→ Not readily available in numpy.

```
from statsmodels import robust
robust.mad(dataset)
```

mean ⟶ median

standard deviation ⟶ Median Absolute Deviation

→ don't have problems with outliers.

IQR :- Inter Quartile Region = 75th percentile − 25th percentile.

↳ 50% of values lie in this range.

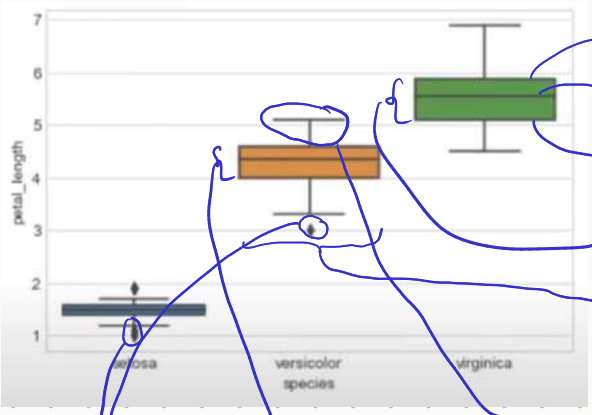
Boxplot & whiskers :-

→ histograms are not good at telling where the percentile lies.

→ We use boxplots for this.

→ ex:- for iris dataset :-

```
sns.boxplot(x="species", y="petal_length", data=iris)
```



→ 75th percentile

→ 50th percentile

→ 25th percentile.

→ how wide is it.

→ this width has no significance

→ Whiskers. There's no fixed way of coming up with them.

Seaborn calculates them as 1.5 * IQR

Whiskers are drawn in such a way that most of the points lie b/w them.

outliers. But not always.

IQR.

→ Box plot also tells us the skewness of the data. If box is cut in half, it means data isn't that skewed. Otherwise it is skewed. Sometimes it may also mean spread of data is high, like in the above graph.

Detecting outliers :-

if the point lies within $[(Q_1 - (1.5 * IQR)), (Q_3 + (1.5 * IQR))]$

it is not an outlier. Otherwise it is.

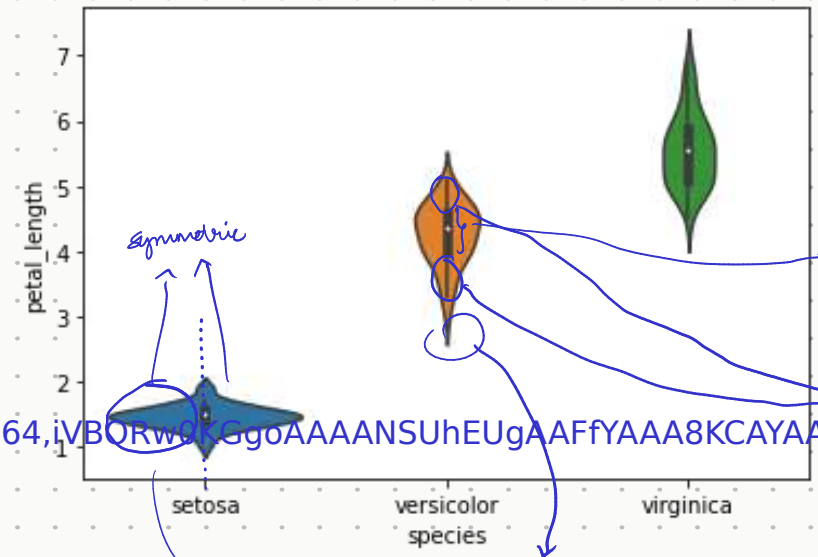
Whiskers are extended based on the following conditions:
The value/Extent of whiskers are decided based on minimum value of the following two conditions:
(1) + or - 1.5 * IQR
(2) Maximum / Minimum value present in the data set.
>The whiskers are extended-above up to
minimum value of (Max value present in data set AND Q3+1.5 IQR)
>The whiskers are extended-below up to
minimum value of (Min value present in data set AND Q1-1.5 IQR)
JUSTIFICATION:
** This rule is mentioned clearly in the Data Mining: Concepts and Techniques
Book by Jiawei Han Page No. 50
Book PDF Source: https://www.pdfdrive.com/data-mining-concepts-and-techniques-3rd-2011-e34881113.html

Violin Plot :-

→ Best of both histogram & box plots.

How to plot :-

```
sns.violinplot(x="species", y="petal_length", data=iris, height=8)
```



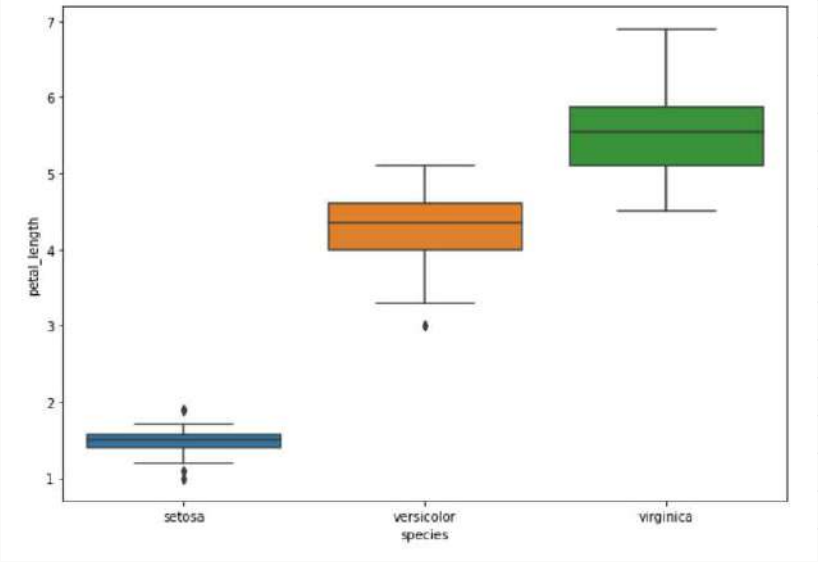
Violin plots have longer tail if there are outliers.

How to have plots of various sizes :-

size of plot

subplot 111

```
plt.figure(figsize=(10,7))
ax=plt.subplot(111)
sns.boxplot(x='species',y='petal_length', data=iris,ax=ax)
plt.show()
```



Multivariate probability Density & contour plot :-

→ Univariate → histogram, CDF, PDF

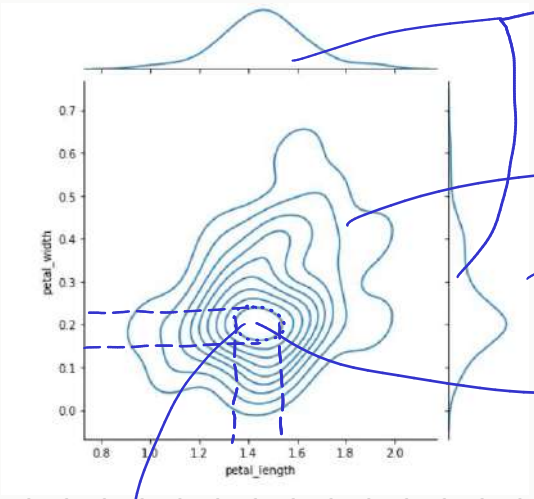
Bivariate → Scatter plot, pair plot.

→ For multivariate we use scatterplot.

How to plot?

```
sns.jointplot(x="petal length", y="petal width", data=iris_setosa, kind="kde")
plt.show()
```

→ The third dimension for this contour plot = density.

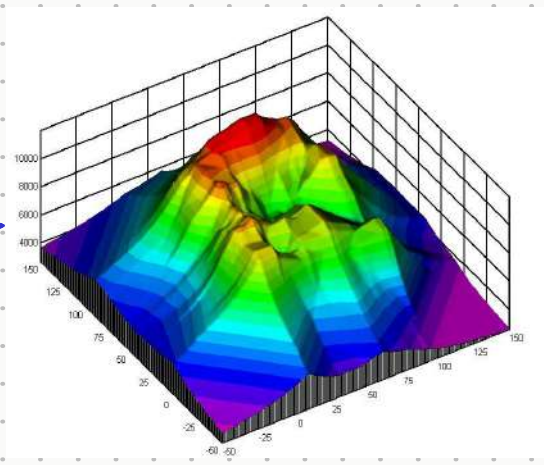


Normal PDFs

contour plot

3D perspective.
Like a hill projection.

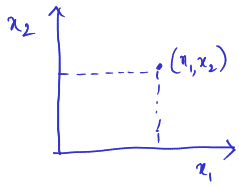
top most point.

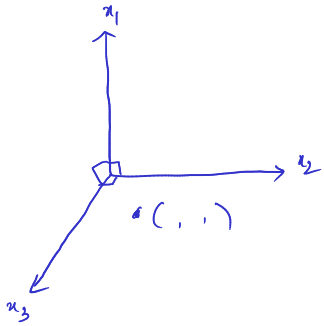


This area has the most density (most points)

Linear Algebra :-

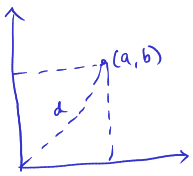
Point / vector :-





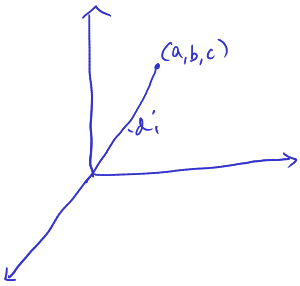
$\Rightarrow$ 4D, 5D, ... can't be visualized.

Distance of point from origin :-



$d = \sqrt{a^2+b^2}$

(From pythagorean theorem)



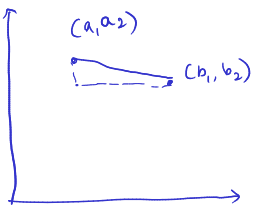
$d_1 = \sqrt{a^2+b^2+c^2}$

$\Downarrow$

nDimensional $\Rightarrow$ $d = \sqrt{a_1^2+a_2^2+\cdots+a_n^2}$ if $p = (a_1, a_2, a_3, \ldots a_n)$

whatever we are using in 1D, 2D we are able to generalize it to nD

Distance b/w two points :-



$d = \sqrt{(a_1-b_1)^2+(a_2-b_2)^2}$

$p = (a_1\ a_2\ a_3)$

$q = (b_1, b_2, b_3)$

$\Downarrow$

$d = \sqrt{(a_1-b_1)^2+(a_2-b_2)^2+(a_3-b_3)^2}$

$\Downarrow$

For n dimensional points,

$d = \text{Sqrt}\left(\sum_{i=1}^{n} (a_i-b_i)^2\right)$

row vector :-

$A = [\overset{1}{a_1}, \overset{2}{a_2}, \overset{3}{a_3}, \ldots, \overset{n}{a_n}]$ $1 \times n$ $\longrightarrow$ Number of columns.

1 $\rightarrow$ Number of rows

$\rightarrow A_{1\times n}$

Column vector :-

$$B = \begin{array}{c} 1 \\ 2 \\ \\ n \end{array}\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \longrightarrow B_{n\times 1}$$

$n \times 1$ $\rightarrow$ column.

n $\rightarrow$ rows

Vector operations:-

$a = [a_1 \; a_2 \; a_3 \; \dots \; a_n]$

$b = [b_1 \; b_2 \; b_3 \; \; b_n]$

(In general, unless specifically mentioned, assume it is a column vector)

$c = a + b = [a_1+b_1 \quad a_2+b_2 \quad a_3+b_3 \quad \dots \quad a_n+b_n]$

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix} \xrightarrow{T} a = [a_1 \; a_2 \; a_3 \; \dots \; a_n]$$

Transpose.

Multiplication:- Two types.

(i) dot product

(ii) cross product (Not often used in ML)

(i) Dot product:-

$$a \cdot b = a_1b_1 + a_2b_2 + a_3b_3 + \dots + a_nb_n$$

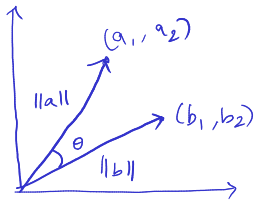
$$= \underset{1\times n}{[a_1, a_2, a_3, \dots, a_n]} \; \underset{n\times 1}{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}}$$

Match

$$\Rightarrow a \cdot b = a \cdot b^T$$

$$= \sum_{i=1}^{n} a_i b_i$$

Geometrical Intuition:-



$\|a\|$ = length of a from origin

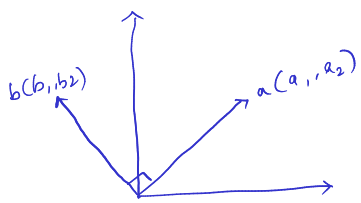
$= \sqrt{a_1^2 + a_2^2}$

$$a \cdot b = \|a\| \, \|b\| \cos\theta$$

$$\boxed{\Rightarrow a_1b_1 + a_2b_2 = \|a\| \, \|b\| \cos\theta}$$

$\Rightarrow$ Angle between two vectors, $\boxed{\theta = \cos^{-1}\left\{\dfrac{a_1b_1 + a_2b_2}{\|a\| \, \|b\|}\right\}}$

→



$$\Rightarrow a \cdot b = \|a\| \, \|b\| \cos\theta$$

$$= \|a\| \, \|b\| \cos 90^\circ$$

$$= \|a\| \, \|b\| \, (0) = 0$$

→ This means that A is being scaled down by a factor equal to length of projection B on A.

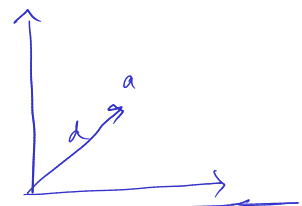
$\Rightarrow$ If dot product $= 0$, then angle b/w vectors $= 90^\circ$

↳ Applied to any dimensions.

→ $a \cdot a = a_1a_1 + a_2a_2 + \dots + a_n$

$$= a_1^2 + a_2^2 + \dots a_n^2$$

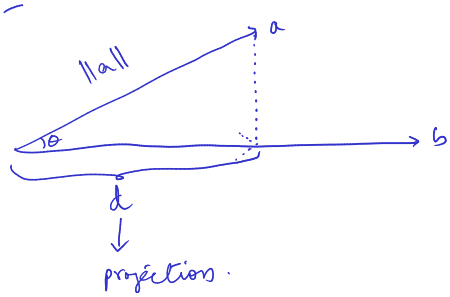
$$= \|a\|^2$$



$d = \sqrt{a_1^2 + a_2^2 + a_3^2 + \dots a_n^2}$

Dot product tells us how parallel two vectors are and cross product tells us how perpendicular they are. Since there is only one way of being parallel, dot product returns scalar.

Projection:-



$$d = \|a\| \cos\theta \longrightarrow (1)$$

$$a \cdot b = \sum_{i=1}^{n} a_i b_i = \|a\| \, \|b\| \cos\theta$$

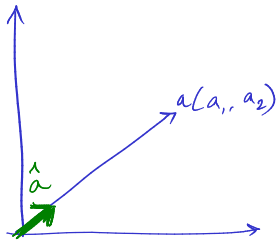
$$\Rightarrow \|a\| \cos\theta = \frac{a \cdot b}{\|b\|}$$

sub in (1)

$$d = \frac{a \cdot b}{\|b\|}$$

$$\boxed{\Rightarrow \text{Projection of 'a' on 'b'} = \frac{a \cdot b}{\|b\|}}$$

Unit vector:- (â) hat/cap



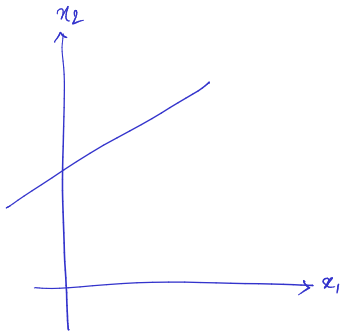
$$\hat{a} = \frac{a}{\|a\|}$$

→ $\hat{a}$ is in the same direction as 'a'

→ $\|\hat{a}\| = 1$

Line :-

2D :-



$y = mx + c$

(or)

$ax + by + c = 0$ → This is called the general form.

→ Both are the same.

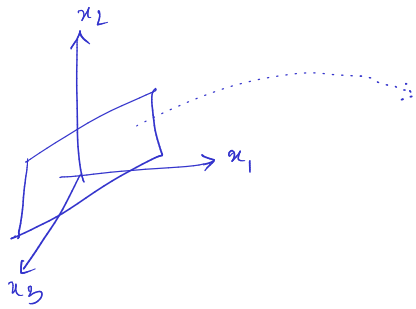
$$\Downarrow$$

$$ax_1 + bx_2 + c = 0$$

$$\Downarrow$$

$$w_1 x_1 + w_2 x_2 + w_0 = 0 \longrightarrow (2D)$$

line becomes plane



$$w_1 x_1 + w_2 x_2 + w_3 x_3 + w_0 = 0 \qquad \text{(Plane)}$$

$$\Downarrow$$

For n dimensions, it is called <u>Hyperplane</u>.

equation of hyperplane $= w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = 0$

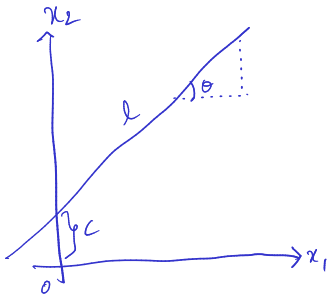
$$\Rightarrow w_0 + \sum_{i=1}^{n} w_i x_i = 0$$

$$\Rightarrow w_0 + \underbrace{[w_1 \; w_2 \; w_3 \; \cdots \; w_n]_{1\times n}}_{W} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{n\times 1}}_{X} = 0$$

$\Rightarrow w_0 + w^T x = 0$ → Generally, unless specified its a column vector.

→ Planes are typically denoted by π

$\pi : w_0 + w^T x = 0$ → eqn of plane in n dimensions.



$y = mx + c$ → slope, → y intercept.

$w_1 x_1 + w_2 x_2 + w_0 = 0$

$x_2 = \frac{-w_0}{w_2} - \frac{w_1}{w_2} x_1$

$m = \frac{-w_1}{w_2}, \quad c = \frac{-w_0}{w_2}$

If line is passing through origin, $c = 0 \Rightarrow \frac{-w_0}{w_2} = 0 \Rightarrow w_0 = 0$

$\Rightarrow$ eqn becomes $w_1 x_1 + w_2 x_2 = 0$

for 3D → $w_1 x_1 + w_2 x_2 + w_3 x_3 = 0$ → plane

⋮

nD → $w_1 x_1 + w_2 x_2 + w_3 x_3 + \dots + w_n x_n = 0$

→ $w^T x = 0$

equation of plane passing through origin.

plane passing through origin

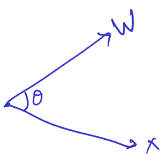
$\pi : w^T x = 0$

$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

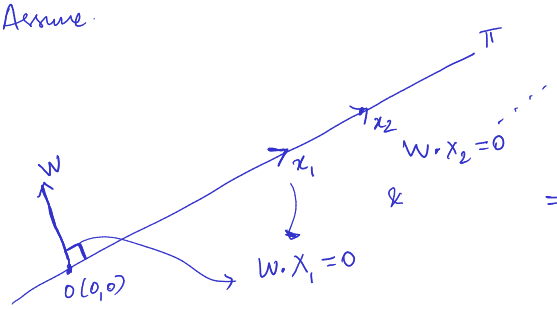
assume w & x as just two vectors

$w \cdot x = w^T x = \|w\| \, \|x\| \cos\theta = 0$

$w \perp x \Rightarrow \theta_{w,x} = 90^\circ$



Assume.



$\Rightarrow$ If $w \perp \pi$, then $w \cdot x_i = 0 \;\forall\; x_i \in \pi$

($w \cdot x_i$ is 0 for all points x_i on plane π)

→ w is just a vector that is perpendicular to the plane, at the origin.

→ w^T is called normal to the plane.

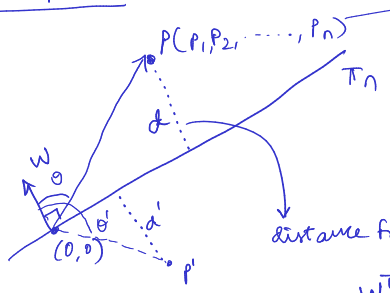
$\rightarrow \hat{W} = \frac{W}{\|W\|}$ $\qquad \hat{W} \cdot x_i = 0 \quad \forall\, x_i \in \pi$

→ For simplicity we always assume our plane passes through origin.

→ The equation of plane with intercepts a, b, c on x, y, z-axis respectively is

$$\frac{x}{a} + \frac{y}{b} + \frac{z}{c} = 1$$

Distance of point from plane :-



→ Because point could be n dimensional

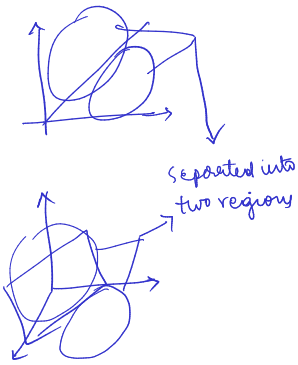
distance from p to π

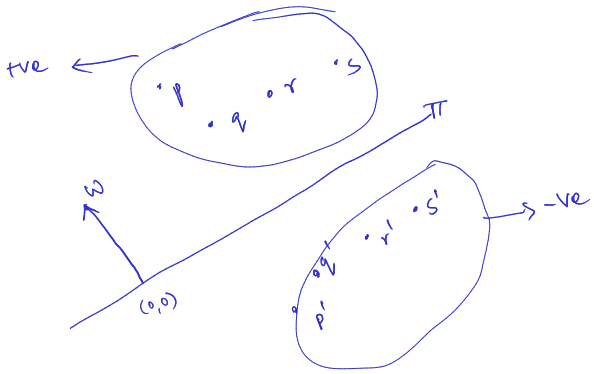
$$d = \frac{W^T p}{\|W\|} = +ve$$

$$d' = \frac{W^T p'}{\|W\|} = -ve$$

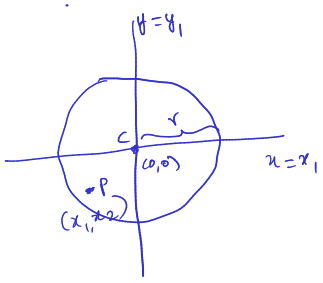
since $W \cdot P = \|W\| \, \|P\| \cos\theta$

This sign tells us on which side the point lies on the plane.





Circle, sphere & hypersphere:-



eqn of circle:-

$x^2 + y^2 = r^2$ (if its center is origin) else

$(x-h)^2 + (y-k)^2 = r^2$ where (h,k) = center

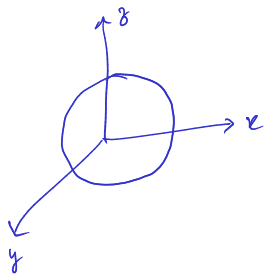
if $x_1^2 + x_2^2 < r^2$, then p lies inside the circle

$x_1^2 + x_2^2 > r^2$, then p lies outside the circle

$x_1^2 + x_2^2 = r^2$, then p is on the circle.

→ can be extended to all dimensions.

3D :-



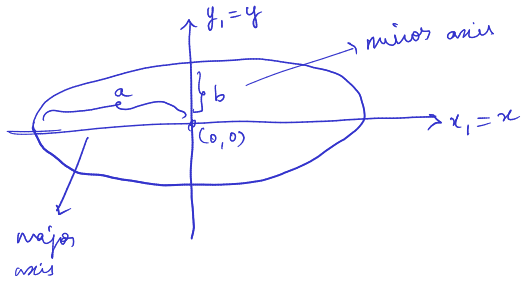
$x_1^2 + x_2^2 + x_3^2 = r^2$ → sphere

nD => $x_1^2 + x_2^2 + \cdots + x_n^2 = r^2$ ⟶ hypersphere

$$\sum_{i=1}^{n} x_i^2 = r^2$$

Ellipse

2D :-



Not that often used in ML

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

$\frac{x^2}{a^2} + \frac{y^2}{b^2} < 1$ ⟶ (x, y) lies inside ellipse

> 1 ⟶ " " outside ellipse

$= 1$ ⟶ on the ellipse

⟶ applied to all dimensions.

3D :- (ellipsoid) ⟶ American Football Ball.

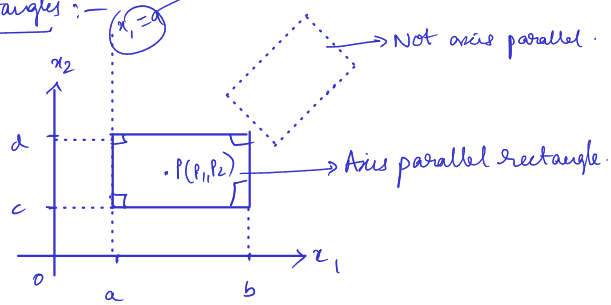
$$\frac{x_1^2}{a^2} + \frac{x_2^2}{b^2} + \frac{x_3^2}{c^2} = 1$$

nD :- (Hyperellipsoid) ⟶ $\frac{x_1^2}{a_1^2} + \frac{x_2^2}{a_2^2} + \cdots + \frac{x_n^2}{a_n^2} = 1$

Square, Rectangles :-

$x_1 = a$ ⟶ Axis parallel line.

2D :-



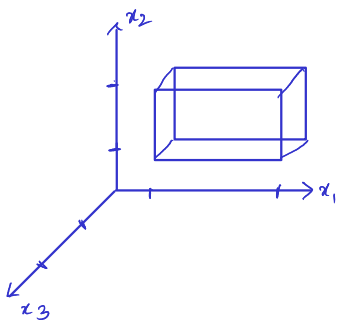
if $p_1 < b$ && $p_1 > a$

if $p_2 > c$ && $p_2 < d$

then p lies inside rectangle.

if its not axis parallel, then we take the equations of the sides & check which side of the line the point is

3D :- (cuboid)



Similar rule can be made for 3D

nD :- hypercuboid.

Probability & Statistics :-

Random variable :-

ex:- rolling dice → 6 sides = {1, 2, 3, 4, 5, 6} } Random Experiment

when rolled → any one of these equal outcome

random variable ← $X = \{1, 2, 3, 4, 5, 6\}$ → Sample Space.

tossing a coin → $Y = \{H, T\}$

$P(X=1) = 1/6$ $P(X=2) = 1/6$ ....

(probability of X being even)

$P(X \text{ is even}) = 3/6 = 1/2$

$(P(X=2) + P(X=4) + P(X=6) = 1/6 + 1/6 + 1/6)$

$P(X \text{ is odd}) = 1/2$

$P(x = x_1) \longrightarrow P(x_1)$ same thing diff notations.

Finite set of values → Discrete random value

→ Height of randomly picked student.

Y could be 162, 180, 120, 140, ... → infinite values → Continuous Random Variable

Outliers :-

Y : height of student

{122.2, 146.4, 132.5, ... 12.2, 156.3, 92.7 ...}

92.7 → could be an outlier

12.2 → outlier → could be human error (or) actual height

→ Outliers can corrupt data

→ A discrete value is obtained by counting

→ A continuous value is obtained by measuring.

Sample Space :- Set of all possible outcomes of an experiment.

→ A random variable value depends on the outcome of a random phenomenon.

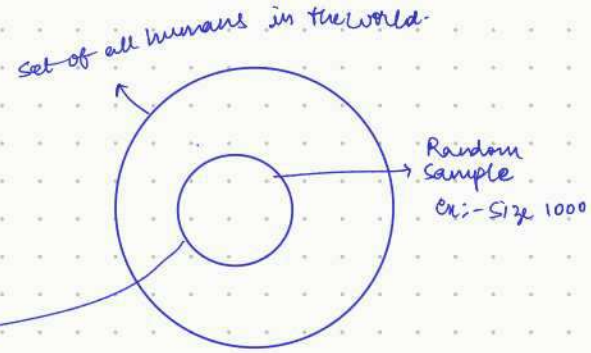
Population & Sample :-

→ Estimating the average height of human

↳ $\mu = \frac{1}{Pop} \sum_{i=1}^{Pop} h_i$ (IMPOSSIBLE)

So we estimate

often represented by $\bar{x}$ ← $\bar{x} = \frac{1}{1000} \sum_{i=1}^{1000} h_{is}$



→ As sample size increases $\bar{x} = \mu$

$\bar{x}$ → Sample mean

μ → original mean

Sampling is of two types:-

(i) Simple Sampling

(ii) Stratified Sampling → Unbiased sampling & more accurate results.

ex:- 1000 people ····· > original sample

- 400 have cars
- 300 have bikes
- 200 have cycles
- 100 have nothing

Sample Size → 250

Simple Random Sampling

250 could have cars
(or)
250 could have bikes
(or)
100 bikes + 150 cars.

Stratified Random Sampling

cars → 100
bikes → 75
cycles → 50
nothing → 25

} These are random but equal imp to all classes

Gaussian Distribution:- (AKA Normal Distribution)

→ If X is a continuous random variable & X has a PDF curve ( ), then we say X has a Gaussian distribution.

PDF = Probability Density Function

Why learn?

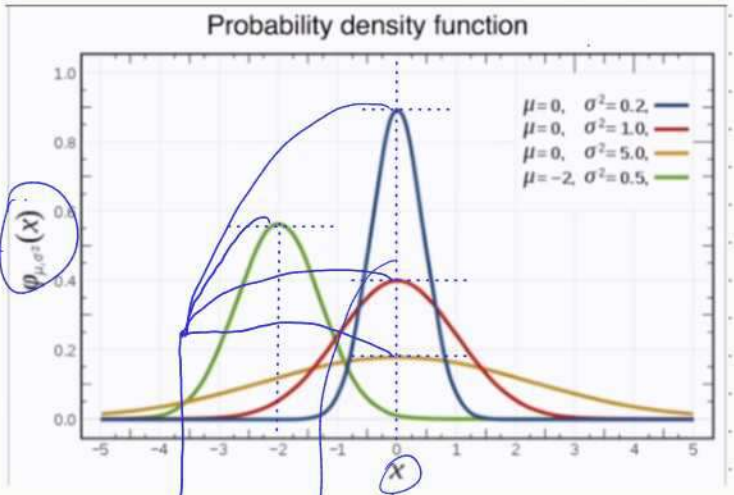
→ Stuff in nature tends to follow G.D.

→ heights, weights of people follow it. (Natural Phenomenon)

→ They are simple models that summarize R.V.

[Mean, Median & Mode are all same for G.D. There are equal number of measurements above & below the mean]

ex:-



PDF

μ = mean
σ^2 = variance } → parameters

[If this is not known, then we can't]

[If we are given μ, σ^2 & told that X follows GD, we can plot PDF. We don't need the whole data]

Variance is a measure of spread.

→ Continuous Random Variable

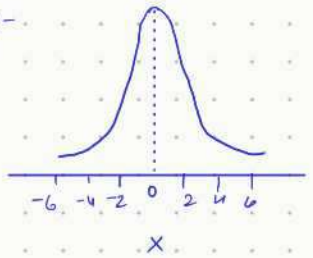
Red, Yellow, Blue have $\mu=0$ but varying variances.

The peak of curve is usually at 'μ'.

→ The parameters of Gaussian Distribution are μ & σ^2

$X \sim N(\mu, \sigma^2)$ ($\Longrightarrow$ X follows Gaussian Distribution with μ & σ^2)

ex:-



$\longrightarrow X \sim N(0,2)$

$$\rightarrow P(X=x) = p(x) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left\{\frac{-(x-\mu)^2}{2\sigma^2}\right\}$$

↑

Probability Density at a point (x) : PDF at any given point gives the probability density at that point. Probability of getting a single discrete value is 0.

ex:- If $\mu = 0$, $\sigma^2 = 1$, $\sigma = 1$

$$f(x) = \frac{1}{\sqrt{2\pi}} \exp\left\{\frac{-1}{2}x^2\right\} = y$$

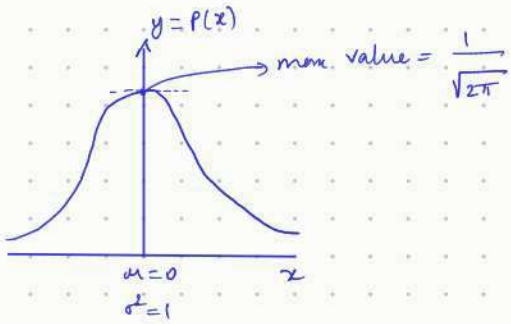
constants

further simplifying

$y = \exp(-x^2)$

→ when plotted

as x increases
y decreases.
as x decreases
y decreases



conclusions :-

1. x moves away from μ, y decreases
2. Graph is symmetric.
3. In this particular graph, y is reducing exponential squared. (e^{-x^2})

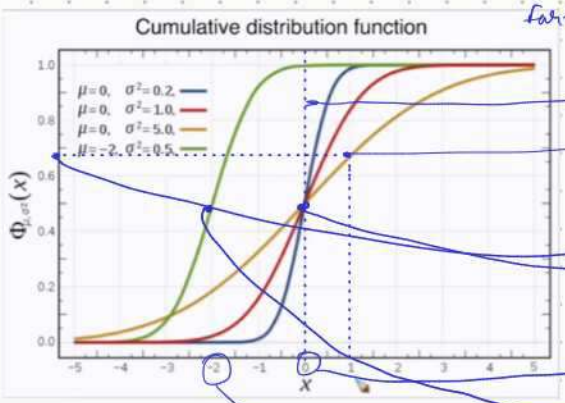
→ If p(x) is the probability density at a point 'x', the probability can be obtained by computing the integral of p(x) over a given interval.

ie, probability of getting $X \in [a, b]$ is $\int_a^b p(x)dx$

Cumulative Distribution Function (CDF) of Gaussian Distribution/Normal Distribution :-

As σ^2 increases, CDF goes farfromcentre line

CDF of a random variable looks like



→ $P(X \le 1) = 0.65$

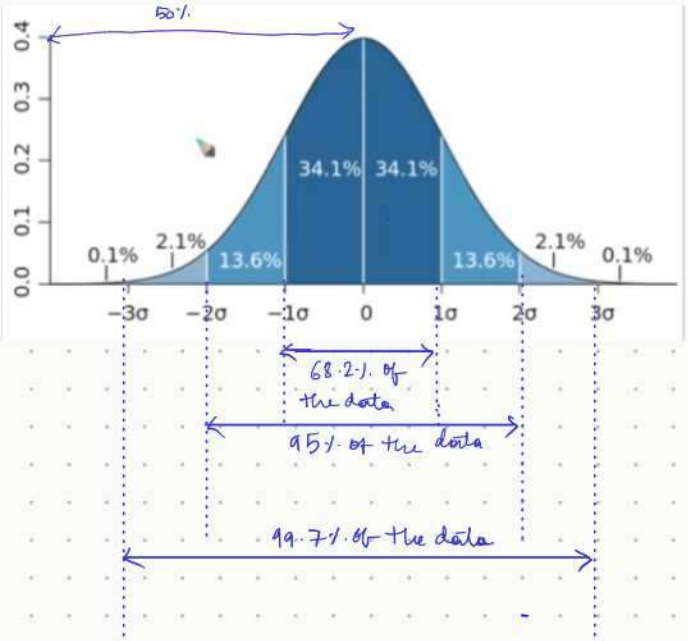
→ $\mu = 0$, centre of CDF is at 0.

→ $\mu = -2$, centre of CDF is at -2

$$CDF = \frac{1}{2}\left[1 + erf\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right] \longrightarrow \text{No need to memorize}$$

68 - 95 - 99.7 rule :-

if $\mu = 0$, $\sigma^2 = 4 \Rightarrow \sigma = 2$ $\qquad X \sim N(0, 4)$



How is this useful?

ex:- if human population height

$X \sim N(150, 25)$
$\mu \qquad \sigma$

$\Rightarrow$ 68.2% of human population lies b/w $(150-25, 150+25)$

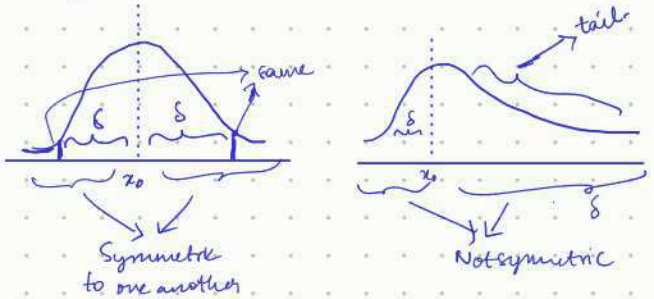
95% of people $(150-50, 150+50)$

99.7% of people $(150-75, 150+75)$

→ A standard gaussian distribution always has a mean of 0 & variance 1.
If it has other mean & variance, it's a non standard gaussian distribution.

Symmetric Distribution, Skewness & Kurtosis :-

→ They help understand shape of PDF



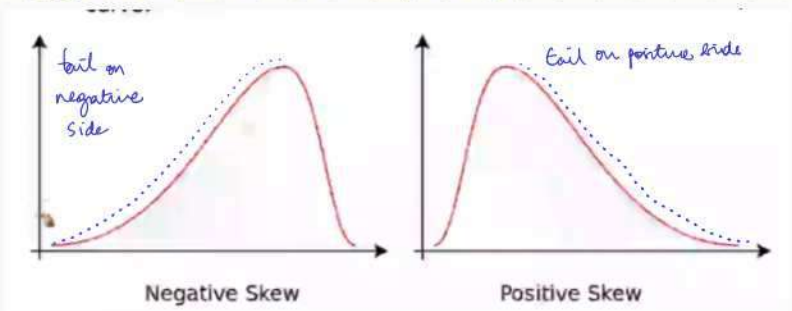
→ A probability distribution is said to be symmetric if and only if there exists a value x_0 such that

$$f(x_0 - \delta) = f(x_0 + \delta) \text{ for all real numbers } \delta$$

$f(x)$ is the height of PDF at any point 'x'

Skewness :-

Skewness is a measure of asymmetry -



$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\text{Skewness} = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^3}{\left[\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2\right]^{3/2}}$$

3 → if 2 = variance

→ sample std-deviation

Kurtosis :-

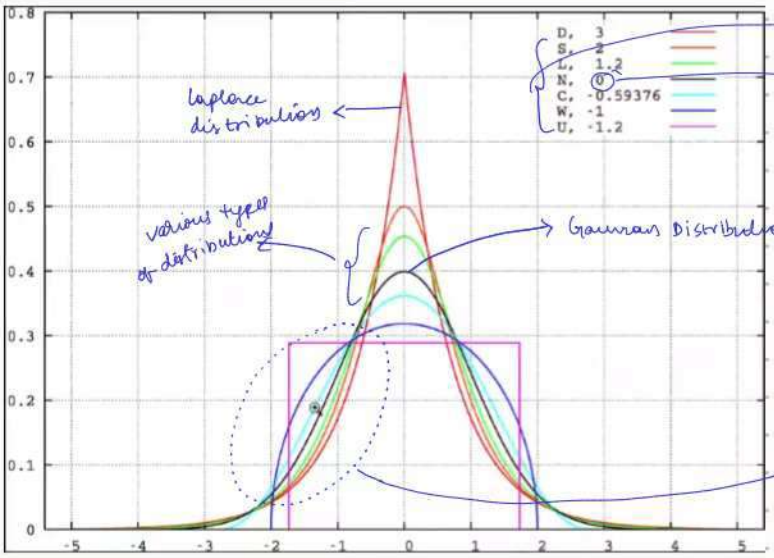
Kurtosis of Gaussian Distribution = 3

→ Measure of tailedness

excess kurtosis = kurtosis − 3

$$\text{excess kurtosis} = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^4}{\Big(\underbrace{\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2}_{\text{variance}}\Big)^2} - 3$$

→ Kurtosis is not a measure of peakedness. Might look like it -



→ excess kurtosis values

→ Gaussian Distribution, excess kurtosis = 0 ⇒ Kurtosis = 3

→ The rate at which the tails touch axis is different

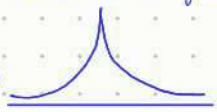
→ Kurtosis tells us if there are outliers are there in the data.

→ Kurtosis is used in analysis of trading, finance etc.

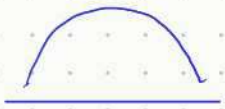
→ Kurtosis measures how much density/weight is in tails compared to middle part

→ If there are some stocks. Some stocks fall ↓
some go up ↑
some go up enormously ↑

To estimate risk after buying some stocks, we model data - Its generally assumed that they follow gaussian distribution for ease of calculation (Pre 2008 financial crisis). Due to gaussian distribution chances of losses are less & has good chance of high profits. But now, instead of assuming the data is a gaussian distribution, we look at the actual data & calculate the kurtosis

If kurtosis is > 3 ⇝  ⇒ chances of extreme profits are high & chances of losses are also high -

If kurtosis = 3 ⇒ Follows gaussian distribution

If kurtosis < 3 ⇒  ⇒ chances of extreme profits are low & chances of losses are also fairly low.

→ Kurtosis has a degree of 4. So if there is large deviations, then it will lead to higher kurtosis value.

→ High kurtosis is caused by infrequent extreme deviations rather than frequent modestly sized deviations.

→ In order to compare kurtosis b/w two curves, they need to have the same variance.

Standard Normal Variate :- (Z)

$Z \sim N(0,1)$

$\mu = 0$

$\sigma^2 = 1$

Let $X \sim N(\mu, \sigma^2)$

$X = [x_1, x_2, x_3, \ldots, x_{50}]$

Standardization :- $x_i' = \dfrac{x_i - \mu}{\sigma}$

$\Rightarrow x_i' \sim N(0,1)$

Standard Normal Variate.

Given any random variable X, where $X \sim N(\mu, \sigma^2)$

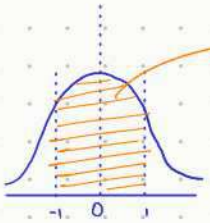
$$Z = \frac{X - \mu}{\sigma}$$

This is also called as basic Z-score formula. This basically tells how many σ is x away from μ.

$\Rightarrow Z \sim N(0,1)$

Why? :-

① After standardization, PDF becomes



68.2% of the data

(∵ 68-95-99.7 rule)

② If we are comparing multiple GDs with different μ & σ^2, doing this helps understand better & interpret better

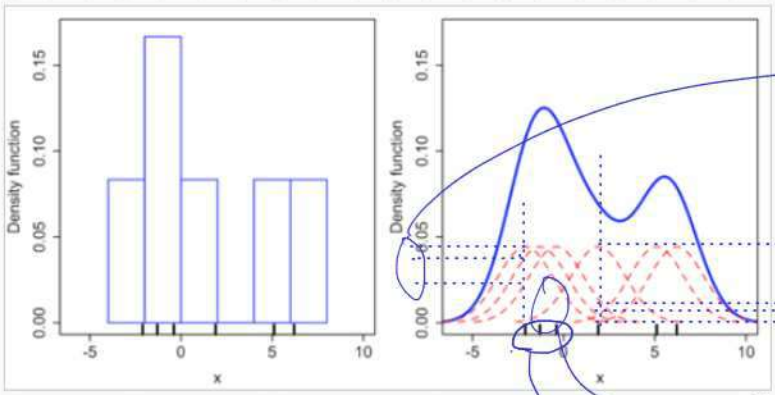
→ We use StandardScaler for standardization & MinMaxScaler for Normalization.

Two techniques of feature scaling.

Normalization → $x_i = \dfrac{x_i - x_{min}}{(x_{max} - x_{min})}$

Kernel Density Estimation :-

→ Used for smoothing histograms to obtain PDFs.



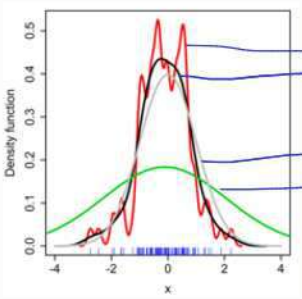
Comparison of the histogram (left) and kernel density estimate (right) constructed using the same data. The six individual kernels are the red dashed curves, the kernel density estimate the blue curves. The data points are the rug plot on the horizontal axis.

gives height at that point

→ Variance of the gaussian kernel is known as Bandwidth.

Gaussian kernel.

Since there are many points here, the number of means will be high so the PDF will be high. In normal histogram,



→ Narrow Bandwidth

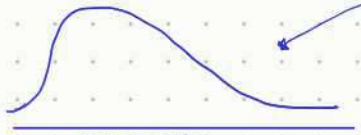
→ Medium Bandwidth

→ Actual Distribution.

→ Wide Bandwidth

Sampling Distributions & Central Limit Theorem :-

Let X be distribution of incomes over population [Not necessarily Gaussian]



population

Let S_1 be random sample of size n (let n=30) → mean = $\bar{x}_1$ → Sample means

" S_2 " " " → mean = $\bar{x}_2$

" "

" "

→ mean = $\bar{x}_n$

These will also have a distribution

The distribution $\bar{x}_i$ = Sampling distribution of sample means.

Central Limit Theorem :- If original distribution 'X' has finite mean (there can be infinite mean ex:- pareto) & variance & 'm' samples are created of size 'n' whose sample means are $\bar{x}_1, \bar{x}_2, \bar{x}_3, \ldots$ whose distribution is called sampling distribution of sampling mean, central limit theorem states that

$$\bar{x}_i \longrightarrow N\left(\mu, \frac{\sigma^2}{n}\right) \text{ as } n \to \infty$$

(But IRL if $n \geq 30$, then it becomes gaussian. Rule of thumb)

mean is same as original mean of distribution

Gaussian Distribution

Why? → By using just m×n datapoints, we are able to estimate μ & σ^2 of any distribution, if we just know that they are finite.

Quantile Quantile Plot (Q-Q-Plot) :-

Let X be a random variable

$X: x_1, x_2, x_3, \ldots, x_{500}$

Question :- Is X Gaussian Distributed. Q-Q-Plot helps us identify. Other techniques such as statistical testing (KS Testing) etc, also exist.

↓ graphical method

more powerful

How? :- ① Sort X & calculate percentiles.

$x'_1, x'_2, x'_3, \ldots, x'_{500}$

$x'_5 \longrightarrow 1^{st}$ percentile $\longrightarrow x^{(1)}$

$x'_{10} \longrightarrow 2^{nd}$ " $\longrightarrow x^{(2)}$

⋮

$x'_{500} \longrightarrow 100^{th}$ percentile $\longrightarrow x^{(100)}$

② take $Y \sim N(0,1)$

$Y: y_1, y_2, y_3, \ldots, y_{1000}$

↓

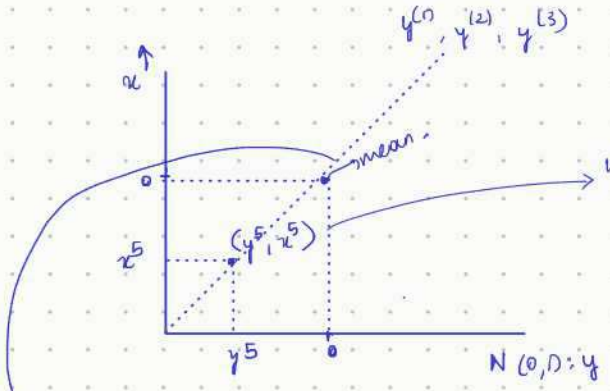
$y'_1, y'_2, y'_3, \ldots, y'_{1000}$

↓

$y^{(1)}, y^{(2)}, y^{(3)}, \ldots, y^{(100)}$

These are called as Theoretical Quantiles

③ Plot Q-Q-Plot using $x^{(1)}, x^{(2)}, x^{(3)}, \ldots$



we get 100 such pairs (y^1, x^1), (y^2, x^2), ⋮, (y^{100}, x^{100}) } All are plotted.

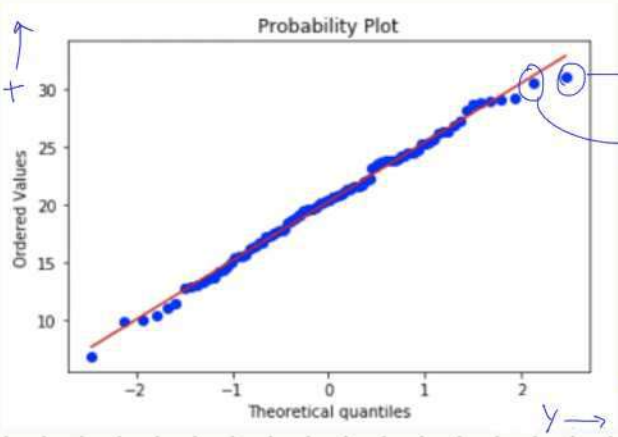
→ If (y^i, x^i) $\forall i \ni 100$, lie roughly on a straight line, they y & x have similar distributions.

⇒ X also has gaussian distributions.

$\boxed{\text{→ Stats.probplot( )}}$

→ If size of X is small, it is hard to interpret Q-Q plot. Won't be a straight line.

→ Another use of Q-Q plot is: given X, Y, does X & Y have the same distribution.



→ 100th percentile

→ 99th percentile

Generating Y :- (theoretical Quantiles)

```
#Q-Q plot
import numpy as np
import pylab
import scipy.stats as stats

# N(0,1)
std_normal = np.random.normal(loc = 0, scale = 1, size=1000)
```

mean std-dev

Generating X :-

```
# generate 100 samples from N(20,5)
measurements = np.random.normal(loc = 20, scale = 5, size=100)
#try size=1000

stats.probplot(measurements, dist="norm", plot=pylab)
pylab.show()
```

μ σ

How & where to use distributions :-

→ All probability concepts are used for EDA most of the time.

→ If data is Gaussian Distributed, many assumptions can be made by plotting PDF & CDF

Margin of error :- how much deviations from original numbers is allowed in samples.

Chebyshev's Inequality :-

→ If we know that data is Gaussian Distributed, we can apply 68-95-99.7 rule.

→ But if we don't know the distribution, but we know μ, σ can we make assumptions like this?

μ (finite), σ (nonzero & finite)

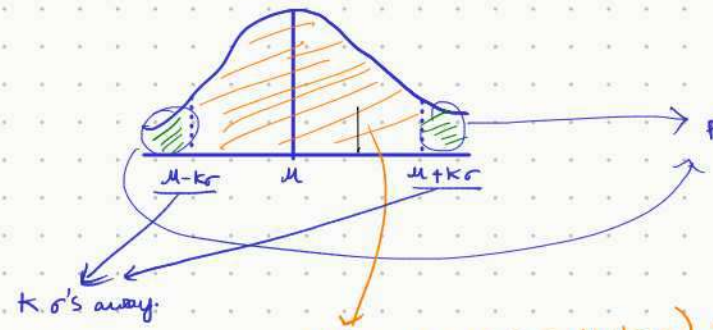
→ Chebyshev's Inequality states that

if X is a random variable with finite mean μ and non zero & finite σ

$$P\left(|X-\mu| \geq k\sigma\right) \leq \frac{1}{k^2}$$

k std deviations

68 → 1 std dev away
95 → 2 std dev away
99.7 → 3 std dev away



$$\Rightarrow P(\mu-k\sigma < X < \mu+k\sigma) \geq 1-\frac{1}{k^2}$$

→ From chebyshev's inequality we can conclude that 75% of any distributions lies within $[(\mu-2*\sigma)$ & $(\mu+2*\sigma)]$ and 90% within $[(\mu-3*\sigma)$ and $(\mu+3*\sigma)]$

→ In case the range is not symmetric, there are asymmetric variations of chebyshev as well.

Uniform Distributions:-

- → Discrete
- → Continuous.

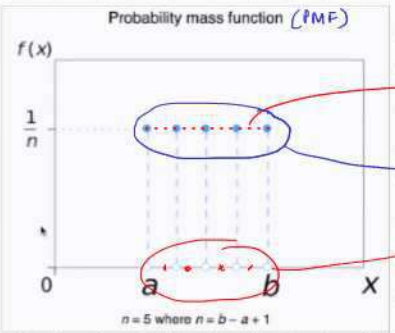
} Just like random variables

PDF (Probability Density Functions) are drawn for continuous random variables.
PMF (Probability Mass Functions) are drawn for discrete random variables.

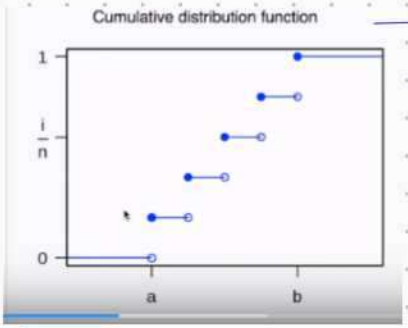
→ Probability of getting a value in dice is same for all values. This is called equiprobable.

→ CRV has $N(\mu,\sigma)$ as parameters.
DRV has (a, b) as parameters.
↳ both are numbers.
and
$n = b-a+1$. Another formula for calculating $n = \frac{(b-a+k)}{k}$ where k is the common difference.
↳ number of outcomes.

→ In uniform distributions all the variables are equiprobable. ie, $\frac{1}{n}$



This line can't be drawn because these points don't exist in uniform distribution
∴ Only these does exist in PMF



→ CDF

→ This is called a non smooth function

Props of URV :-

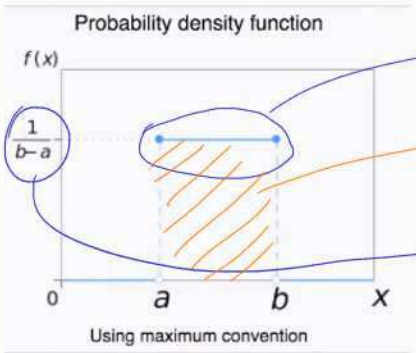
Mean $= \frac{a+b}{2}$

Median $= \frac{a+b}{2}$

Variance $= \frac{(b-a+1)^2-1}{12}$

Skewness $= 0$

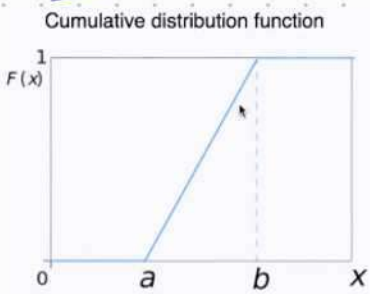
Continuous Random Variable :-



→ Line can exist here because it's continuous.

→ $\frac{1}{b-a}$ because this area has to be 1. The length is $b-a$. So the height is $\frac{1}{b-a}$.

CDF :-



Properties of CURV :-

Parameters: a, b

$$PDF = \begin{cases} \frac{1}{b-a} & \text{for } x \in [a,b] \\ 0 & \text{for everything else.} \end{cases}$$

Mean $= \frac{a+b}{2}$

Median $= \frac{a+b}{2}$

Variance $= \frac{(b-a)^2}{12}$

PDF → Density of data at the point

PMF → Probability of finding that point there.

Random Number generators :-

→ Most random number generators generate uniform random variables unless explicitly specified.

→ Python's random.random() picks a number b/w 0 & 1 with uniform probability.

ex :-

→ Picking 30 random values from iris

```
n=150 //Length of iris
m=30 //Sample size
p = m/n   //0.2
sample_data = []
for i in range(0,n):
    if random.random() ≤ p:
        sample_data.append(Iris[i,:])
len(sample_data)
```

↳ ≈30

random.random() picks value b/w 1 & n with equal probability.
chances of getting val ≤ 0.2 = 20%.
30 of 150 = 20%.

→ An applications of continuous uniform random variables.

Bernoulli & Binomial Distributions:-

→ Discrete Distributions
→ Bernoulli has only two outcomes. Value 1 has probability p & value 0 has probability of 1-p

ex:-

$X \sim \text{Bernoulli}(p=0.5)$

$$PMF = \begin{cases} q=(1-p) & \text{for } k=0 \\ p & \text{for } k=1 \end{cases}$$

Mean = p
Variance = pq

Binomial Random variable:-

Let X be tossing a coin

$X \sim \text{Bernoulli}(p=0.5)$

Let Y = {Number of heads when coin is tossed n times (let n=30)}

$Y \in \{0, 1, 2, 3, \ldots, 10\}$

$Y \sim \text{Binomial}(n, p)$

p → probability of value 1
n → number of trials

Parameters:-

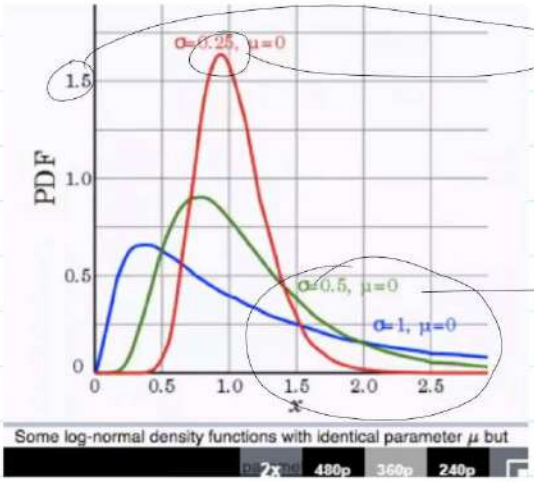
$$PMF = \binom{n}{k} p^k (1-p)^{n-k} \quad \text{for } P(Y=k)$$

Not used often in Machine Learning.
→ If in Binomial distribution, n=1 ⇒ it's a Bernoulli's distribution.
→ Bernoulli's is unbiased.
→ Haberman's dataset is not Bernoulli Dist as Attributes of Haberman are continuous Random variable & Bernouli, Binomial is performed on discrete random variable having fixed outcomes.

Log Normal Distributions :-

$X \sim \text{Log-Normal}(\mu, \sigma)$ if and only if $\log(X)$ is Normally distributed.

↳ Natural log aka ln



The PDF here is > 1, but its integral, the CDF will be lesser than 1 because the region is narrow
(The PDF can be > 1 but it can't be > 1 for bigger ranges)

as σ increases, curve becomes more skewed.

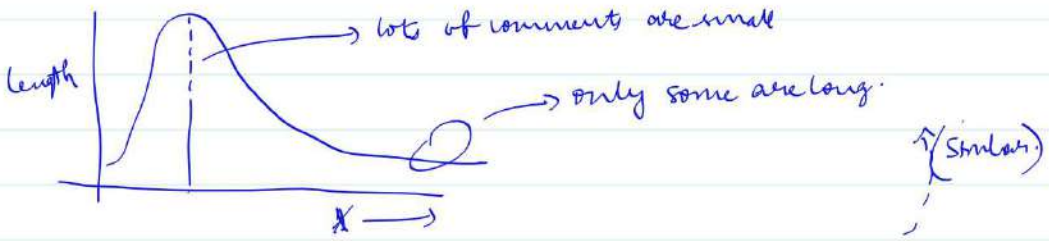
→ For log-normal, the mean is not where the curve peaks.

$$PDF = \frac{1}{x\sigma\sqrt{2\pi}} e^{-\frac{(\ln(x-\mu))^2}{2\sigma^2}}$$

$$\text{Skewness} = (e^{\sigma^2}+2)\sqrt{e^{\sigma^2}-1}$$, Not 0 like seen in Normal Distribution.

Applications :-

→ lengths of comments in internet forum posts follow log-normal distribution.



→ User's dwell time on online articles follows a log normal distribution.

→ In economics, 97-99% population's income follows log-normal. Distributions of higher income follows Pareto Distribution.

→ If we are given a log normal distribution, we can easily find its normal distribution & apply ML models.

Checking $X \sim \text{Log-normal}(\mu, \sigma)$ or not :-

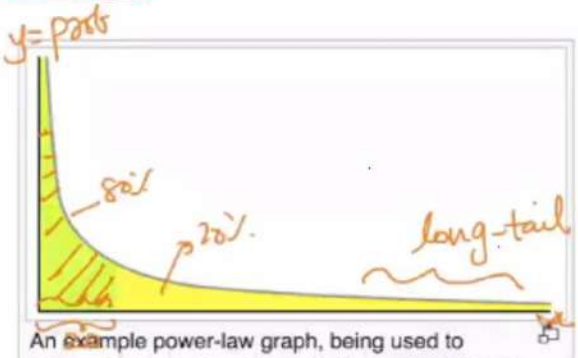
$x_1 \; x_2 \; x_3 \; x_4, \ldots, x_n$

$\ln(x_1) \; \ln(x_2) \; \ldots \; \ln(x_n)$

$(y_1) \; (y_2) \ldots\ldots (y_n)$ ⟶ QQ - plot ⟶ check if it's gaussian.

→ log normal more common than normal distributions IRL.

Power-Law :-



An example power-law graph, being used to demonstrate ranking of popularity. To the right is the long tail, and to the left are the few that dominate (also known as the 80–20 rule).

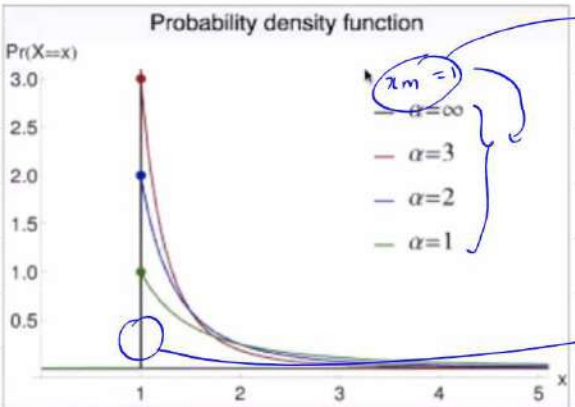
If distribution follows power-law, it's called as a Pareto distribution

Occurs a lot in nature.

Parameters of pareto distribution :-

$x_m > 0$ (Scale) $\longrightarrow (\mu)$

$\alpha > 0$ (shape) $\longrightarrow (\sigma)$



Pareto Type I probability density functions for various α with $x_m = 1$. As $\alpha \to \infty$ the distribution approaches $\delta(x - x_m)$ where δ is the Dirac delta function.

→ peak

as $\alpha \downarrow \rightarrow$ tail size ↑

→ $\alpha = \infty$ (Dirac Delta function)

Applications :-

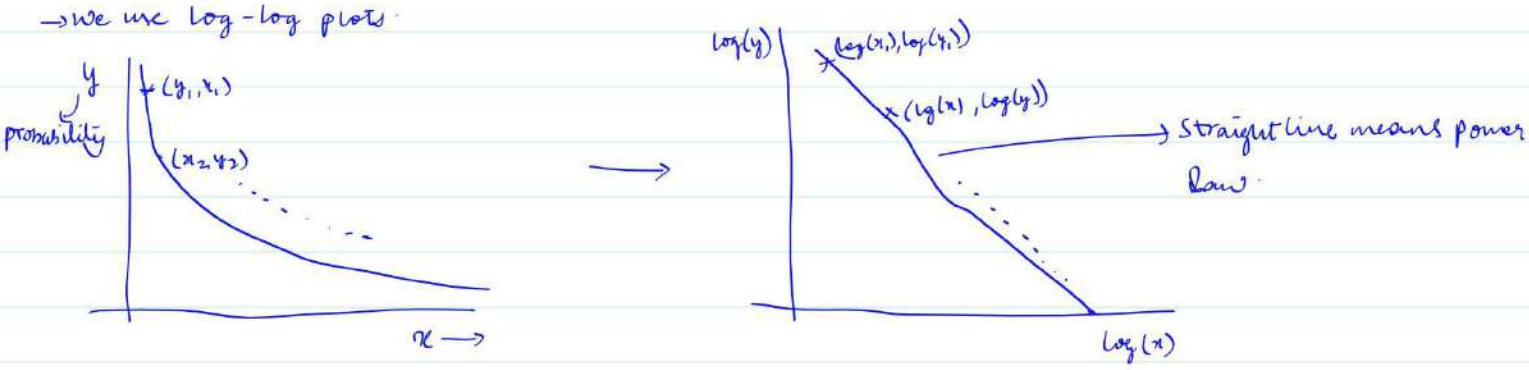
→ size of human settlements (villages, cities etc.)

→ Filesize distribution of Internet.

→ HDD error rate

How to check for pareto/powerlaw? :-

→ We use log-log plots.



→ To check for pareto, use Q-Q plot with pareto & observations.

Box Cox Transform :-

In ML we sometimes make assumptions that data is Gaussian Normal Distributed.

$X \xrightarrow{?} Y$ → We use Power transform / Box Cox transform for this conversion.

(pareto) (Gaussian)

$X \rightarrow$ Pareto $\rightarrow [x_1, x_2, x_3, \ldots, x_n] \rightarrow x$

$Y \rightarrow$ Normal $\rightarrow [y_1, y_2, y_3, \ldots, y_n] \rightarrow Y$

Steps:-

① boxcox(X) $\rightarrow$ returns 'λ' [λ computation is complex and No need to memorize]

② $$y_j = \begin{cases} \dfrac{x_i^{\lambda} - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \log(x_i) & \text{if } \lambda = 0 \end{cases} \quad \forall j : 1 \rightarrow n$$

$\Rightarrow x_i$ is log-normal

How?

scipy-stats.boxcox(x) $\rightarrow$ returns boxcox power transformed array and λ value.

Note:- Box Cox transform is not guaranteed to work on all pareto/powerlaw distributions. It only works on some of them. We need to apply boxcox & observe QQ plot to be certain.

Applications of Gaussian Distributions :-

→ useful in generating random numbers.

→ There are 100s of distributions.

Why are there so many? :- A well studied distribution gives a theoretical model on how a random variable behaves. There are 100s of distributions.

Most commonly used distributions :-

① Bernoulli

② Uniform Distribution

③ Binomial Distribution

④ Normal Distribution

⑤ Poisson Distribution

⑥ Exponential Distribution

Co-Variance :-

. If there are two datasets that are related, to find the relationship b/w these, we have 3 types of measurements

① Covariance ② Pearson Correlation Coefficient ③ Spearman Rank Correlation Coefficient

ex:- students height & weight

	h	w
s_1	150.	54
s_2	162	72
s_3	135	42
s_4	180	80

is there a relationship b/w height & weight?

$$cov(x, y) = \frac{1}{n} \sum (x_i - \mu_x) * (y_i - \mu_y)$$

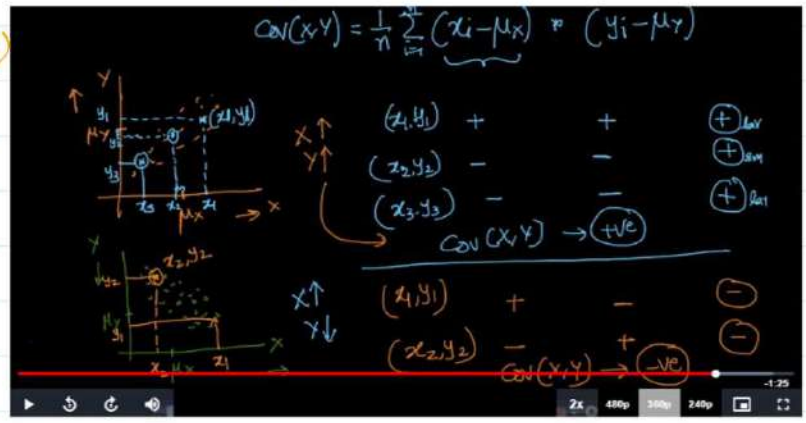
$$\left(\text{similar to } var(x) = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu_x) * (y_i - \mu_x) \right)$$

$cov(x, x) = var(x)$

$cov(x, y) = +ve$ if $x \uparrow \Rightarrow y \uparrow$

$= -ve$ if $x \uparrow \Rightarrow y \downarrow$

(reasoning)



Drawbacks :-

$$\underbrace{cov(\underset{(cms)}{height}, \underset{(kg)}{weight}) \neq cov(\underset{(ft)}{height}, \underset{(lbs)}{weight})}$$

same data but diff covariance.
The sign will be the same
Magnitude of covariance is also important as it tells how strongly related two RVs are.

→ Correlation $= \dfrac{\text{Covariance}(x, y)}{\sigma_x \sigma_y}$

→ Extreme outliers drastically affect covariance. So they need to be removed with the noise & other outliers.

If we replace mean in covariance formula with median it's called robust covariance.

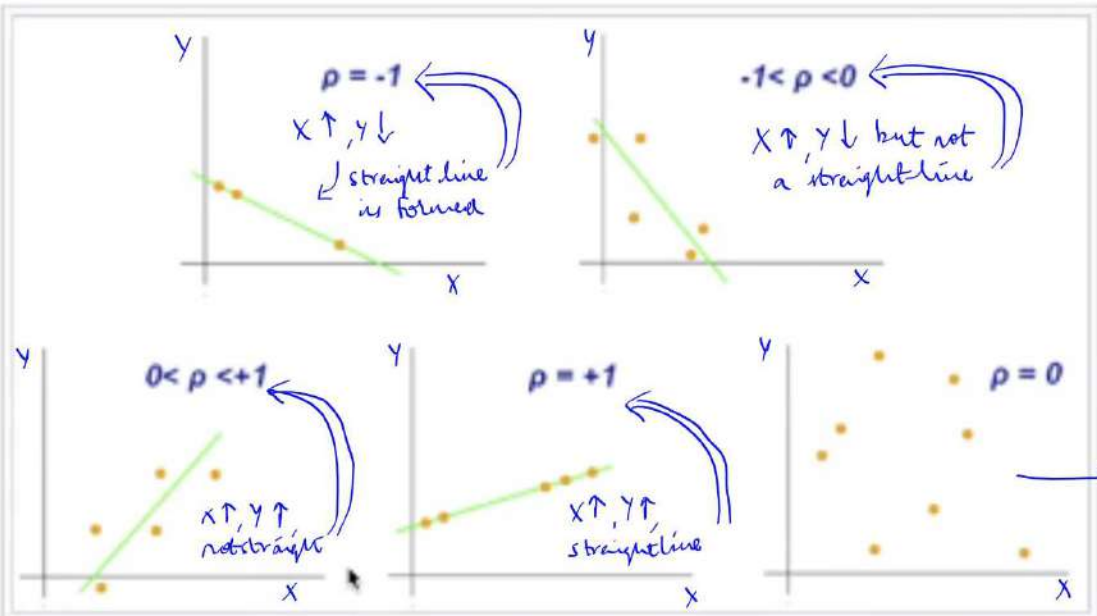
Pearson Correlation Coefficient (PCC) :-

$$\rho_{x,y} = \frac{cov(x, y)}{\sigma_x \sigma_y}$$

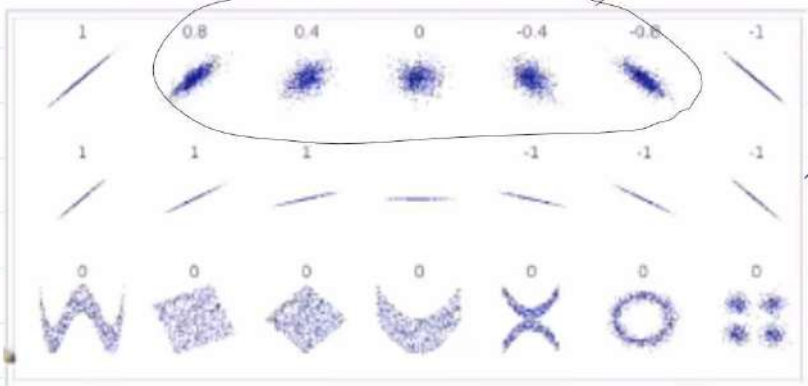
→ Cov does not take variability into consideration. This does.

→ $-1 \leq \rho \leq +1$

→ PCC is heavily biased towards linear dependant variables.



Examples of scatter diagrams with different values of correlation coefficient (ρ)

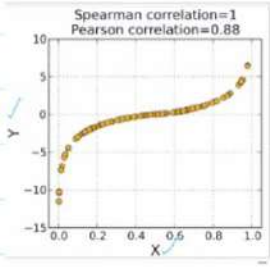


→ As it becomes flatter, value decreases.

→ PCC does not care about the slope of the line.

→ Does not capture complex / non linear relationships

Monotonically non decreasing → if $x_2 > x_1$ and corresponding $y_2 \geq y_1$
monotonically increasing → if $x_2 > x_1$ and corresponding $y_2 > y_1$



→ Correlation is what defines the change in one item when another item is modified.

Spearmen Correlation Coefficient:-

→ Useful when we have slightly non linear relationships.

ex:-

	X	Y	r_x	r_y
S_1	160	52	4	3
S_2	150	66	2	4
S_3	170	68	5	5
S_4	140	46	1	1
S_5	158	51	3	2

5 → largest x

1 → smallest x

During rank calculations, if two vals have same value, then their mean is taken as rank. Fractional rank (0)
Same rank depends on the implementation.

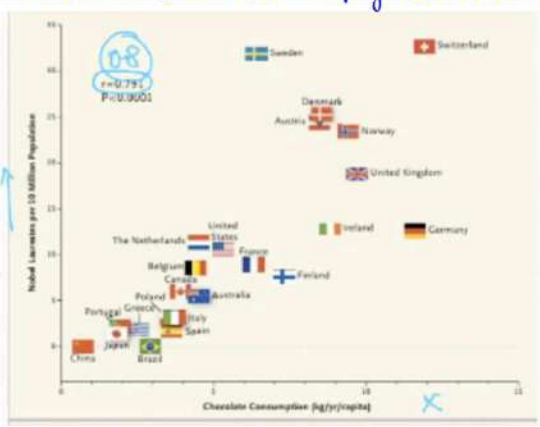
$r = \rho_{r_x, r_y}$ → Ranks are strictly increasing.

r → Spearmen coefficient

$\rho = 1 \Rightarrow x \uparrow \; y \uparrow \Rightarrow r = 1$
$\rho = -1 \Rightarrow x \uparrow \; y \downarrow \Rightarrow r = -1$
→ irrespective of whether they are linear or not. As long as there is monotonicity, it will work.

→ Correlation does not imply causation. For causation there are things like causal models.

ex:-



Applications of Correlations :-

ex:- (Salary & house square footage) → used by real estate brokers.
(# of years of education & income) → not looking for cause. Correlation only.
(Time spent on ecommerce website & money spent in the next 24 hours) → Amazon, Flipkart etc,
(# of unique visitors in one day & $ sales in one day) → ecommerce websites.
(dosage of medicine & blood sugar level of person) → medicine.

→ Ideas from causal inference can be used to determine causations.

Confidence Interval:-

Let X : heights with $\underbrace{x_1, x_2, x_3, \ldots, x_{10}}_{\text{Random Sample with size 10}}$

estimate the population mean of $X = \mu$

point estimate ← $\mu \approx \bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$ → Simple Average

μ → population mean, $\bar{x}$ → sample mean.

As n increases, $\bar{x} \to \mu$

Point estimate of $\mu = \frac{1}{10} \sum_{i=1}^{10} x_i = 168.5$ cm (Assume)

We can say $\mu \in [162.1, 174.9]$ with 95% probability → This is better than just saying this

μ → population mean

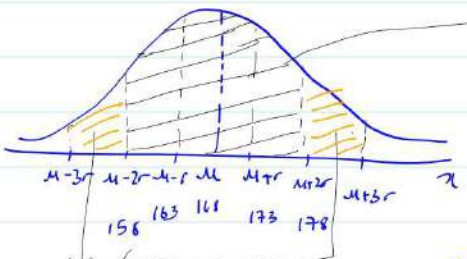
[162.1, 174.9] → Interval; 95% → Confidence → Confidence Interval

This statement means that when new samples are taken & their means are calculated, there is a 95% chance that it falls in this interval. Doesnt mean that 95% of the data lies in this interval.

Computing Confidence Interval gives a distribution :-

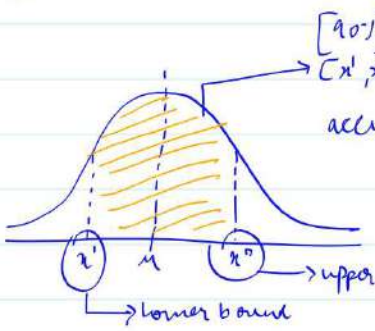
We can calculate CI easily if we know the underlying distribution.

Let $X \sim N(\mu, \sigma)$ with $\mu = 168$, $\sigma = 5$ cm



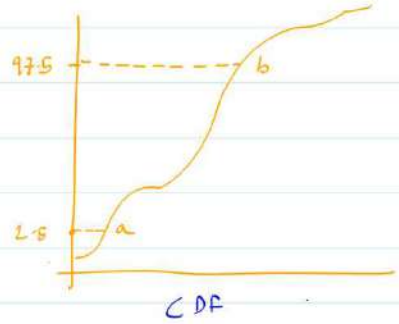
→ $(\mu - 2\sigma, \mu + 2\sigma)$ contains 95% of the data → c (Confidence interval)

2.5% + 2.5% = 5% → Remaining 5% of the data.



[90%] → $[x', x'']$ with 90% confidence. We can use normal distribution tables to get the accurate values.

→ CDF can be used to calculate confidence interval.



[a, b] has confidence interval of 95%.

Confidence Interval of Random Variable :-

Let $X \sim F(\mu, \sigma)$ $\{x_1, x_2, x_3, \dots, x_{10}\}$ $n = 10$

unknown dist.

What is the 95% CI of μ of this dist?

↳ case ①:- $\boxed{\sigma \text{ is given}}$

Using CLT, $\bar{x} = \frac{1}{10} \sum_{i=1}^{n} x_i$

$$\bar{x} \sim N\left(\mu, \frac{\sigma}{\sqrt{n}}\right)$$

μ, σ

$$\Rightarrow \mu \in \left[\bar{x} - \frac{2\sigma}{\sqrt{n}}, \bar{x} + \frac{2\sigma}{\sqrt{n}}\right] \text{ with 95\% of CI.}$$

$(\mu - 2\sigma)$ $(\mu + 2\sigma)$

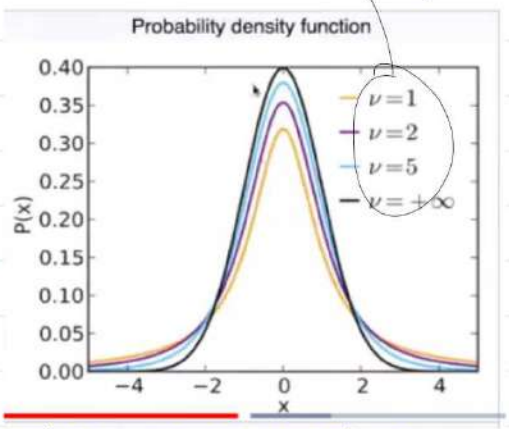
Looks like a gaussian curve.
As degrees of freedom increases, peak also increases.

↳ case ②:- σ is also unknown. ⇒ CLT can't be applied.

In this case we use t-distribution. It states

$\bar{x} \sim t(n-1)$

↳ degrees of freedom.



t-dist is created to estimate mean of RV when σ is unknown.

→ To estimate stuff like σ / 90th percentile etc., we use Bootstrap based Confidence Intervals.

Confidence Interval Using Bootstrap :-

→ prog based techniques for CI & other statistics

Let $X \sim F(\mu, \sigma)$ task: estimate CI for median of X

X: sample size 10 $\{x_1, x_2, x_3, \dots, x_{10}\}$

Using only this sample, compute median of X.

Steps:- ① Generate new samples using this sample. ⇒ Some values may repeat. It's called sampling with repetition.

$x_1^{(1)}, x_2^{(1)}, x_3^{(1)}, x_4^{(1)}, \dots, x_m^{(1)}$ such that $m \leq n$

↑

Random sample of size m generated from S

→ We can use uniform random variables to pick these values ($U(1, n)$ - There can be reps)

$$S = \{x_1, x_2, x_3, \ldots, x_n\}$$

↓ Using sampling with repitition - K samples.

Size $m \le n$. These are known as Bootstrap samples.

$$\begin{cases} S_1 : x_1^{(1)}, x_2^{(1)}, x_3^{(1)}, \ldots, x_m^{(1)} \longrightarrow m_1 \\ S_2 : x_1^{(2)}, x_2^{(2)}, x_3^{(2)}, \ldots, x_m^{(2)} \longrightarrow m_2 \\ S_3 \quad \vdots \\ \vdots \\ S_K : x_1^{(K)}, x_2^{(K)}, x_3^{(K)}, \ldots, x_m^{(K)} \longrightarrow m_K \end{cases}$$

→ medians. If variances are computed instead, we can calculate variance CI.

Lets say we get $m_1, m_2, \ldots, m_{1000}$

↓ sort these in increasing order.

$$m_1', m_2', m_3', \ldots, m_{1000}'$$

$$m_1', \ldots m_{25}' \ldots m_{975}', \ldots, m_{1000}'$$

⇓

95% confidence interval $= [m_{25}', m_{975}']$

→ This technique is called non-parametric technique. (No assumptions are made about distribution)

→ Used a lot in ML.

How? :-

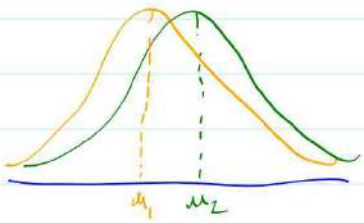
```
from sklearn.utils import resample
:
s = resample(x, n_samples = m)
```

→ To generate bootstrap sample.

→ Sample size.

Hypothesis Testing :-

Assume there are two classes cl1, cl2. Both class heights are taken. Histograms are plotted.



Question :- Is there a difference in heights of both classes.

Sol :- ① Choosing a test statistic

↳ $\mu_2 - \mu_1$ → if it's 0, then there's no difference

② Null Hypothesis (H_0) :-

This uses proof by contradiction.

H_0 : Assume there's no diff in μ_1, μ_2.

Alternative Hypothesis (H_1) :- Assume there's diff in μ_1 & μ_2.

We assume H_0 is true & prove H_1 wrong (or) We assume H_1 is true & prove H_0 is wrong.

③ p-value :- It says what is the prob of observing $(\mu_2 - \mu_1)$ if H_0 is true.

⇒ Assume H_0 is true. ⇒ $\mu_1 \approx \mu_2$ ⇒ cl1 ≈ cl2

if p-value = 0.9 ⇒ probability of observing $\mu_2 - \mu_1$ is 0.9 if H_0 is true

Since p-val is high, we accept H_0 is true.

if p-value = 0.05 ⇒ 5% chance that 10cm if H_0 is true.

Hypothesis Testing (Additional Video):-

Cl1 → 50 students → μ_1 (mean)

Cl2 → 50 students → μ_2 (mean)

Step 0:- Δ (or) $X = \mu_2 - \mu_1 = 10$ (assume) → This is an observation. Ground truth.

Step 1:- What is the probability of observing a value of $X \geq 10$ cm if there was no difference in class heights.
given a hypothesis

$$P(X \geq 10 \text{ cm} \mid H_0)$$

Teststatistic

Null hypothesis

if p value is small = 0.01 (or) 1%.

→ < 5% is typically considered small

$$\Rightarrow P(X \geq 10 \text{ cm} \mid H_0) = 0.01 \text{ or } 1\%.$$

⇓

true. (Since its an observation that's been made already)

↓

Since the probability is small, implies H_0 is less probable ⇒ H_0 may not be true.

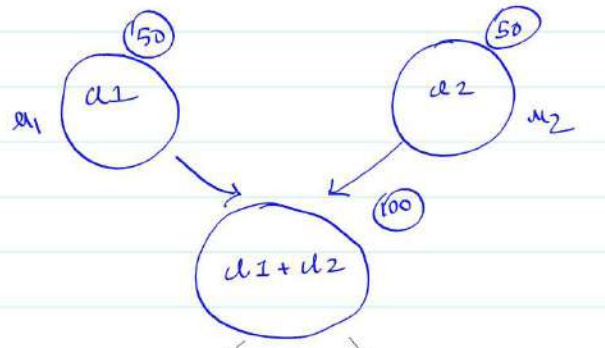
∴ We reject the null hypothesis (H_0)

⇒ The original assumption that there's no difference is incorrect

⇒ There is difference b/w the heights.

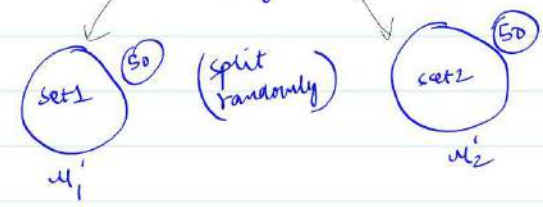
How to calculate p-value? We only have sample data from two classes

The way we calculate is, we use resampling & permutationtesting.



$X = \mu_2 - \mu_1$

Null hypothesis (H_0):- There is no diff in class heights. We simulate null hypothesis on class heights (combined)



repeated multiple times

⇓

$\underbrace{\delta_1, \delta_2, \delta_3, \ldots \delta_{10,000}}$

Simulated X's with H_0 as true.

↳ sorted: $\underbrace{\delta_1', \delta_2', \delta_3', \ldots}_{\leq 10 \;\; 8000} \mid \underbrace{\ldots, \delta_{10000}'}_{\geq 10 \;\; 2000}$

$\delta_1 = \mu_2' - \mu_1'$

↳ simulated diff in class heights with sample size = 50 & with H_0 = true

↳ Since randomly picked.

$\Rightarrow P(X \geq 10 \text{ cm} \mid H_0) = 20\% \Rightarrow H_0$ is rejected.

$\delta_1', \delta_2', \delta_3', \ldots \ldots \mid \delta_{10000}'$

≤ 10 | ≥ 10

9900 | 100 $\Rightarrow P(X \geq 10 \text{cm} \mid H_0) = 1\%$. $\Rightarrow$ H_0 is true. Hence accepted.

→ How to pick null hypothesis? We pick H_0 such that it is easy to simulate. Hence we picked that there is no diff b/w the heights.

Hypothesis Testing:-

ex ①:- Given a coin, determine if coin is biased towards heads or not.

$\Rightarrow P(H) > 0.5$

Not biased $\Rightarrow P(H) = 0.5$

Solving using base probability:-

experiment:- Flip a coin 5 times and count number of heads

↳ X → random variable. Also called a test statistic

performing experiment:- H, H, H, H, H $\Rightarrow$ $X = 5$ → observation by experimentation.

$P(\underbrace{X = 5}_{\text{Observation}} \mid \underbrace{\text{coin is not biased towards heads}}_{\text{Assumption}}) = P(\text{obs} \mid H_0)$

⇓

Null Hypothesis (H_0):- Coin is not biased towards heads.

$P(X = 5 \mid H_0) = \Rightarrow \frac{1}{2} * \frac{1}{2} * \frac{1}{2} * \frac{1}{2} * \frac{1}{2} = \frac{1}{2^5} \approx 0.03 = 3\%$.

Five heads in 5 tosses.

coin is not biased, $P(H) = 1/2$

$P(X = 5 \mid H_0) = 3\%$. $\Rightarrow$ There is 3% chance of getting 5 heads in 5 flips if the coin is not biased

p-value [Probability (Observation by experiment | assumption is true) = 3%

↳ low (<5%)

Since it's low, H_0 may be incorrect.

$\Rightarrow$ Null hypothesis is not true & we reject hypothesis

$\Rightarrow$ We reject the idea that coin is not biased.

Alternative hypothesis (H_1) ← [$\Rightarrow$ Coin IS biased.

Rejecting H_0 = Accepting H_1

Accepting H_1 = Rejecting H_0

→ This expt is reliant on how many times coin has been flipped (Sample Size)

if coin was flipped 3 times, then $P(X \mid H_0) = 1/2^3 = 1/8 = 12.5\%$ → Not low Significant

$\Rightarrow$ We accept H_0

$\Rightarrow$ Coin is not biased.

3 important things in Hypothesis testing:-

1. Design of the experiment
2. Defining Null hypothesis.
3. Design of test statistic 'X'.

→ Typical p-value in scientific computing = 0.05%. In some cases 0.01 is also used -
↳ when you don't want any outliers at all.

→ In olden days, when there wasn't enough computing power, people used to use distributions & use 68-95-99/ chebyshev to get the pvalue.

K-S Test for Similarity of two distributions:-

Assume $X_1 : [x_1, x_2 \dots x_n]$ (n size sample)
$X_2 : [x_1', x_2', \dots x_m']$ (m size sample)

Question:- Are both X_1 & X_2 coming from the same distribution.

How?:- We plot the CDF of both samples. If they both overlap, that means they belong to the same distribution.

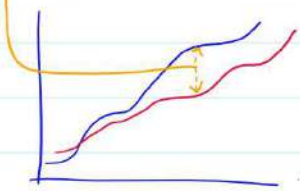
Null hypothesis, H_0 : X_1 & X_2 have the same distribution

Test statistic, difference b/w the CDFs at any given point

⇒ if CDFs completely overlap, difference = 0

Test statistic, $D_{n,m} = \sup_x \left| F_{1,n}(x) - F_{2,m}(x) \right|$

supremum.
The maximal value



→ KS test was used before the concept of resampling & hypothesis testing. So after lot of research, they came up with closed form equation

Null hypothesis is rejected at level α, if [α = required p-level]

$$D_{n,m} > c(\alpha)\sqrt{\frac{n+m}{nm}}$$

α	0.20	0.15	0.10	0.05	0.025	0.01	0.005	0.001
$c(\alpha)$	1.073	1.138	1.224	1.358	1.48	1.628	1.731	1.949

ex:- if $n = 1000$, $m = 5000$, $\alpha = 0.05$, then if $D_{n,m} > 1.358\sqrt{\frac{1000+5000}{1000 * 5000}}$

⇒ if $D_{n,m} > 0.047$, we reject H_0 at 0.05 sig level.
↳ Max diff b/w CDFs

In general $c(\alpha) = \sqrt{-\frac{1}{2}\ln\alpha}$

QQ plot is a graphical method whereas as KS test gives us what level of 'α' we are accepting/rejecting.

Implementing KS Test :-

from scipy import stats

stats.kstest(dataset, 'norm') → type of dist to compare against.

→ Returns two values → ① $D_{n,m}$

② p-value

→ KS Test is a weak test as it is not designed for testing normality. It is better to use Anderson-Darling test to test for normality. KS Test is good for comparing distributions.

→ As long as p-value > 0.05, we accept the null hypothesis.

Hypothesis testing ②:-

task: $\boxed{\text{city 1}}$ $\boxed{\text{city 2}}$

determine if $\mu(\text{heights}_{\text{city 1}}) = \mu(\text{heights}_{\text{city 2}})$

Population. Not sample.

Getting actual population is expensive. So we use sample mean.

how? :-

$$\begin{array}{cc} \text{city 1} & \text{city 2} \\ \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_{50} \end{bmatrix} & \begin{bmatrix} h_1' \\ h_2' \\ \vdots \\ h_{50}' \end{bmatrix} \end{array}$$

random

μ_1 → sample mean ← μ_2

ex:- 162 cm 167 cm

test statistic :- $X = \mu_2 - \mu_1 = 167 - 162 = 5$ cm

Null hypothesis (H_0) :- No diff in population means.

compute :- $p(x = 5\text{cm} \mid H_0)$ [Probab of observing diff of 5cm in sample mean heights of size 50 b/w C_1 & C_2 if there is no diff in mean heights.]

→ Difference in sample means with sample size 50 (observation)

case 1 :- $p(x = 5 \mid H_0) = 0.2 = 20\%$

↳ There is 20% chance of observing diff in means as 5 if H_0 is true

There is no difference in heights.

Significant ⇒ Assumption is true ∴ H_0 is accepted.

case 2 :- $P(x = 5 \mid H_0) = 0.02 = 2\%$ ⇒ low. Assumption is incorrect. Alternative Hypothesis (H_1) is accepted ⇒ Heights are not the same.

Resampling & Permutation test for cities example:-

$x = \mu_1 - \mu_2$ → diff in sample means (Let it be 5 cm)

H_0 = Null hypothesis (No difference in means. For this example)

$$\begin{array}{cc} c_1 & c_2 \\ \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ \vdots \\ h_{50} \end{bmatrix} & \begin{bmatrix} h_1' \\ h_2' \\ h_3' \\ \vdots \\ h_{50}' \end{bmatrix} \end{array}$$

Step ①:- $S = \{h_1, h_2, h_3, \ldots, h_{50}, h_1', h_2', h_3', \ldots, h_{50}'\}$

$c_1 \cup c_2$

Step ②:-

S

S_1 S_2 → Randomly selected two sets 50 size each (Resampling)

$\{\cdots\}$ $\{\cdots\}$ ★ (We are simulating the null hypothesis) ★

μ_1 μ_2

$\mu_2 - \mu_1$ → 3 cm (δ^1)

repeat, $\mu_2 - \mu_1$ → −2 cm (δ^2)

" , $\mu_2 - \mu_1$ → 1 cm (δ^3)

⋮ ⋮ ⋮

k times, $\mu_2 - \mu_1$ → 6 cm (δ^k), Let $k = 1000$

Step ③:- Sort δ_i's

$\delta_1' \le \delta_2' \le \delta_3' \le \delta_4' \ldots \le \delta_k'$

case ①:- $P(\text{diff} \ge 5\text{cm} \mid H_0)$

$$\underbrace{\delta_1' \le \delta_2' \le \delta_3' \le \delta_4' \ldots \le \delta_{800}'}_{\substack{\le 5\text{cm} \\ 80\% \text{ of the data}}} \le \underbrace{\ldots \delta_{1000}'}_{\substack{> 5\text{cm} \\ 20\% \text{ of the data}}}$$

$P(\text{Obs-diff} \mid H_0) = P(x \ge 5\text{cm} \mid H_0) = 20\%$ → p-value

↳ significant (> 5%)

∴ Assumption must be true. H_0 is accepted.

case ②:- $\underbrace{\delta_1' \le \delta_2' \le \delta_3' \le \delta_4' \ldots \le \delta_{970}'}_{97\% \text{ of the data}} \le \underbrace{\ldots < \delta_{1000}'}_{3\% \text{ of the data}}$ → Simulated values originated from assuming that null hypothesis is true.

$P(x \ge 5\text{cm} \mid H_0) = 3\%$ = p-value is not significant. Assumption must be incorrect. H_0 is rejected.

→ if $P(\text{obs} \mid H_0)$ is small, we reject H_0 because the observation is real but we got a small probability implying H_0 is false.

→ if $P(\text{obs} \mid H_0)$ is very high in this case, it implies error in sample & if we increase sample size, well be fine.

How to use hypothesis testing :-

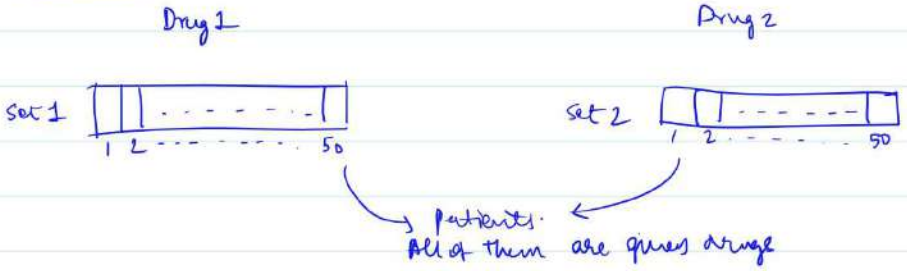
KS testing → used hypothesis testing to determine if two RV have the same dist or not.
→ application of hypothesis testing.

ex:- Designing medicines

Drug 1 (Company 1) (Available in the market already) → Reduces fever in 4 hours

Drug 2 (Company 2) (new drug) → claim: Can reduce fever significantly faster than Drug 1.

experiment :- To determine if claim is true or not.



100 patients → 50 – drug 1, 50 – drug 2 } randomly separated.

Each patient's fever recovery time after taking drug is tracked.

$\mu_1 = 4$ hrs $\mu_2 = 2$ hrs

Instead of concluding drug 2 is better from this example, we perform hypothesis testing

① H_0 :- D_1 & D_2 take same time to reduce fever → Null hypothesis

H_1 :- D_1 & D_2 take different times → Alternate hypothesis

② Test Statistic :- $X = \mu_2 - \mu_1 = 4 - 2 = 2$ hours

compute $P(X \geq 2 \mid H_0)$.

Let $P(X \geq 2 \mid H_0)$ is small → < 5%. ⇒ The probability of observing $X \geq 2$ is very small (< 5%)

$X \geq 2$ → observation

⇒ Observation & H_0 do not agree with each other.

⇒ H_0 is incorrect since observation is true & real

⇒ we reject H_0 & accept H_1

→ Resampling & permutation tests are used to find $P(X \geq 2 \mid H_0)$

③ Significance level (α) :- Generally significance level = 5%.

if p-Value < α, we reject H_0. Else we accept H_0.

We take smaller α if we want to be very very sure (ex:- Medicine, $\alpha = 1\%$)

Ecommerce, $\alpha = 3\%$.

Proportional Sampling :-

Let d be an array

d_1	d_2	d_3	d_4	d_5
2.0	6.0	1.2	5.8	20.0
1	2	3	4	5

$n = 5$

Task :- pick an element from the n elements such that Probability of picking an element is proportional to d_i

Normally, each has the same probability.
But acc to proportional sampling, d_5 should have the highest probability.
d_3 should have the least probabability

How?

Step 1a :- $S = \sum_{i=1}^{n} d_i = 35$ (in this case) [compute the sum]

Step 1b :- $d_i' = d_i / S$ [normalizing values using the sum]

Now all values lie b/w 0 & 1 and sum of all values $(d_i') = 1$

Step 1c :- Computing Cumulative Normalized Sum

$d_1' = 0.0571$	$\tilde{d}_1 = d_1 = 0.0571$
$d_2' = 0.171428$	$\tilde{d}_2 = \tilde{d}_1 + d_2' = 0.228528$
$d_3' = 0.0343$	$\tilde{d}_3 = \tilde{d}_2 + d_3' = 0.262828$
$d_4' = 0.1657$	$\tilde{d}_4 = \tilde{d}_3 + d_4' = 0.428528$
$d_5' = 0.5714$	$\tilde{d}_5 = \tilde{d}_4 + d_5' = 1.00$

Cumulative Normalized Value

Step 2 :- Sample one value from uniform random variable b/w 0 & 1

$r = \text{numpy.random.uniform}(\underbrace{0.0, 1.0}_{\text{Range}}, \underbrace{1}_{\text{Number of values}})$

Step 3 :- Proportional Sampling.

if $r \le \tilde{d}_1$ return 1
else if $r \le \tilde{d}_2$ return 2
else if $r \le \tilde{d}_3$ return 3
else if $r \le \tilde{d}_4$ return 4
else if $r \le \tilde{d}_5$ return 5

Dimensionality Reduction :-

2D, 3D → Scatter plots can be used to visualized -

4D, 5D, 6D, ... → pair plots can be used but not enough.

What if there are more than 10 Dimensions?

10D, 100D, 1000D → can't be visualized easily

n-Dimensions $\xrightarrow{\text{reduced}}$ 2D / 3D

techniques used :- ① Principal Component Analysis (old)

② t-Distributed Stochastic Neighbouring Analysis (modern state of the art)

In Iris dataset PL, PW, SL, SW → dependent variables / Features } Data is 4 dimensional.

Species → Target Variable / class

Row vector, Column Vector :-

iris → each (flower) → [SL, PL, SW, PW]

flower → datapoint

[SL, PL, SW, PW] → real values

→ i^{th} Datapoint : $x_i \in \mathbb{R}^D$ → x_i is a D dimensional column vector

$\mathbb{R}^D$ → Real Space → Real Numbers

$$x_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ x_{i3} \\ \vdots \\ x_{id} \end{bmatrix}$$ → column vector

→ Default is a column vector.

→ $y_i = [\, y_{i1} \;\; y_{i2} \; \cdots \; y_{in} \,]_{1\times n}$ → row vector

→ Generally assume it's a column vector.

Representing a dataset :-

$D = \{x_i, y_i\}_{i=1}^{n}$ → number of datapoints

$y_i = \{$setosa, virginica, versicolor ...$\}$ → in case of iris dataset

$x_i \in \mathbb{R}^D$, $x_i \in \mathbb{R}^4$

[x_i → data points y_i → class labels]

→ For a single value, X will be column vector & Y will be a single value. For the overall dataset, X will be matrix & Y will be a column vector.

Representing Dataset as a Matrix :-

let $D = \{x_i, y\}$

$x_i \in \mathbb{R}^d$

$y_i \in \{S, Vi, Ve\}$

d - features. By default it's a column vector

$$X = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ \vdots \\ n \end{array} \begin{array}{c} \begin{array}{cccc} f_1 & f_2 & f_3 & \cdots f_d \end{array} \\ \begin{bmatrix} & & & \\ & & & \\ & & & \\ \leftarrow & (x_i^T) & \rightarrow & \\ & & & \\ & & & \end{bmatrix}_{n\times d} \end{array}$$

→ columns represents a Features

Row represents a datapoint

Transpose because

$$X = \begin{array}{c|c} & 1 \;\; 2 \;\; 3 \;\; 4 \;\; \cdots \; j \; \cdots \; n \\ \hline \begin{array}{c} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_i \\ \vdots \\ f_d \end{array} & \begin{bmatrix} & & & & & & \\ & & & & \updownarrow & & \\ & & & & x_i & & \\ & & & & \downarrow & & \end{bmatrix}_{d \times n} \end{array}$$

→ Alternative way - Both are valid

Data Preprocessing : Feature Normalization :-

$$X = \begin{array}{c|c} & f_1 \;\; f_2 \;\; f_3 \cdots f_j \cdots f_d \\ \hline \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{array} & \begin{bmatrix} & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ \longleftarrow & & x_i^T & & & & \longrightarrow \\ & & & & & & \\ & & & & & & \end{bmatrix}_{n \times d} \end{array} \qquad Y = \begin{bmatrix} y \\ \\ \\ y_i \\ \\ \end{bmatrix}$$

Preprocessing = set of math operations before building ML models.

Done before dimensionality reduction.

obtain data → preprocessing → datamodelling

Column normalisation :-

$$\text{Let} \quad X = \begin{array}{c|c} & f_1 \;\; f_2 \;\; f_3 \cdots f_j \cdots f_d \\ \hline \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_i \\ \vdots \\ x_n \end{array} & \begin{bmatrix} & a_1 & & \\ & a_2 & & \\ & a_3 & & \\ & \vdots & & \\ & a_i & & \\ & \vdots & & \\ & a_n & & \end{bmatrix}_{n \times d} \end{array} \qquad Y = \begin{bmatrix} \\ \\ \\ \\ \\ \end{bmatrix}$$

This is a column. Each column is preprocessed in the same way.

The max & min values from $a_1, a_2, \ldots\ldots, a_n$ are taken a_{max} & a_{min}.

Now $a_1, a_2, a_3, \cdots, a_n$ are transformed to $a_1', a_2', a_3', a_4', \cdots, a_n'$ where

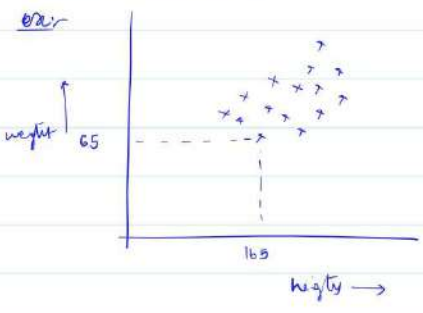
$$a_i' = \frac{a_i - a_{min}}{a_{max} - a_{min}}$$

Column Normalization ← All a_i' values lie b/w 0 & 1

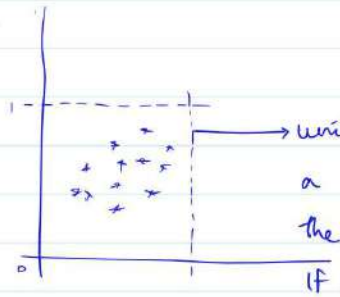
why? :- Get rid of scales and entire data is in the same scale. ex:- Heights (cm), weights (Kg) dataset when normalised becomes [0 & 1] dataset.

Geometric Intuition :-

ex:-



⟹



→ unit square. All of the data is squished into a 1×1 square without disturbing any of the relationship b/w the data.

If data is 3D, unit cube, ...

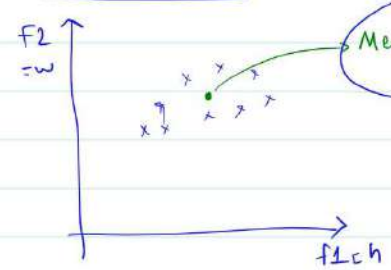
Mean of a matrix :-

$$X = \begin{array}{c|c} & f_1 \;\; f_2 \;\; f_3 \cdots f_i \cdots f_d \\ \hline \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ \vdots \\ i \\ \vdots \\ n \end{array} & \begin{bmatrix} & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ \longleftarrow & & x_i & & & & \longrightarrow \\ & & & & & & \\ & & & & & & \end{bmatrix}_{n \times d} \end{array}$$

mean vector $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$

→ mean vector = central vector.

Geometric Intuition :-



$\bar{x} = [h_{\bar{x}}, w_{\bar{x}}]$, $h_{\bar{x}} = \text{mean}(h_i)_{i=1}^{n}$

$w_{\bar{x}} = \text{mean}(w_i)_{i=1}^{n}$

Column Standardization :-

column normalization → b/w 0 & 1, gets rid of scale, converts to a unit hyper cube.

column standardization → more commonly used in practice.

$$X = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ \vdots \\ i \\ \vdots \\ n \end{array} \begin{array}{c} \begin{array}{ccccccc} f_1 & f_2 & f_3, & \cdots & f_j & \cdots & f_d \end{array} \\ \begin{bmatrix} & & & & a_1 & & \\ & & & & a_2 & & \\ & & & & a_3 & & \\ & & & & \vdots & & \\ & & & & a_i & & \\ & & & & \vdots & & \\ & & & & a_n & & \end{bmatrix}_{n \times d} \end{array}$$

$a_1, a_2, a_3, a_4, \cdots, a_i, \cdots a_n \longrightarrow$ any distribution

$\downarrow$

$a_1', a_2', a_3', \cdots\cdots, a_i', \cdots a_n' \longrightarrow \text{mean}\{a_i'\}_{i=1}^{n} = 0$

$\text{std-dev}\{a_i'\}_{i=1}^{n} = 1$

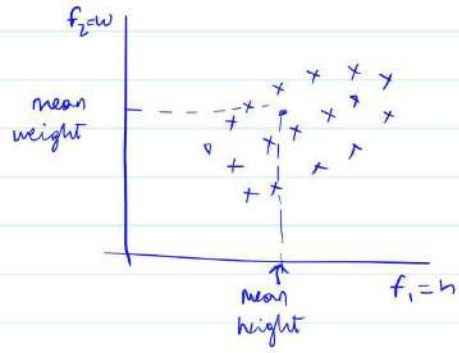
$\bar{a} = \text{mean}\{a_i\}_{i=1}^{n}$ $\qquad s = \text{Std-dev}\{a_i\}_{i=1}^{n}$

$a_i' = \dfrac{a_i - \bar{a}}{s}$ $\longrightarrow$ Looks similar to standard Normal Variate $\quad z = \dfrac{x-\mu}{\sigma}$

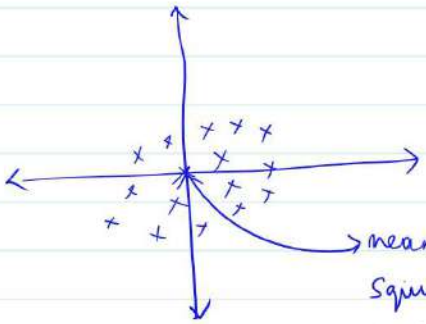
$x \sim N(\mu, \sigma)$

$z \sim N(0,1)$

Geometric Intuition :-



$\xrightarrow[\text{standardization}]{\text{column}}$



→ mean-vector is at origin. Squished such that standard deviation is 1 while keeping relationship b/w data intact

Column Standardization = mean centering + scaling.

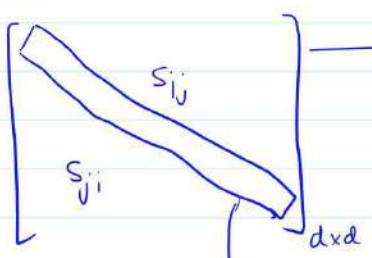
Covariance of data Matrix :-

$$X = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ \vdots \\ i \\ \vdots \\ n \end{array} \begin{array}{c} \begin{array}{ccccccc} f_1 & f_2 & \cdots & j & \cdots & f_d \end{array} \\ \begin{bmatrix} \\ \\ \\ \\ \\ \\ \\ \end{bmatrix} \end{array}$$

covariance matrix of X, $S = \begin{bmatrix} & & \\ & s_{ij} & \\ & & \end{bmatrix}_{d \times d}$ → Square Matrix

$s_{ij} = \text{cov}(f_i, f_j)$ $\qquad \text{cov}(x,y) = \dfrac{1}{n} \sum_{i=1}^{n} (x_i - \mu_x)(y_i - \mu_y)$

$cov(f_i, f_j) = var(f_i)$
$cov(x, x) = var(x)$
$cov(f_i, f_j) = cov(f_j, f_i)$ } Properties of covariance.

$S =$ 

→ Square symmetric matrix

→ variance of features (diagonal elements) = 1. Since variance of element with itself is 1.

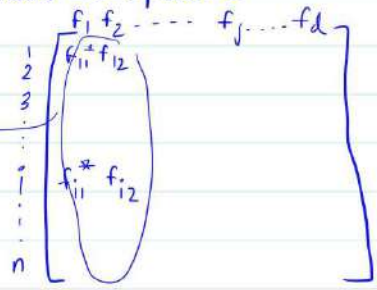
Let X : column standardized → mean $(f_i) = 0$
std-dev $(f_i) = 1$

ex:- $cov(f_1, f_2) = \frac{1}{n} \sum_{i=1}^{n} (x_{i1} - \mu_1)(x_{i2} - \mu_2)$

→ 0. Since column standardized.

$\Rightarrow cov(f_1, f_2) = 1/n \sum_{i=1}^{n} x_{i1} * x_{i2}$ component wise multiplication = dot product

$\Rightarrow cov(f_1, f_2) = (f_1^T f_2) * 1/n$



$S = \underbrace{\frac{1}{(n-1)} (X^T)_{d \times n} (X)_{n \times d}}_{\text{Assuming sample data is given}}$ = $d \times d$ matrix. (assuming X has been std'ized)

If population is given,

$S = \frac{1}{n} (X^T)(X)$

→ For any parameter θ, our estimate $\hat{\theta}$ is unbiased if, $E\{\hat{\theta} - \theta\} = 0$
→ Covariance is not a dimensionality reduction technique. Covariance matrix is used for PCA
→ We can use np.cov to get covariance.

Exploration of MNIST Dataset :-

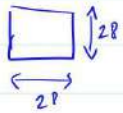
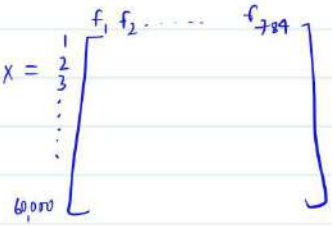
→ 28×28 size image for each number
→ 60K train + 10K test
→ $D = \{x_i, y_i\}_{i=1}^{60K}$ $y_i \in \{0, 1, 2, \ldots 8, 9\}$ x_i : 

image → numerical matrix (NOT DATA MATRIX) → This is flattened
→ All row elements are placed one after the other resulting in long matrix with single column
→ (row flattening)

$\Rightarrow X =$ 

[784×1] size matrix

→ tSNE is used to convert 784 dimension dataset to 2 dimensional

Principal Component Analysis :-

MNIST → 784 dim → 2 dimensional & visualize -

d dim → d' dimensions & d' < d

→ PCA is one of the simplest dimensionality reduction technique.

* Feature Engineering → Reducing dimensionality by creating new features from given features (PCA, t SNE)
* Feature Selection → Reducing dimensionality by selecting subset of important features from given features. (Forward Feature Selection, Backward elimination)

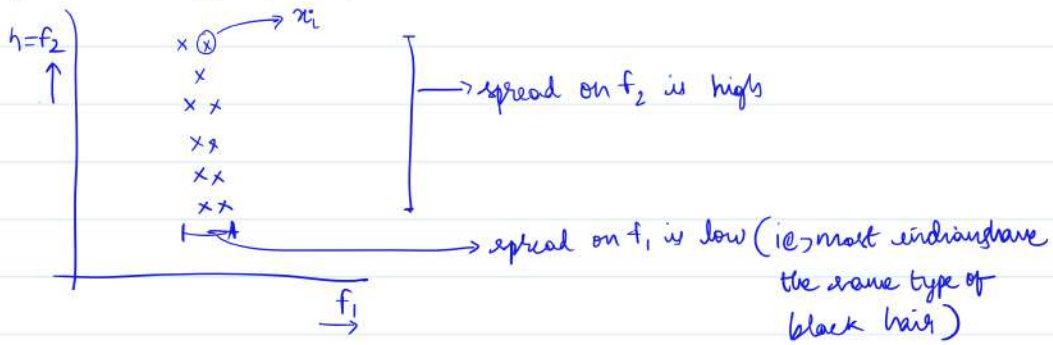
→ Categorical features :- Contain finite number of category/groups. They might not have a logical order (gender, country etc)

Continuous features :- Numerical features that have an infinite number of values b/w two values (time, price etc)

Geometric Intuition :-

Let $X = \begin{array}{c|cc} & f_1 & f_2 \\ \hline 1 & & \\ 2 & & \\ 3 & & \\ \vdots & & \\ n & & \end{array}$

f_1 = blackness of hair $\in \mathbb{R}$

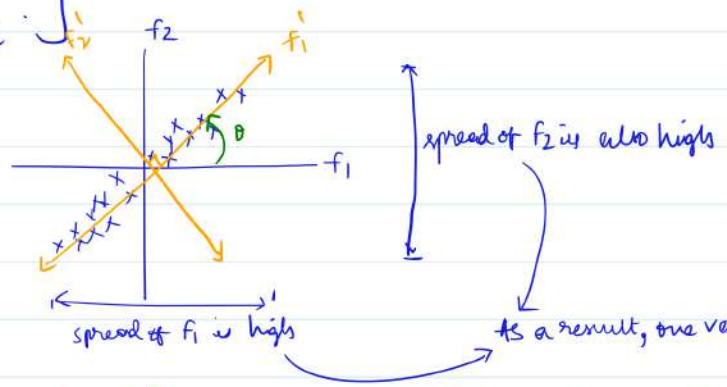


If we want to convert this 2dimensional data to 1dimension we can skip f_2 as it has low variance & keep only f_1

$\Rightarrow X' = \begin{array}{c|c} & f_2 \\ \hline 1 & \\ 2 & \\ \vdots & \\ n & \end{array}$

⇒ We are preserving the direction with maximal spread/variance/information

Let $X = \begin{array}{c|cc} & f_1 & f_2 \\ \hline 1 & & \\ 2 & & \\ 3 & & \\ \vdots & & \\ n & & \end{array}$ ⇒



As a result, one variable can be eliminated

(i) Find $f_1' \perp f_2'$ (ii) Drop f_2' (iii) Project x_i's onto f_1', then

spread on $f_2' << $ spread on f_1' 2D → 1D

→ We want to find direction of f_1' such that the variance of x_i's projected onto f_1' is max. Achieved by rotating axis by θ° to find f_1' with max variance.

→ Before going for PCA, column standardization needs to be performed. As it brings feature into a standard metric.

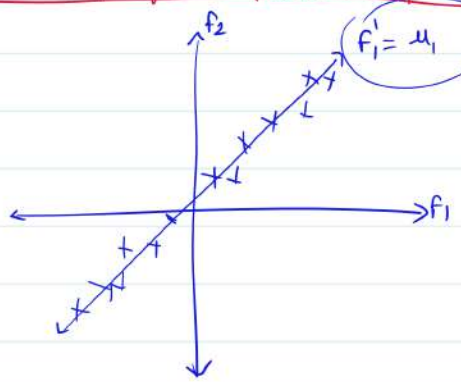
→ PCA is useless for dimensionality reduction if features are completely uncorrelated.

Variance inflation :- When variance of dataset increases with addition of new columns that are completely correlated to already existing features of dataset (completely correlated = repeated features).

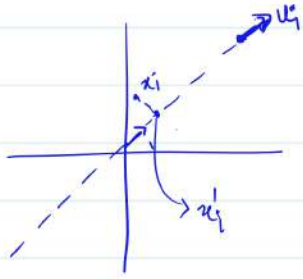
First principal component :- The direction in space along which projections have the largest variance.

Second principal component :- Direction which maximizes variance among all directions orthogonal to the first.

Mathematical objective function of PCA :-



→ we only need the direction. u_1 is the unit vector/direction $\Rightarrow \|u\| = 1$



x_i' = projection of x_i on u_i

$D = \{x_i\}_{i=1}^n$

$D' = \{x_i'\}_{i=1}^n$

$$x_i' = \text{proj}_{u_i} x_i = \frac{u_i \cdot x_i}{\|u_i\|^2} = u_i^T x_i$$

$$x_i' = u_1^T x_i$$

$$\Rightarrow \bar{x}_i' = u_1^T \bar{x}$$

$\bar{x}_i'$ → mean $\{x_i'\}_{i=1}^n$

$\bar{x}$ → mean $\{x_i\}_{i=1}^n$

Objective :- Find u_1 such that $\text{var}\{\text{proj}_{u_1} x_i\}_{i=1}^n$ is maximal.

$$\text{var}\{\text{proj}_{v_1} x_i\}_{i=1}^n = \frac{1}{n}\sum_{i=1}^n \left(\underbrace{v_1^T x_i}_{} - \underbrace{v_1^T \bar{x}}_{}\right)^2$$

$\bar{x}_i$ → If X is column standardized, then mean becomes 0, $\Rightarrow$ this value $= 0$

$$v_1^T x_i = \underbrace{\underset{1\times n}{v_1^T} \times \underset{n\times 1}{x_i}}_{1\times 1 \rightarrow \text{scalar}}$$

$$\Rightarrow \text{var}\{x_i'\}_{i=1}^n = \frac{1}{n}\sum_{i=1}^n (v_1^T x_i)^2$$

We want to find v_1 such that this is maximized while keeping $v_1^T v_1 = 1 = \|v\|^2$

↓ Objective of an optimization problem

↓ constraint

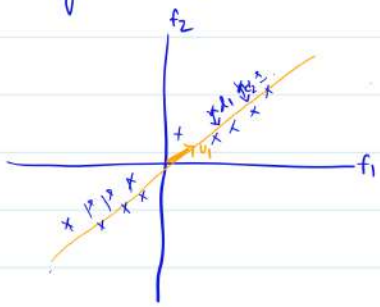
+

Optimization problem.

Data → Column standardization → After standardization, all features have same spread so we dont know which one to choose → Axis transformation → Find the unit vectors in order for axis to line up with features that have the highest spread

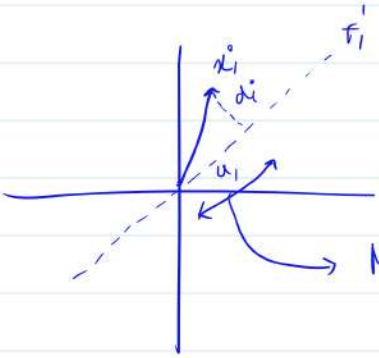
Alternative Formulazation of PCA :- Distance Minimization

previously :- Find v_1 that maximizes variance.



Alternatively we want to find out v_1 that minimizes $\sum_{i=1}^{n} d_i^2$

$$\min_{v_1} \sum_{i=1}^{n} d_i^2$$



v_1 : unit vector $\quad v_1^T v_1 = 1 = \|v_1\|^2$

projection of x_i on v_1 = $\text{proj}_{v_1} x_i = v_1^T x_i$

Pythagorean theorem: $d_i^2 = \|x_i\|^2 - (v_1^T x_i)^2$
$= x_i^T x_i - (v_1^T x_i)^2$

$$\Rightarrow \min_{v_1} \sum_{i=1}^{n} \underbrace{\left(x_i^T x_i - (v_1^T x_i)^2\right)}_{d_i^2,\ x_i \text{ is known}} \text{ such that } v_1^T v_1 = 1 \longrightarrow \text{distance minimization PCA}$$

Previously, we had $\max_{v_1} \frac{1}{n} \sum_{i=1}^{n} (v_1^T x_i)^2$ such that $v_1^T v_1 = 1 \longrightarrow$ Variance Maximization PCA.

The v_1 that achieves these two is the same

Eigen Values & Eigen Vectors in PCA :-

Solution to optimization problems :-

Let $X = \begin{array}{c|c} & 1\ 2\ 3 \cdots d \\ \hline 1 & \\ 2 & \\ 3 & \\ \vdots & \\ n & \end{array}$ $\Big\}_{n \times d}$ $\longrightarrow$ column standardized $\Rightarrow \underbrace{S_{d \times d}}_{\text{Square Symmetric}} = X^T X$

eigen values :- $\lambda_1, \lambda_2, \lambda_3, \ldots$
eigen vectors :- $v_1, v_2, v_3, \ldots$

can get these using numpy

Eigen Values of $S = \underbrace{\lambda_1, \lambda_2, \ldots, \lambda_d}_{d \text{ values since } S \text{ is a } d \times d \text{ matrix}}$

Eigen Vectors $= \underbrace{v_1, v_2, v_3, \ldots, v_d}_{\text{one eigen vector for corresponding Eigen value.}}$

Assume $\lambda_1 > \lambda_2 > \lambda_3 > \lambda_4 \ldots > \lambda_d$

λ_1 → Maximal Eigen value.

Eigen Value def :- $\lambda_1 v_1 = S_{d\times d} v_1$

λ_1 → Scalar ⇓ Eigen Value

v_1 → vector ⇓ Eigen Vector

If $\lambda_1 > \lambda_2 > \lambda_3 \dots > \lambda_d$

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$

$v_1 \quad v_2 \quad v_3 \quad v_d$

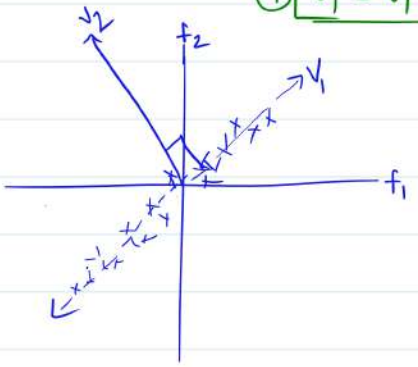
Any two v_i & v_j → $v_i \perp v_j$ ⇒ $v_i^T v_j = 0 = v_i \cdot v_j$

$u_1 = v_1$ = Eigen vector of $S (= X^T X)$ corresponding to the largest eigen value $(= \lambda_1)$

From Optimization problem: = Maximal Variance Direction

Steps to find u_1 :-

1. Column Standardize X
2. Find $S_{d\times d} = X^T X$
3. Compute 'd' Eigen Values & vectors
4. $\boxed{u_1 = v_1}$

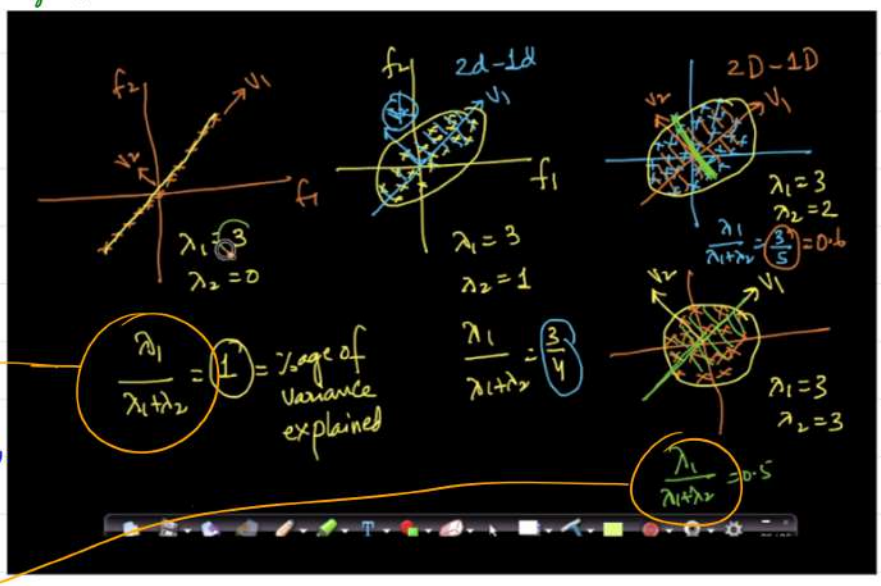


v_1 → First eigen vector with maximal variance.

v_2 → Second Eigen Vector with maximal variance.

v_d → Least variance last eigen vector.

Significance of λ's (Eigen Values) :-



$\frac{\lambda_1}{\lambda_1+\lambda_2} = 1$ = %age of variance explained

How much info is present on v_1. Since the data is in a straight line, all of it on v_1, its 100%.

Since data is circular, 50% of it is on v_1 & 50% is on v_2.

Eigen value :- How much data is present on each Eigen vector.

Eigen Vector :- Which direction the first maximal is present & the second maximal is present.

PCA for Dimensionality Reduction & Normalization :-

$$X = \begin{array}{c|cc} & f_1 & f_2 \\ \hline 1 & & \\ 2 & & \\ 3 & & \\ \vdots & & \\ n & & \end{array} \xrightarrow{PCA} X' = \begin{array}{c|c} & v_1 \\ \hline 1 & \\ 2 & \\ 3 & x_i' \\ \vdots & \\ n & \end{array}$$

$x_i' = x_i^T v_1$ (from max variance method PCA)

Alternative method :-

$$X' = \begin{array}{c|cc} & v_1 & v_2 \\ \hline 1 & & \\ 2 & & \\ 3 & & \\ \vdots & & \\ n & & \end{array}$$

$\lambda_1 > \lambda_2 > \lambda_3 > \dots > \lambda_d$

$x_i' = [x_i^T v_1, x_i^T v_2]$

v_1, v_2 → Eigen vectors (top 2)

Eigen vals

→ PCA can not only be used for converting to 2D.

ex:- $X = \begin{array}{c|c} & 1 \quad 2 \quad 3 \cdots 100 \\ \hline 1 & \\ 2 & \\ 3 & \\ \vdots & \\ n & \end{array}$ → 100 features.

$\lambda_1 > \lambda_2 > \lambda_3 \cdots \lambda_d$

$v_1 > v_2 > v_3 \cdots v_d$

v_j

↓ PCA → 100 to 50d.

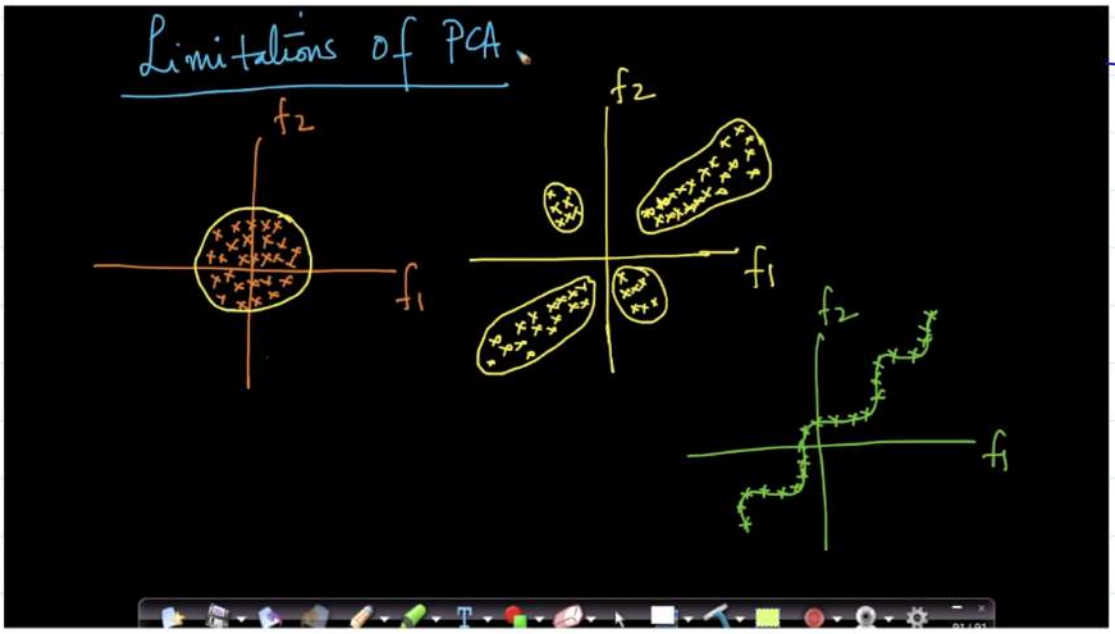
$X' = \begin{array}{c|c} & 1 \quad 2 \quad 3 \cdots 50 \\ \hline 1 & \\ 2 & \\ 3 & \\ 4 & \\ \vdots & \\ n & \end{array}$ $x'_{ij} = x_i^T v_j$

→ d features to d' features where $d' < d$

→ Sometimes it's asked such that PCA needs to preserve 99% of the variance & the original number of features = 100.

Then we look for a value like let $\dfrac{\lambda_1 + \lambda_2 + \cdots + \lambda_{51}}{\sum_{i=1}^{100} \lambda_i} = 0.99$ → $d' = 51$. It could be anything

→ sklearn inverse_transform can be used to convert d' to d dimensions.



→ In all these cases some information is lost.

Advantages :-

① Removes correlated features by reducing dimensions

② improves algorithm performance.

~~③ Reduces overfitting which mainly occurs when there are too many variables~~

④ improves visualization

Disadvantages :-

① Independent variables become less interpretable. They are converted to PCs.

② Data standardization is must before performing PCA.

③ Information loss - If number of PCs are not selected with care.

PCA for dimensionality Reduction (Not visualization) :-

$d \xrightarrow{PCA} d'$

↳ d' when its 2/3... it can be used for visualization sometimes, we just need to reduce the dimensions & visualization is not the main purpose.

ex:- $d = 784$, $d' = 200$, $n = 15000$

$$X_{15000 \times 784} \times V_{784 \times 200} = X'_{15000 \times 200}$$

↳ First 200 eigen values vectors are chosen.

$$\begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ v_1 & v_2 & \cdots & v_{200} \\ \downarrow & \downarrow & & \downarrow \end{bmatrix}$$

→ $\dfrac{\lambda_1 + \lambda_2 + \cdots + \lambda_{d'}}{\sum_{i=1}^{d} \lambda_i}$ gives the percentage of variance explained in d' dimensions. This helps us in picking the correct value of d'.

Generally we want to find 90% retaining d'.

→ Its not always guaranteed that applying ML model after PCA shows better results than just model. So performing both & using performance metrics helps pick which one is better.

t-Dist Stochastic Neighbourhood Embedding :- (t-SNE)

→ 2008 first paper (very new)

→ One of the best dimensionality reduction techniques (especially for visualization)

→ Multi dimensional scaling (MDS), Sammon Mapping, Graph based techniques are some of the other techniques -

→ d dimensions $\xrightarrow{\text{t-SNE}}$ 2D

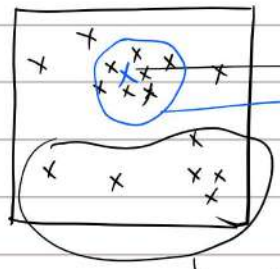
→ PCA tries to preserve the global structure of the data while t-SNE tries to preserve the local structure of the data.

→ t-SNE is great for visualization because similar items can be plotted next to each other & reduces crowding.

→ PCA gives best chance of ① preserving important information
② Generalizable results
③ Efficient dimension reduction
④ Visual methods.

Neighborhood of a point, Embedding :-

Let a d dimensional space be

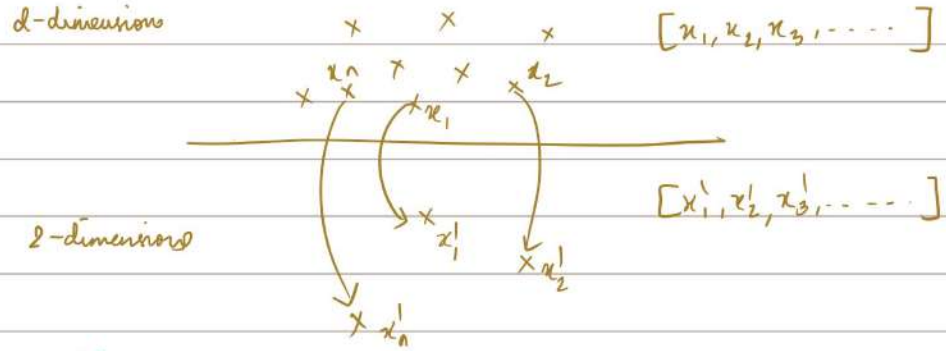


x_i , $N(x_i)$ = Neighbours of x_i

$N(x_i) = \{x_j$, such that x_i & x_j are geometrically close$\}$

→ Probably not neighbours of x_i

Embedding :- For every point in higher dimension space we place an equivalent point in lower dimension space (2D). Process is called embedding.



Geometric Intuition of t-SNE :-

d-dimensional	2-dimensional
$N(x_1) = \{x_2, x_3\}$	$N(x_1') = \{x_2', x_3'\}$
$N(x_4) = \{x_5\}$	$N(x_4') = \{x_5'\}$

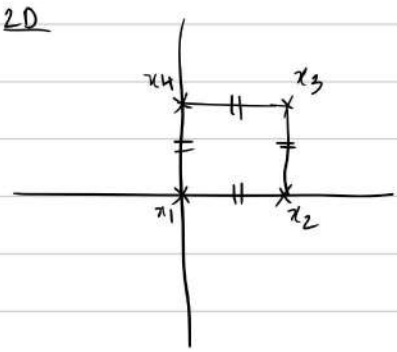
→ t-SNE tries to preserve distance in a neighbourhood.

→ $d(x_1, x_2) \approx d(x_1', x_2')$

→ It does not promise neighbourhoods are going to be at the same distance as that of d-dimensions.

Crowding Problem :-

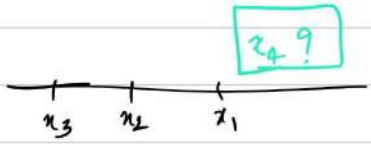
2D



All points are equidistant from one another

$\xrightarrow{SNE}$

1D



x_4 can't be placed at an appropriate position

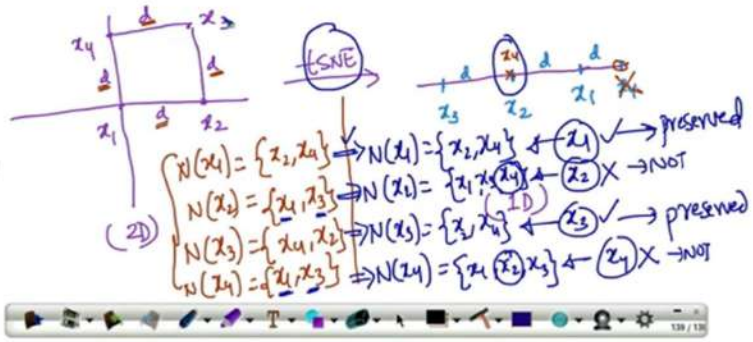
→ This is called the crowding problem.

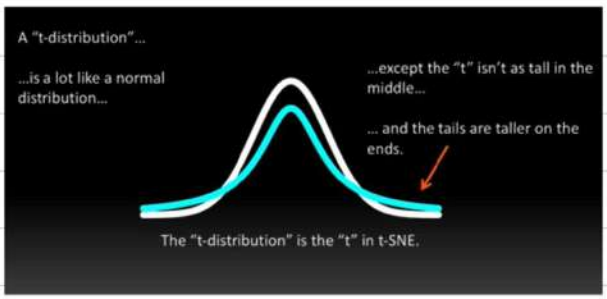
→ SNE couldn't resolve this. t-distribution is used to resolve this problem.

Why cant points be placed on top of another point ?

Crowding in t-SNE

Question : (by Marreddy Krishna Chaitanya Reddy @ Youtube)





→ t-distributions are used to find nearby points. t-distributions explain variance better than any other plot because it has more tail region.

→ t-SNE is an iterative algorithm. Every iteration embedding is performed.

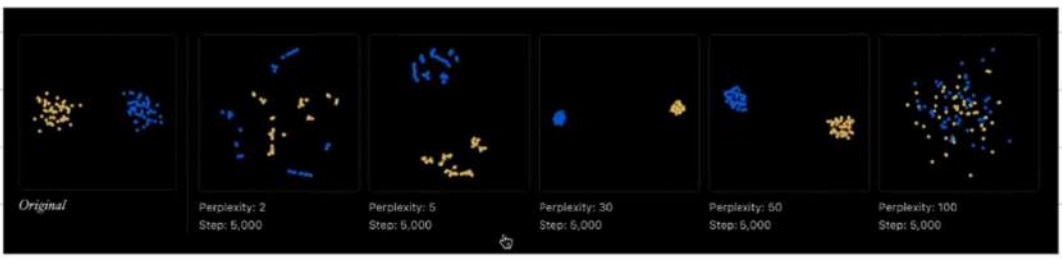
→ 2 parameters to t-SNE :-

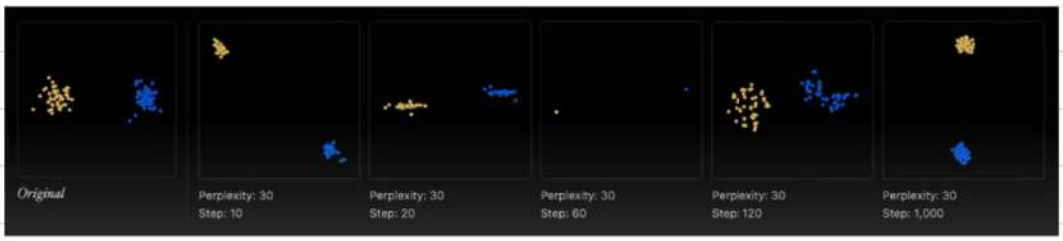
① perplexity (number of neighbors)

② step size (# of iteration)

→ Never run t-SNE only once. Modify/change the perplexity and the number of steps as the result varies as these values change.

→ Try to keep perplexity < # of data points.





(Keep changing step size till a figure/shape is consistent)

→ Data → [Algorithm] → same result every time → Deterministic Algorithm
→ different results each time → Probabilistic Algorithm.

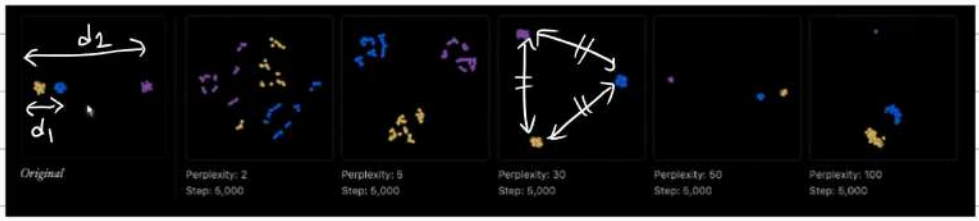
→ t-SNE is a probabilistic Algorithm.

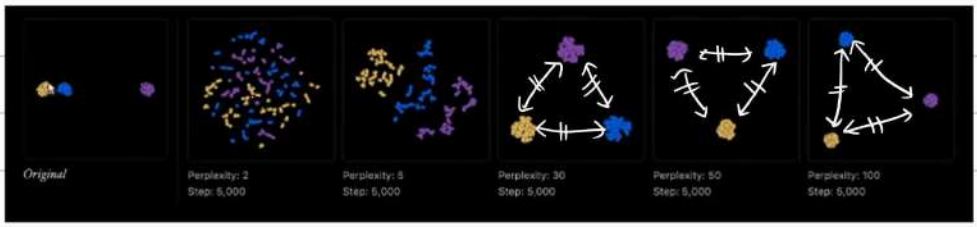
t-SNE
↳ Stochastic → Probabilistic

So as a result, everytime we might get diff results for same dataset & hyper parameters

→ t-SNE tries to expand dense clusters. And it tries to compress/shrinks sparse clusters.
So we can't say just because two clusters are of the same size, it doesn't mean it's a good configuration

→ t-SNE does not preserve local structure.





→ Never make conclusions from random data. t-SNE might group data points into clusters when used with random data but it doesn't mean anything. Always look at diff perplexities.

→ We might see shapes in t-SNE. Never draw conclusions from one perp value. Change the values & then look at the plot.

Conclusions :-

① Run steps/iterations till shapes stabilize
② perplexity $2 \leq p < n$.
③ Never run t-SNE once. Rerun with varying perplexities & step values to see if they are stable or not. Because conclusions can't be made from just one iteration.

→ In general we run it with a large enough number of steps as the result converges at a large number of steps. Then we change perplexity and see if shape changes significantly due to the change.

→ Since t-SNE is stochastic in nature (ie, different results each time) it's best not to use it for dimensionality reduction/classification tasks as we cannot reproduce the results
Reason why t-SNE does not work for classification is because it does not learn from previous data.

→ Epsilon in t-SNE is the learning rate.

→ t-SNE on MNIST tries to group points based on visual similarity.

→ On a high level, low perplexity preserves structure on a local level & high perplexity preserves the global structure

→ np.vstack(a,b) puts 'b' under 'a' like it's adding a new row to a.

→ UMAP (Uniform Manifold Approximation and Projection for Dimension Reduction) is a new technique from 2018 which is faster than t-SNE but yet to be rigorously tested.

→ sklearn.manifold refers to a space that is modelled in Euclidean Spaces. (t-SNE's topological space/curve is in Euclidean Space).

SQLite :-

→
```
pd.read_sql_query("""SELECT name FROM sqlite_master WHERE type='table' ORDER BY name; """, con)
```

	name
0	Reviews

→ Prints list of tables.

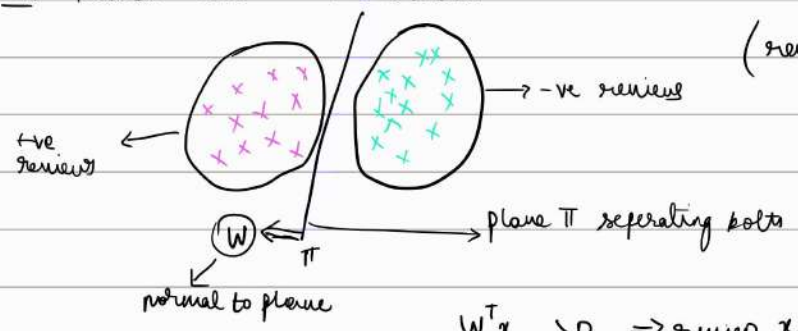
→ df.drop_duplicates (subset = {" ", " " ... "}, keep = "first") (Pandas)

↳ drops duplicate values with these column name & keeps the first occurance.

→ Use common sense to clean data. No general technique.

→ how to convert text to numerical vectors?

ex:- Review text → n-dim vector



(reviews are represented by 'X' after converting to vectors)

$W^T x_1 > 0$ ⇒ review x_1 is a positive review (on same side)

$W^T x_2 < 0$ ⇒ review x_1 is a negative review (on diff side)

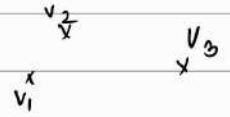
→ Rules for conversion from text to vector :-

$r_1 \quad r_2 \quad r_3$

assume semantic similarity b/w $(r_1, r_2) > (r_1, r_3)$

then distance b/w $(v_1, v_2) <$ dist b/w (v_1, v_3)

ie) if they are similar, vector must be close.



Bag of words :-

Step ① :- Constructing dictionary (set of all unique words)

Step ② :- Assume there are d unique words across all 'n' documents/reviews. [Collection of documents is called a corpus]

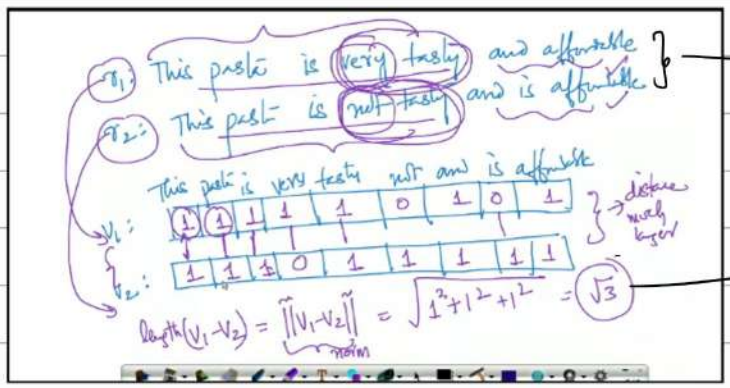
For each review (d dimensional vector)

r_1 :- | 0 | 1 | 1 | 2 | 1 | - - - - - | 0 |

r_2 :- ⋮

The number in box represents the number of times that word occured in the review.

Each word is a different dimension.



→ Diff sentences but distance is less compared to length 'd' (9)

Boolean bag of words / binary bag of words :-

Instead of counting number of occurrances, it only counts occurances -

$\|v_1 - v_2\| \approx \sqrt{\#\text{ of differing words}}$

→ We can notice that some of the words in BoW don't matter (is, and, a, the etc,) → Stopwords

→ Bag of words is still used for creating baselines with Generalized Linear Models.

→ Removing stop word = smaller dimensions data.

These are called text preprocessing steps.

→ from nltk.corpus import stopwords

→ We also make everything lowercase.

→ Stemming :- tasty, tasteful, tastey all have same meaning.

converting all into one word 'taste'. This is called stemming.

Porterstemmer } two techniques for stemming.

Snowball Stemmer

→ more powerful.

→ tokenization :- Breaking sentences into words. ex:- if we use space as delimiter, New York are separate words.

→ In BoW we are not taking semantic meaning.

ex:- delicious tasty convey same meaning. We use word2vec to solve this -

Lemmatisation is the algorithmic process of determining the lemma of a word based on its intended meaning. Unlike stemming, lemmatization depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document.

→ Algorithm combines inflected words into a common word called lemma.

{go, going, went} => 'go' is the lemma given by algo.

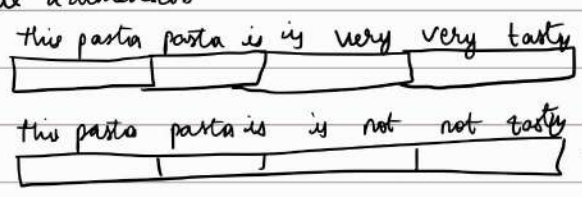
→ NLTK makes use of parts of speech. This is called tagging. Makes sense of context and helps pick the right lemma.

→ Stopwords are not removed in stemming. All words are reduced to base words and the tails are removed.

unigram / Bi-gram / n-gram :-

unigram :- each word is considered as a dimension

bigrams :- pairs of words



} The difference will be higher than that of unigrams.

n = 3 → trigrams

⋮

→ unigrams BoW completely discards sequence information

Bigrams, trigrams, ... retain partial sequence information.

→ But the catch is $n(\text{bigrams}) > n(\text{unigrams})$

→ As n increases, dimensionality increases drastically.

→ But if there are no repeated words in text, #unigrams > # bigrams > # trigrams, ...

TF-IDF :- (term frequency inverse document frequency) (industrialized version of BoW)

$$N \text{ docs} \begin{cases} r_1: & w_1 \quad w_3 \quad w_2 \quad w_5 \quad w_1 \\ r_2: & w_3 \quad w_1 \quad w_4 \quad w_1 \quad w_6 \quad w_2 \\ \vdots \end{cases}$$

w_1	w_2	w_3	w_4	w_5
2	1	1	0	1

$\vdots$

$$TF(w_i, r_j) = \frac{\#\text{ of times } w_i \text{ occurs in } r_j}{\text{total number of word in } r_j}$$

→ TF-IDF was originally meant for information retrieval.

→ TF gives how often word occurs. (Probability)

$$0 \le TF(w_i, r_j) \le 1$$

IDF :-

$$D_c = \{r_1, r_2, \ldots, r_n\}$$

$$D_c = \text{Document corpus} \begin{cases} r_1: & __ w_1 __ \\ r_2: & _____ \\ \vdots & __ w_1 __ \\ \vdots & ____ w_1 __ \end{cases}$$

$$\rightarrow IDF(w_i, D_c) = \log\left(\frac{N}{n_i}\right)$$

n_i → Number of docs that contain w_i

$$n_i \le N \Rightarrow \frac{N}{n_i} \geq 1$$

$$\Rightarrow \log\left(\frac{N}{n_i}\right) \geq 0$$

⇒ As n_i increases, N/n_i decreases. As N/n_i decreases, $\log(N/n_i)$ also decreases as log is monotonic function.

If a word is more frequent in corpus, then its IDF is low. If it's less frequent in corpus, then it's IDF is high

→ Combining both

$$D_c \begin{cases} r_1 :- & _____ \\ r_2 :- & _____ \\ \vdots \\ r_n :- & _____ \end{cases}$$

$v_1 :-$ [| | ..... | | ...]

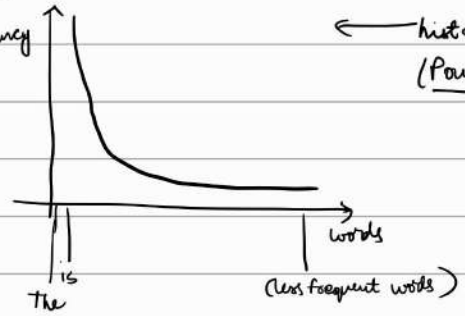
$$w_i = TF(w_i, r_1) * IDF(w_i, D_c)$$

→ Advantage is that we give more importance if → word is more frequent (from TF)
→ word is less frequent (from IDF)

→ Drawback :- Still no semantic importance.

Why do we use log in IDF ? :-

Zipf's law :- frequency



← histogram
(Power law distribution)

↳ If X is a power law distribution, it can be converted to Gaussian Distribution by using transforms like Box-Cox transform (we take log(X))

When tf-idf was introduced, no mathematical explanation was given as to why log was used in idf

→ Another reason :-

	N/n_i	$\log(N/n_i)$
"the" →	almost all docs ≈ 1	≈ 0
"is" →	" ≈ 1	" ≈ 0
⋮		
less frequent pages →	if 1 in 1000 → 1000	→ ≈ 6.9

→ lot of difference During calculation of TF-IDF, IDF will dominate if there's no log.

→ Sometimes if vocabulary is passed explicitly, words might not occur in training phase, but might appear in testing phase. As a result of this N/n_i is going to be ∞. To prevent this, we sometimes do

$$1 + \log\left(\frac{1+N}{1+n_i}\right)$$

(sk-learn implementation)

→ TF :- How frequently word occurs in a document.

IDF :- How rare is the word in the whole corpus. If word is more frequent in the document corpus, it is less important.

TF-IDF :- Importance of the word in the whole corpus.

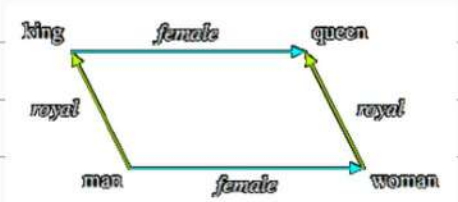
Word 2 Vec :- State of the art word to vector conversion that takes semantic meaning into consideration.

→ The dimensions are usually really high (like 300 ...)

→ It tries to achieve the following :-

① Semantically similar words have closer vectors

② It also tries to keep relationships



$(V_{man} - V_{woman}) \parallel (V_{King} - V_{queen})$

→ Even for country capitals etc,

→ It does this automatically without being programmed to do so. Came out in 2013.

→ (very large data corpus) → (word2vec) → word : vectors
⋮

The more the dimension of vector, the more information rich it is.

→ At it's core, word2vec looks at the neighborhood of words and looks for words with similar neighborhood.

$w_1\ w_2\ \underline{w_3}\ w_4\ w_5\ w_6$

neighborhood → $N(w_3)$ ⟹ if $N(w_i) \approx N(w_j)$
then $v_i \approx v_j$

→ Google trained word2vec on it's Google News dataset & this word2vec gives 300 dimension vectors.

Avg word2vec, tfidf weighted word2vec :-

word ⟶ word2vec ⟶ vectors.

→ For sentence, sentence2vec exists but there are other methods like average word2vec.

Average word2vec :- $r_1 :- w_1\ w_3\ w_4\ w_2\ w_1\ w_5$

↳ $v_1 :- \frac{1}{n_1}\left[w2v(w_1) + w2v(w_3) + w2v(w_4) + w2v(w_2) + w2v(w_1) + w2v(w_5)\right]$

n_1 = number of words in r_1.

→ Not perfect but simplest method.

tfidf w2v :- first tfidf vector of all words of sentence is calculated.

then each word's w2v is multiplied with it's tfidf score. and whole thing is added & divided by sum of tfidf of all words.

$$tfidf\text{-}w2v(r_1) = \frac{\left[t_1 * w2v(w_1) + t_2 * w2v(w_2) + t_3 * w2v(w_3) + \cdots + t_n * w2v(w_n)\right]}{(t_1 + t_2 + t_3 + \cdots + t_n)}$$

t_n → tfidf of word w_n

$$= \frac{\sum_{i=1}^{n} (t_i * w2v(w_i))}{\sum_{i=1}^{n} t_i}$$

if tfidf of all words is 1, then their avg will be 1, then tfidf w2v becomes avg w2v

→ tfidf w2v & avg w2v are weighted schemes and are simple models. They are not always perfect.

→ thought vectors is a technique that is bit advanced. (Seen in deep learning chapter)

→ Each row of sparse matrix is called sparse vector.

→ If an $n \times m$ matrix has only 'K' non zero elements, sparsity of the matrix $= \frac{(n*m) - K}{(n*m)}$

density of matrix = 1 − sparsity

vallakatla Charith — 9 Votes

How to avoid removing stop word 'not' from text ?

Reply — May 04, 2018 11:19 AM

AppliedAICourse

you can remove the 'not' from stopwords set, syntax: set.remove(element) [stopwords.remove('not')] will do the work

Reply — May 04, 2018 11:55 AM

Pavan Kalyan Reddy Ramireddy — 5 Votes

Can you explain the parameters of CountVectorizer such as 'preprocessor ' and 'tokenizer '

Reply — Apr 23, 2019 11:22 AM

Applied AI Tech Admin

So far you have seen preprocessing like converting the uppercase words into lower case, stopword removal, removal of punctuations, etc.

preprocess:

When you apply CountVectorizer, it splits the text into features directly into the words on the basis of spaces. So before in case if we need to perform any addition preprocessing on the text before splitting the text into the features we have to define those additional preprocessing steps in the form of a function(say f1) and pass this function to the parameter **preprocess**(ie., preprocess=f1)

tokenizer:

If we want to go for a customized way of tokenizing the text(ie., rather than splitting the text into words on the basis of white space, if we want to split the text in a different and a customized way), then define those new tokenizing steps in the form of a function(say f2) and pass the function name as a value to the parameter **tokenizer**. (ie., tokenizer=f2)

Reply — Apr 23, 2019 13:41 PM

→ min_df = 0.1 → ignore words that appear in less than 10% of documents

= 10 → ignore words that appear in less than 10 documents.

max_df :- Ignore words that appear too frequently. AKA corpus specific stop words.

→ In W2V (Google), words & their stems might not be similar

Vijay Maurya — 27 Votes

Hello Team I have downloaded google model bu this method

1. For download

!wget -c "https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz"

2. For extraction

!7z e GoogleNews-vectors-negative300.bin.gz

3. Google model

from gensim.models import KeyedVectors

w2v_model_google=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)

Don't need download anything run it in Google Colab

→ To load big datasets in collab w/o downloading.

How Classification Works :-

Amazon Dataset

$r_i \longrightarrow$ Text $\longrightarrow$ Vector

$\longrightarrow$ +ve/-ve

$\rightarrow r_i \longrightarrow (v_i, +ve/-ve)$

{Given a review Text, determine/predict if review is +ve/-ve

$\rightarrow y = f(x)$

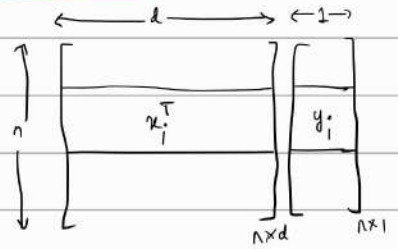
↳ In most ML problems, we are trying to find this f() function.

Amazon Dataset :-

training dataset $\longrightarrow$ $\boxed{f}$ (Algorithm) (training stage)

(testing dataset) $x_q \longrightarrow \boxed{f} \longrightarrow y_q$ (testing/evaluation stage)

Data Matrix Notation :-



$r_i \longrightarrow$ Text $\rightarrow$ vector

↳ class (+ve/-ve)

$y = f(x)$

For Amazon Dataset, $D = \{(x_i, y_i)_{i=1}^{n} \mid x_i \in \mathbb{R}^d, y_i \in \{0,1\}\}$

$r_i \longrightarrow$ vector x_i

$y = f(x)$

→ If target classes = 2, (like Amazon Dataset), classification is called Binary classification

= n, " " " Multi Class Classification. (IRIS dataset, MNIST dataset)

If y_i is not finite, it becomes regression. $y_i \in \mathbb{R}$ not a set of finite values.

→ classification → Discrete values, Regression → continuous variables

→ Algorithms like Decision trees, random forests can perform classification as well as regression

K-Nearest Neighbours :-

Assume $D = \{(x_i, y_i) \mid x_i \in \mathbb{R}^2 \; y_i \in \{0,1\}\}$

× → positive (1)

× → Negative (0)

Geometrically all points close to x_q are positive.

We can conclude x_q is positive

Steps in KNN :-

① Find 'k' nearest points. Let k=3 in this example $= \{x_1, x_2, x_3\}$

$\{y_1, y_2, y_3\}$ → In this case, majority = positive

② Majority vote b/w $\{y_1, y_2, y_3 \cdots, y_k\}$

(Avoid taking even 'k', as there could be a chance of tie)

→ In scikit learn, there are 3 methods to calculate nearest neighbours

(i) Brutforce :- Effective in small datasets.

(ii) KD tree :- distance information for the sample. The basic idea is that if point A is very distant from point B, and point B is very close to point C, then we know that points A and C are very distant, *without having to explicitly calculate their distance*. In this way, the

(iii) Ball tree :- A ball tree recursively divides the data into nodes defined by a centroid C and radius r, such that each point in the node lies within the hyper-sphere defined by r and C. The number of candidate points for a neighbor search is reduced through use of the *triangle inequality*:

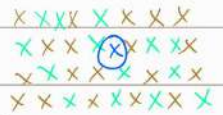
$$|x+y| \leq |x| + |y|$$

→ KNN is not used in realtime applications as it is a lazy algorithm.

Failure Cases of KNN :-

→ when point is very far away from rest of the points. Taking the nearest neighbours doesnt make much sense.

→ Very jumbled randomly spread data.



No useful information in this type of spread.

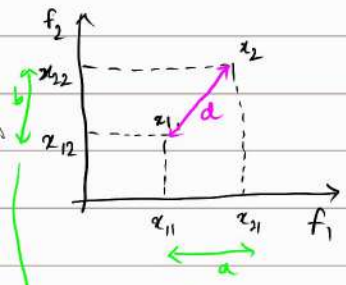
Most ML algos fail when data is like this.

→ Determining threshold.

Note:-Determining Threshold is Data-Specific

1. Take Train Data
2. Take each point in data-set and compute its distance from its first neighbor and store all these distance
3. Sort them, some are small and some are large
4. Take 99.99% percentile as your threshold, let's name it D-99
5. Now for our Query point, if its distance from its first neighbor is greater than D-99 then take it as an outlier and don't classify it

Distance Measures :-



$$x_1 = (\overset{f_1}{x_{11}}, \overset{f_2}{x_{12}})$$
$$x_2 = (x_{21}, x_{22})$$

d = len of shortest line from x_1 to x_2

$$d = \sqrt{(x_{21}-x_{11})^2 + (x_{22}-x_{12})^2} = \|x_1 - x_2\| \longrightarrow \text{Euclidean distance.}$$

if $x_1 \in \mathbb{R}^d$ & $x_2 \in \mathbb{R}^d$, $\|x_1 - x_2\|_2 = \left(\sum_{i=1}^{d} (x_{1i} - x_{2i})^2\right)^{1/2}$

L2 Norm of a vector

→ if only one vector ie, $\|x_1\|_2 \longrightarrow$ distance of x_1 from origin $= \left(\sum_{i=1}^{d} x_{1i}^2\right)^{1/2}$

(a+b)

Manhattan Distance :- $\sum_{i=1}^{d} |x_{1i} - x_{2i}|$.

Also written as $\|x_1 - x_2\|_1$. L1 Norm. $\|x_1\|_1 = \sum_{i=1}^{d} |x_i|$

Lp Norms :- corresponding value is called Minkowski Distance

$$\|x_1 - x_2\|_p = \left(\sum_{i=1}^{d} |x_{1i} - x_{2i}|^p\right)^{1/p}$$

If $p=2$, Minkowski distance = Euclidean Distance

If $p=1$, Minkowski distance = Manhattan Distance.

$$\|x_1\|_p = \left(\sum_{i=1}^{d} |x_{1i}|^p\right)^{1/p}$$

→ Distances are b/w two points. Norms are b/w two vectors.

ex:- Euclidean distance (x_1, x_2) = L2 Norm of $(x_1 - x_2)$ = $\|x_1 - x_2\|_2$

Hamming Distance :-

Used for Boolean Values (ex:- Binary Bag of Words)

$x_1 = [0,1,1,0,1,0,1,1,1]$

$x_2 = [1,0,1,1,0,1,0,0,1]$

Hamming Dist (x_1, x_2) = # of locations where Binary Vectors differ.

For above example :- HD = 7

→ Can also be used for strings - Number of locations where strings differ

→ Can also be used for Gene Sequencing. They are basically sequence of letters

→ Sklearn kNN uses Minkowski by default.

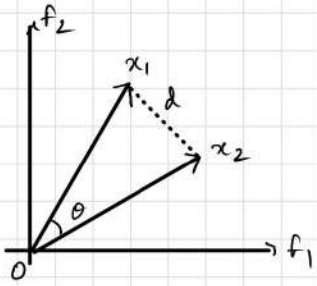
Cosine Similarity and Cosine Distance :-

→ As similarity increases, distance decreases. Similarity $\propto \frac{1}{\text{distance}}$

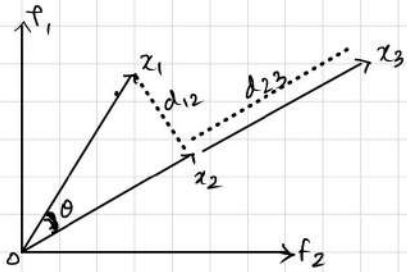
→ $\underbrace{1 - \text{cos-similarity}(x_1, x_2)}_{\text{this lies b/w } [-1, 1]} = \text{cos-dist}(x_1, x_2)$

If they are very similar, cos-sim = +1
if they are very dissimilar, cos-sim = -1

→



d = Euclidean distance
cos-similarity = $\cos(\theta)$



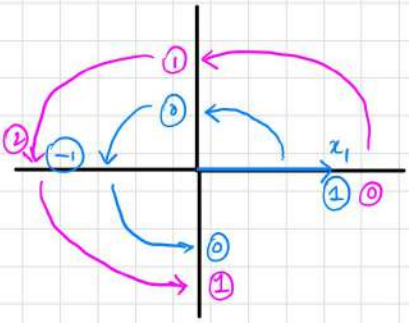
cos-sim $(x_1, x_2) = \cos(\theta)$

cos-sim $(x_2, x_3) = 1$ $(\cos(\theta) = 1)$

cos-dist $(x_2, x_3) = 1 - \text{cos-sim}(x_2, x_3)$
$= 1 - 0 = 1$

But $d_{12} < d_{23}$ (Euclidean)

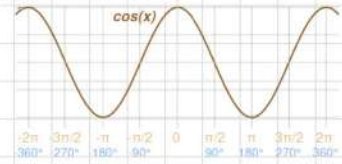
→



green = cos-dist
orange = cos-sim

angle b/w (x_1, x_2)		cos-dist
$0° - 90°$	→	0 → 1
$90° - 180°$	→	1 → 2
$180° - 270°$	→	2 → 1
$270° - 360°$	→	1 → 0

cosine - graph :-



→ We know that $\cos(\theta) = \frac{x_1 \cdot x_2}{\|x_1\| \, \|x_2\|}$

↳ if no L_p is mentioned, it is 2 by default.

if x_1, x_2 are unit vectors, $\cos(\theta) = x_1 \cdot x_2$

→ Relationship b/w euc-dist & cos-sim :-

if x_1 & x_2 are unit vectors, $[\text{Euc-dist}(x_1, x_2)]^2 = 2(1 - \cos(\theta)) = 2(\text{cos-dist})$

→ Whenever we find cosine distance, we need to perform row normalization → Converting them to unit vectors basically.

Measuring effectiveness of KNN :- (2.8)

$$D = \begin{array}{c} 1 \\ 2 \\ 3 \\ \vdots \\ n \end{array} \left[\begin{array}{c} \longleftarrow x_i \longrightarrow \end{array} \right] \left[\begin{array}{c} y_i \end{array} \right] \qquad y_i \in \{0, 1\}$$

Text → BoW / TFIDF

Problem :- Given a new text, what is its polarity (+ve/-ve)

$(x_q \xrightarrow{\text{text}}) \longrightarrow (y_q)$ Measure = KNN + majority neighbour vote

Approach :-

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ \vdots \\ n \end{array} \begin{array}{c} x_i \\ \left[\begin{array}{c} \\ \\ \\ \\ \\ \end{array} \right] \end{array} \begin{array}{c} y_i \\ \left[\begin{array}{c} \\ \\ \\ \\ \\ \end{array} \right] \end{array} \begin{array}{l} \left.\begin{array}{c} \\ \\ \\ \end{array}\right\} D_{train} \\ \left.\begin{array}{c} \\ \\ \end{array}\right\} D_{test} \end{array}$$

$\forall x_i \in D_n$, $x_i \in D_{train} \mid x_i \in D_{test}$

$D_{train} \cap D_{test} = \phi \mid D_{train} \cup D_{test} = D_n$

How to split?

Simple way is to do it randomly. Not the only way.

$n_1 \approx 0.70\%$ $n_2 \approx 30\%$.

(D_{train}) → train (D_{test}) → test.

(i) Train KNN using D_{train}

(ii) For each point in D_{test}, x_q = point.

Use D_{train} & KNN to predict the location & polarity (y_q)

(iii) Compare $y_q == y_{point}$.

count ++

At the end, count = # of points for which D_{train} + KNN gave correct answer.

$$\text{Accuracy} = \frac{\text{count}}{n_2}$$

n_2 → # of points in D_{test}.

$0 \le \text{Accuracy} \le 1$

ex :- if accuracy = 0.91, ⟹ 91% times KNN predicted correctly.

→ KNN is said to be a lazy learner because there is no training phase. In KNN the "training phase" is basically loading all datapoints into RAM

→ random state parameter in sklearn prevents from getting different train test datasets each time (thereby getting different accuracies)

Time & Space Complexity (KNN) :- (2.9)

$$\underbrace{x_q \xrightarrow{KNN} y_q}_{\text{time ?}}$$

Input :- D_{train}, K, $x_q \in \mathbb{R}^d$

Output :- y_q

K is generally small (5 - 10)
d is large (thousands)

Pseudo code :-

```
KNNpoints = [ ]
O(n) ← for each x_i in D_train :
O(d) ←     compute distance (x_i, x_q). Store d_i s.
O(K) ←     keep smallest K-distances ; (x_i, y_i, d_i) → Add this tuple to KNNpts
O(1)
```

↓

O(nd) to find KNN

```
cnt_pos = 0,  cnt_neg = 0
O(K) ← for each x_i in KNNpts :
O(1)        if y_i is +ve :
                cnt_pos += 1
            else :
                cnt_neg += 1
O(1) ← if cnt_pos > cnt_neg :
            return y_q = 1
        else :
            return y_q = 0 .
```

Time complexity $= O(nd) + O(1) + O(1)$

$= O(nd)$

if d is small ie., $d << n$

$= O(n)$

↓

for single query point.

All data is stored in RAM

In ML we also include space used by RAM contrary to algos in general where input space is not considered.

Space Complexity $= O(nd)$

→ In all other algorithms D_{train} computes the equations (components) of hyperplane seperating all points. This hyperplane is used to classify components of D_{test}. KNN doesnt have a concept of hyperplane. So it requires D_{train} during test as well.

→ min heap can be used to extract smallest point in O(1) time.

KNN Limitations :- (2.10)

→ Amazon fine food reviews has 364K reviews & 100K dimensions.

→ time & space comp of KNN = O(nd)

→ space = 364K * 100K ≈ 36 GB ram usage.

→ time ≈ 36 billion computations. Internet companies need answers in milliseconds

Such systems are called low latency applications.

Not all systems need to be low latency. ex:- medical

→ O(nd) is bad for low latency applications

→ There are other tech such as Kd-tree & LSH (Locality Sensing Hashing) that reduce TC & SC of KNN

(Kevin Zakka KNN blog - Summary)

→ KNN is used as powerful benchmarking tool for other classifiers. KNN can outperform other powerful classifiers. Used in economic forecasting, data compression & genetics.

→ KNN is non parametric and Instance based.

non parametric → Makes no assumptions about data or 'f'.

Instance based → It doesn't explicitly learn a model. It memorizes the training instances and used as "Knowledge" during testing.

→ Using test set for hyper parameter tuning can lead to overfitting. So we ~~k-f~~ can use k-fold cross validation to tune hyper parameters in KNN

K-fold cross validation :- We create a subset in training set called validation set. In K fold, the training set is divided into k folds & each time it is validated with one fold & the remaining folds to train data.

We can use cross validation to get the best 'k' value.

For each k value, kfold CV is applied & the CV score (average of all ~~test~~ kfold tests) is added to a list). This list is used to get the best CV.

scikit learn :- cross_val_score(knn, x_train, y_train, cv = 10, scoring = 'accuracy')

→ classification problem.

Pros of KNN :-

→ works better with multi class algos.

(4) Cons of KNN :-

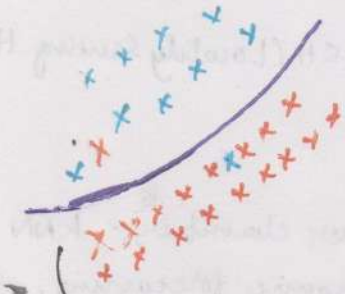
→ computationally Expensive.

→ suffers from skewed class distributions.

Decision Surface for KNN as K Changes :- (2.11)

→ How to pick K ? 'K' is a hyper parameter.

→ curves that seperate positive points (class) from -ve are called decision surfaces.



smoothcurve

~~(Linear~~

(2 errors)

jaggedcurve.

(no errors)

curves in 3D = surfaces.

curves in nD = hyper surface.

Since these curves decide the class, it's called decision surface.

→ Decision tree is drawn by expecting the output class when 'K' in KNN = 1

As K increases, smoothness of curve increases.

→ when K = n, output will be the dominant class. Does not depend on location of point.

→ Decision surface is useful to see visually.

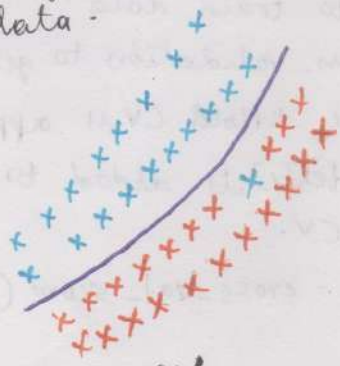
Overfitting and Underfitting (2.12)

→ finding the right function/algorithm 'f' to get $f(x_q) = y_q$ is called fitting a function to the data.

Overfitting

there's high chance that this is just noise.

'try hard'



Well fit/

Appropriate fit

ignores noisy points.

Robust.

K = n

⇓

Predicted class = majority class

Underfitting

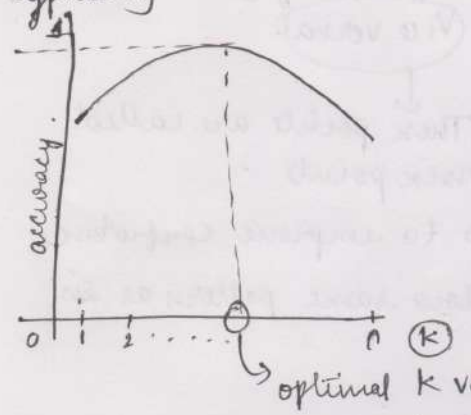
Doesnt even try to create a hyperplane

Underworking.

If K is too small → overfit
if K is too large → underfit.

Cross validation (2.12) :-

typically



→ ML algorithm needs to work well on future unseen points.
When an algo performs well on future unseen points & not just on D_{test}, it is called a generalized algorithm.
Most imp property of ML algorithm

$D_n \rightarrow D_{train}$, $D_n \rightarrow D_{cv}$ } → to obtain 'f'. D_{test} is not used.
$D_n \rightarrow D_{test}$ → unseen data.

The accuracy obtained on D_{test} can be said about the algorithm in general.

60-20-20 , 70-10-20 general ratios

Generalization Accuracy. (Generalization error is the opposite to this)

K-Fold cross validation :- (2.13)

→ In simple cross validation, we are losing a big part of the dataset.
→ In ML, more the training data, better the model.
→ Explanation in page 3
→ Since D_{test} is "unseen", we can generalize the accuracy.
→ Rule of thumb is 10 fold cross validation.
→ Time taken to compute best K increases by K' times if K-F CV is used.
we cannot say $O(Knd)$ because K is a small integer.

Visualizing train, CV & test datasets (2.14):-

→ D_{train} & D_{CV} do not overlap perfectly.

→ If there are many +ve points from D_{train} in a region, highly likely to find many +ve points from D_{CV} in the region. Vice versa.

These points are called noise points.

→ D_{train}, D_{CV} & D_{test} are not picked in same order to improve confidence.

→ Basic assumption in ML is future data will follow same pattern as in train data.

Determining Overfitting and Underfitting :- (2.15):

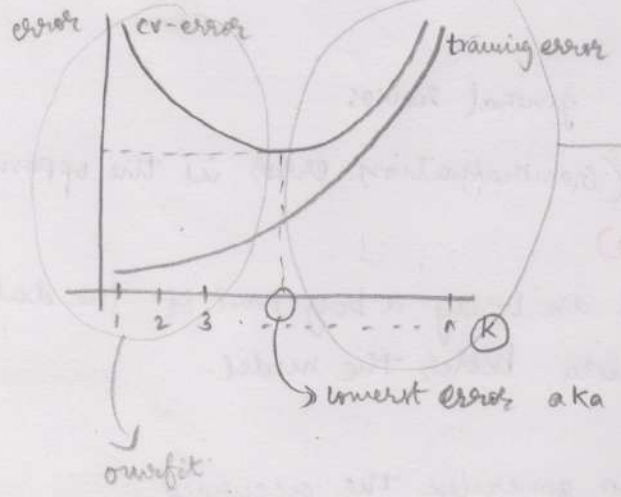
Plots can be used to determine if we are overfitting/underfitting.

$$\text{accuracy} = \frac{\#\text{ correctly classified points}}{\text{total }\#\text{ points}} \qquad \text{error} = 1 - \text{accuracy}.$$

training error = error obtained by testing model (f) on training data

validation error = error obtained by testing model on CV data

test error is only used to report final accuracy of model.



if both training error & CV error are high, then we are underfitting.

if train error is small & CV error is high then we are underfitting.

→ This plot is not really necessary for KNN, but beneficial for other neural net models.

→ Even if model is tested on training data there will still be some error.

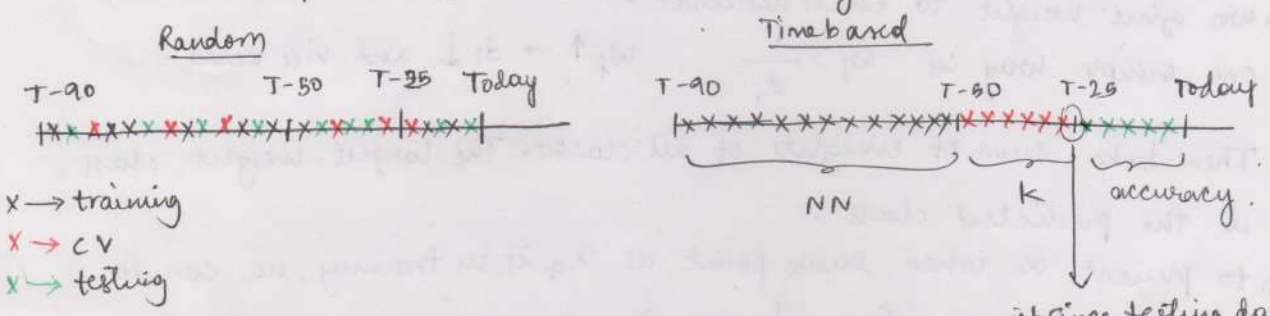
Time Based Splitting (2.17):-

→ We first sort data in increasing order of time. If timestamp is not available, time based splitting is not possible.

→ Then take first $x\%$ as training, next $y\%$ as trds validation and next $z\%$ as testing data (most recent data)

$$x + y + z = 100.$$

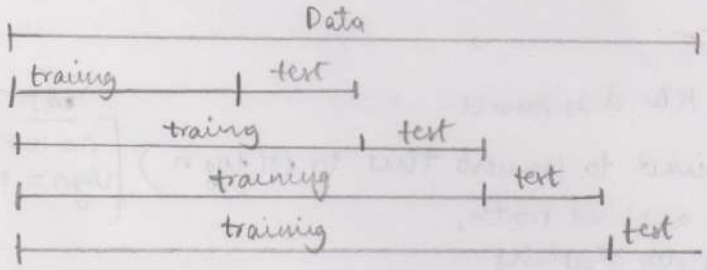
→ with time product & their reviews change.



~~if~~ Since testing data is unseen, this means, model's prediction's accuracy carries to the future data as well.

→ whenever time feature is available & things/behaviours/data changes over time, timebased splitting is preferable.

→ rolling based k fold cross validation can be used.



KNN for regression (2.18):-

$$D = \{(x_i, y_i)_{i=1}^{n} \mid x_i \in \mathbb{R}^d, y_i \in \mathbb{R}^d\}$$ → only difference.

Instead of majority vote of y_i of nearest neighbours (like in classification) we do mean / median.

Lot of classification techniques can be extended to regression.

② Weighted KNN (2.19) :-

→ The distances to the nearest neighbours are taken into consideration.

blue points are closer to red than green but greenies majority.

→ we give weight to each distance.

One simple way is $w_i = \frac{1}{d_i}$. $w_i \uparrow \rightarrow d_i \downarrow$ and vice versa.

Then take sum of weights of all classes. The largest weight class is the predicted class.

→ to prevent ∞ when same point as x_q is in training, we can do

$$w_i = \min\left(10, \frac{1}{d(x_q, x_i)}\right).$$ ie, max weight possible = 10.

→ KNN is highly affected by outliers. So it is suggested to handle them before KNN.

Voronoi Diagram (2.20) :-

Each color has one point & any points nearest neighbour is the point in cell. (1NN)

Usage in science, art etc.,

Binary Search tree (2.21) :-

TC of KNN $= O(n)$ if K & d is small.

Kd tree can be used to convert this to $O(\log n)$ $\left[\begin{array}{l} n = 1024 \\ \log n = 10 \end{array}\right]$ ex:-

↳ used in CS, applied math, computer graphics.

Binary Search Tree :- We need a sorted array.

TC to search for an element $= O(\log n)$

↓ also the depth of the tree when number of elements $= n$

Kd-tree :- (2.22) :

→ Similar to BST but can operate on multiple dimensional data

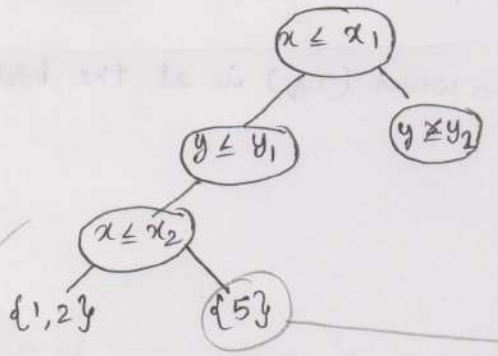
Steps :-

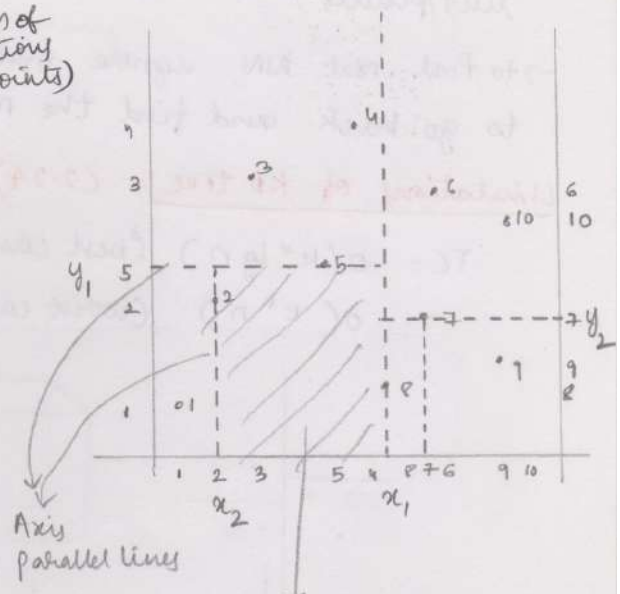
① Project points onto x axis. Compute median of projection (not points)

② Draw Axis parallel lines.

③ Repeat on y axis.

④ Keep repeating.





$\boxed{x \le x_1,\ x > x_2,\ y \le y_1}$

→ We are basically breaking down space into rectangles/cuboids/hyper cuboids using axis parallel lines/planes/hyperplans etc.,

→ Main point is we go through each axis one after another till we reach leaf nodes.

→ Doesnt necessarily have to be split in the same order of axes. But the first split order should be carried throughout the process.

Finding nearest neighbours using Kd tree (2.23) :-

→ We use this tree to find the potential closest point.

→ if dist b/w that point & x_q is d, we draw a circle centered at x_q with radius 'd'.

If any other point is within that circle, then the new point is the closest neighbour. Else original one is. This is visual method. Geometrically, we check if the circle touches any other parallel line/plane. Then we backtrack in ~~that~~ the tree to that plane rule & look for leaf again.

→ In best case, (no other points in circle) number of comparisions = depth of tree, $= O(\lg(n))$

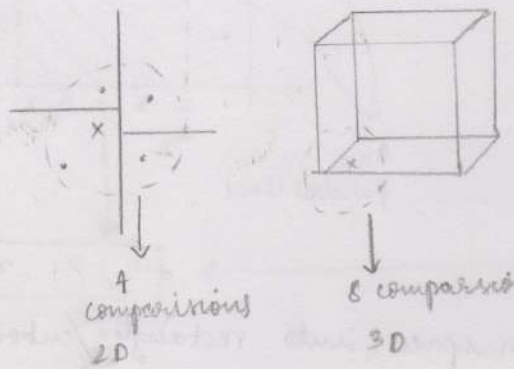
~~worst~~ case $= O(n)$. The number of comparisions increase as the number of intersections with other hyperplanes increase. In worst case it intersects with almost all other axis parallel planes.

-> We check if there's intersection with any axis parallel lines/planes by measuring dist b/w centre of circle/hypersphere to the axis parallel lines/planes.

-> to find next NN, ignore first NN & find next NN. We use backtracking to go back and find the next NN.

Limitations of KD tree :- (2.24)

$TC = O(k * \lg n)$ (best case)
$O(k * n)$ (worst case) } if d is small.



→ when (x_q) is at the borders.

→ 2^d comparisions in n dimensions

if $d = 10 \rightarrow 2^d = 1024$
$d = 20 \rightarrow \approx 1$ million

As dimensionality increases, worst case comparisions increase exponentially.

when d is not small {2, 3, 4, 5 etc.} $TC = O(2^d \log n)$

when $2^d \approx n$, $TC = O(n \log n)$ → worse than brute force approach.

∴ KD tree is better when d is very small.

Even when d is small, TC is $O(\lg n)$ when data is uniformly distributed. IRL data is never uniformly distributed.

As data becomes less & less uniformly distributed, TC becomes closer to $O(n)$

KD tree is not designed for ML. Designed for computer graphics applications where it's mostly 2D & 3D.

→ We can use dimensionality reduction but model looses complete model interpretability.

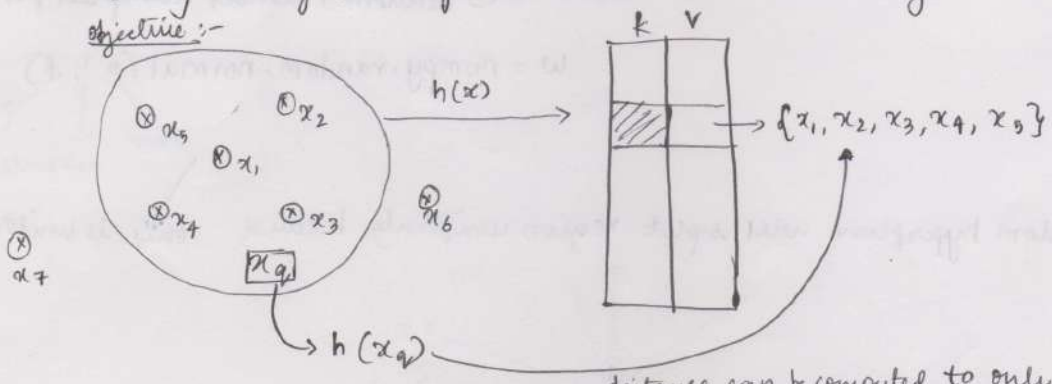
Extension to Kd tree :- (2.25)

Ball tree :- Uses balls instead of hyperplane to seperate.
Quad tree :- Splits into two dimensions simultaneously. So each node has 4 children
⋮
→ used in scikit learn.

Hashing vs LSH :- (2.26)

Locality Sensing Hashing Even works when d is large

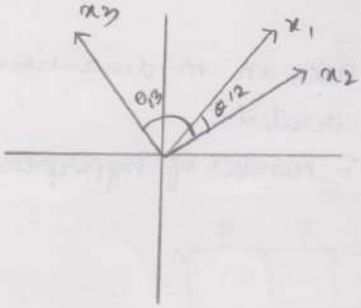
objective :-

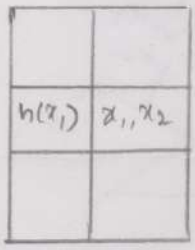


distance can be computed to only these points.

Locality Sensitive Hashing for cosine similarity (2.27)

cosine similarity → as angle increases, cosine similarity decreases.
All about angular similarity. Not Euclidian.

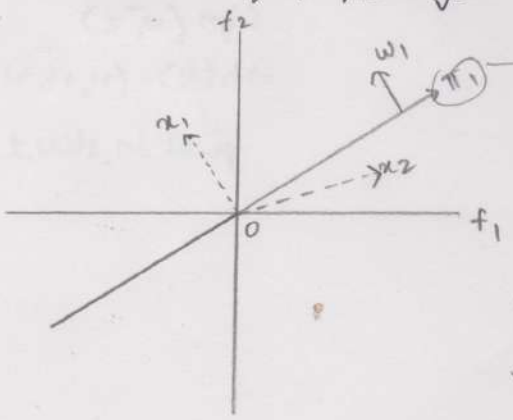




$h(x_1) = h(x_2) =$ same value.

$\theta_{12} \approx 0$ (small) $\Rightarrow x_1, x_2$ are similarly close.

→ LSH is called a randomized algorithm. (Not always the right answer but correct answer with high probability)



→ random hyperplane. w_1 = unit normal to π_1

$w_1^T x_1 \geq 0$ (since both are going towards same direction)

$w_1^T x_2 \leq 0$ (both are going towards opposite direction)

→ core idea behind LSH.

(12) how to generate hyperplane?

→ eqn of plane passing through origin → $w^T x = 0$

w = normal to the plane.

$w \to d$ dimensional vector

0	1	2	3	. . . .	d-1

↳ finding this = getting random hyperplane.

is generated such that

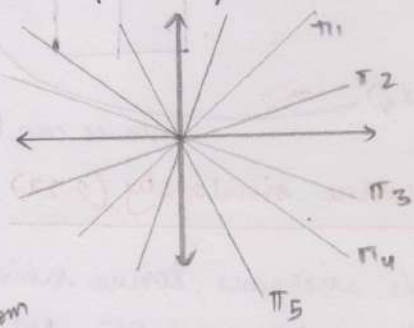
0	1	...	i	i+1	...	d-1	d

↳ random number sampled from $N(0,1)$

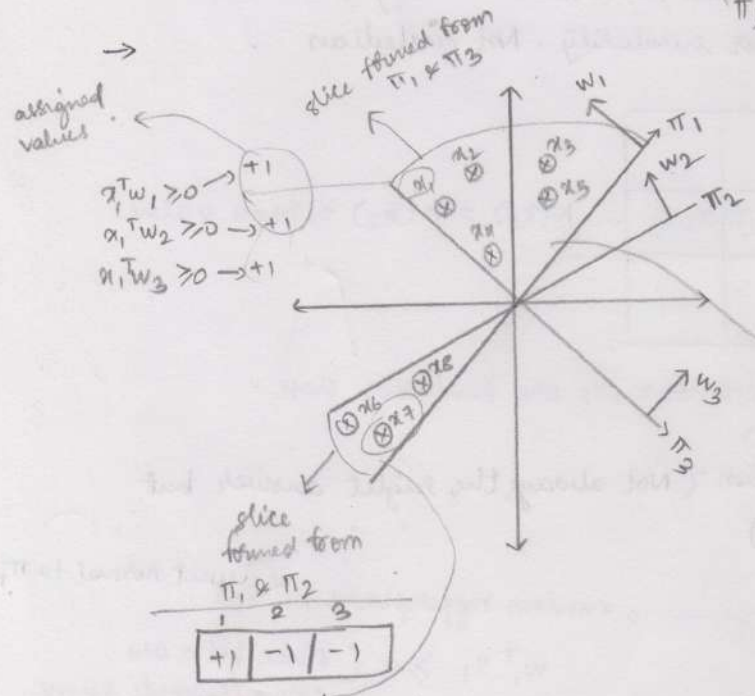
w = numpy.random.normal(0, 1, d)

mean, variance, number of points.

→ random hyperplane will split region uniformly because this is uniform distrib



→



$\Rightarrow h(x_7) = (+1, -1, -1)$

∀ slice an m dimensional vector is created.

m = number of hyperplanes.

1	2	3
+1	+1	+1

↳ when all values in slice are multiplied with π_1, π_2 & π_3, the result is stored.

sign $(w_1^T x)$

$\Rightarrow h(x_1) = (+1, +1, +1)$

point in slice 1

Computing hash table:-

$x \longrightarrow h(x)$ (vector of size 'm')

this is used as key in the hash table.
All elements in the slice = values

ex:-

(+1, +1, +1)	x_1, x_2, x_3, x_4, x_5

Space $\approx O(nd)$
comp

time complexity $\approx O(md\,n)$

each point 'md' multiplications to get sign.

→ given a query point x_q.

step ①:- compute $h(x_q)$ =

1	2	. . . .

sign $(w_1^T x_q)$ → sign $(w_2^T x_q)$ . . .

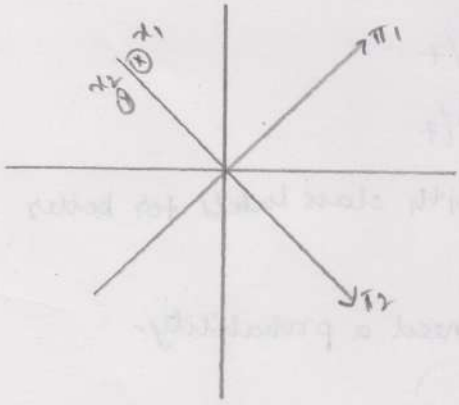
step ②:- $h(x_q)$ key. returns all points in slice.
Then cosine sim b/w x_q & all points in slice to get NN

Given a hash table,

TC ⟶ $O(md)$ to get $h(x_q)$
if there are n' elements in slice, $O(n'd)$ comps

$TC = O(md + n'd)$
if n' is small / $n' \approx m$, $TC = O(md)$

Typically, $m \approx \lg(n) \Rightarrow TC = O(d * \lg n)$



x_1, x_2 are close but different slices.
→ LSH could miss NN.
→ To get around this

$h_1(x)$ → | | | | . . . | | → m hyperplans

$h_2(x)$ → | | | | . . . | | → m hyperplanes.

→ different

⋮

l such hash tables.

In one of those instancy, we are bound to get correct NN

→ We calculate union of all points obtained from LSH & calculate distances.

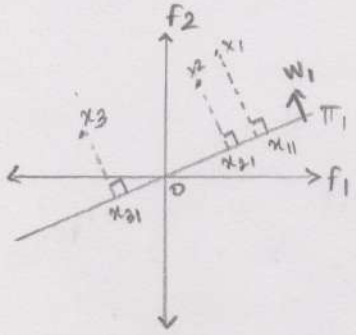
TC to query gives L hashtables

$O(mdL)$. Typically L is small

As number of hyperplanes increases, # of slices increases, # of points per slice ↓

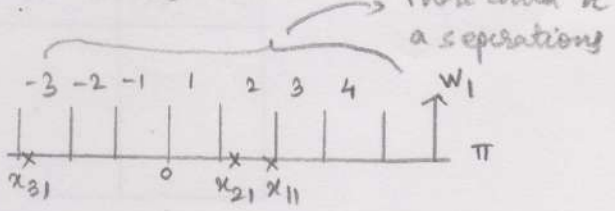
LSH for Euclidean Distance :- (2.28)

→ simple extension to cosine similarity.



just the hyperplane →

$d(x_1, x_2) << d(x_1, x_3)$

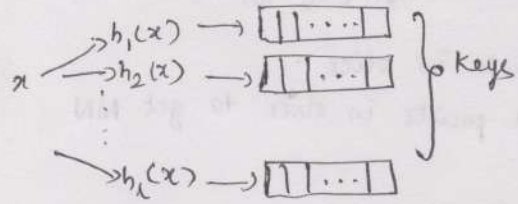


$h(x_{11}) =$ [2 | . . . . . |] → m → # of hyperplane

$h(x_{31}) =$ [-3 | . . . . . |] m

cosine sim LSH → hash values are either +1 / -1

Euclidean sim LSH → hash values are any number b/w ~~-∞~~ 1 & a



2 points have the same key ⇒ they tend to be closer.

→ LHS is not perfect. It is probabilistic. Used extensively with computer vision.

Probabilistic class Label :- (2.29)

7NN → x_q : 4 -ve points 3 +ve points → y_q = -ve (less certain)

→ x_q' : 7 -ve points 0 +ve points → y_q' = -ve (more certain)

Quantifying this uncertainity

$P(y_q = -ve) = 4/7$ $\quad P(y_q = +ve) = 3/7$

$P(y_q' = -ve) = 7/7$ $\quad P(y_q' = +ve) = 0/7$

→ probabilities are to be returned along with class labels for better understanding.

→ Useful in medical applications where we need a probability.

→ Can be combined with weighted KNN as

$$P(x_q = +ve) = \frac{w_1}{w_1 + w_2 + w_3} \qquad P(x_q = -ve) = \frac{w_2 + w_3}{w_1 + w_2 + w_3}$$

$$w_1 = +ve = \frac{1}{d_1} \qquad w_2, w_3 = -ve = \frac{1}{d_2}, \frac{1}{d_3}$$

KNN Interview Questions :- (3.1)

→ Why do we use Euclideans instead of manhattan?
A:- Euclidean gives the shortest path. Manhattan doesn't.

→ Which of the following can be used for imputing missing values for both categorical & continuous variables.
(a) KNN ✓ (b) Linear Regression (c) Logistic Regression.
A: KNN because it makes predictions based on points in the neighbourhood while others work based on hyperplane seperating classes.

→ KNN for regression: Prediction based on mean/median of K most similar instances.

→ To reduce overfitting due to curse of dimensionality, (KNN) we can use dimensionality reduction (or) feature selection.

→ We can implement 2NN classifier by ensembling the 1NN classifier.
(using it multiple times /ensembling = using multiple algorithms)

→ The boundary becomes smoother by increasing the value of K.

→ The Euclidian distance treats each feature as equally important

→ if n is very large, TC = O(nd)

→ Training time for any value of K in KNN is true.

Classification Algorithms in various situations (4.1):

→ Knowing # of techniques is not important. Knowing scenarios & real world cases is important.

→ Scikit learn can't be used in large scale datasets. We use big data technologies for that.

Imbalanced v/s Balanced dataset :- (4.2) :-

$D_n \rightarrow n_1 \rightarrow +ve$
$\rightarrow n_2 \rightarrow -ve$

$n_1 \approx n_2 \rightarrow$ Balanced dataset

$n_1 << n_2 \,/\, n_2 >> n_1 \rightarrow$ Imbalanced dataset.

$n_1 <<<< n_2 \,|\, n_2 >>>> n_1 \rightarrow$ Severly/Heavily imbalanced dataset.

n_2 → majority/dominating class. Sometimes because they are majority class there might be some bias.

(16)

How to work around imbalanced dataset :-

① Undersampling :- randomly sample less # of dominating class & keep the class with less samples & create new sample. New sample is balanced.

Problem with undersampling :- Model will have less amount of data.

$$|D_n'| << |D_n|$$

so throwing away data should always be avoided.

② Oversampling :- Repeat the minority class to get a balanced sample

Some variations to oversampling :-

(i) ~~Ex~~ Extrapolation :- Creating artificial points that belong to the minority class.

class weight :- give proportionally more weight to minority class.

→ one minority class point counted as multiple points

Not implemented for KNN in sikit-learn.

→ With imbalanced dataset, we can get very high accuracy with dumb model.

→ Imbalanced datasets (skewed datasets) are very common in e-commerce & medical field.

→ Oversampling must be preferred over undersampling.

→ Multiclass classification, the algorithm internally uses one v/s rest approach that internally creates binary classifier for each class. If we are balancing class weights, then we can use class_weight = "balanced" parameter that internally balances the weights for each binary classifier.

→ one v/s all (OvA) :- red, blue, gree → red v/s [blue, green]
→ green v/s [red, blue]
→ blue v/s [red, green]

→ downside is, one model for each class.

→ It's a heuristic approach (Practical. Not perfect)

Multi Class Classification (4.3):-

Multiclass classifiers :- $y_i \in \{\underbrace{\ldots\ldots}_{>2 \text{ classes}}\}$

→ Some algos do this automatically some dont

ex:- KNN can be extended to multiclass.
Logistic Regression is fundamentally a binary classifier. Cannot do multi class classification as easily.

→ Can we convert multi class classifier to binary classifier? If we can, we can use models like logistic regression to solve them.

ex:- 1 v/s rest :-

Multiclass problem → "c" classes $\{1, 2, 3, \ldots, c\}$

→ $f_1(x) \rightarrow 1$ (or) not 1
→ $f_2(x) \rightarrow 2$ (or) not 2
⋮
→ $f_c(x) \rightarrow c$ (or) not c

TC will increase since there are c models being trained.

→ If new query point is encountered (during production phase) ~~test example~~ it belongs to class 'k' if it has the highest probability among all classes. If two classes have same highest probability then we choose the one with more points in dataset.

→ we can apply multinomial logistic regression without one v/s rest for multiclass classification.

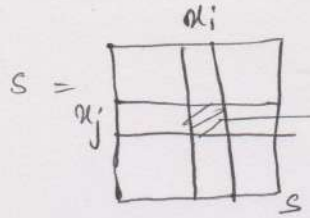
→ During 1 v/s rest, we also need to balance dataset while looking for class 'x' v/s all other classes. Then build classifier & make predictions & compute performance metrics.

KNN given a distance measure (4.4):-

→ Sometimes, $[x_i \in \mathbb{R}^d]$ problem, writing x_i in vector form might not be easy. ex:- Chemical compounds. But similarity can still be measured.

ex:- domain experts like pharmacists can tell if two chemical compounds x_i & x_j are similar.

So sometimes we dont get vectors as inputs. We are given similarity matrices.

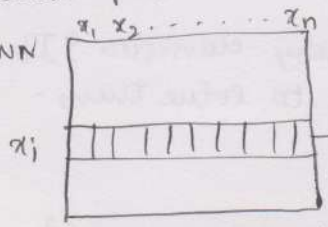


$\rightarrow S_{ij} = \text{similarity}(x_i, x_j)$

Distance matrix can be calculated as $d_{ij} = \frac{1}{s_{ij}}$

→ Some techniques work with distance/similarity matrices. Some don't.

ex:- KNN



→ get 5 lowest vals for 5 nearest neighbours

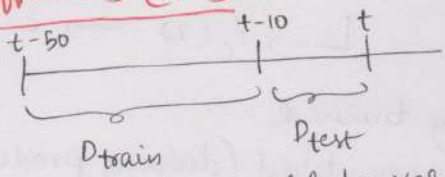
→ Logistic regression ~~that~~ will not work. It needs x_i vectors explicitly.

→ correlation matrix ($d \times d$) tells how correlated two features are.

Similarity matrix ($n \times n$) tells how similar two datapoints are.

Train & test set differences (4.5) :-

time based split :-



Sometimes D_{train} & D_{test} could be very different.

ex:- Amazon might have added a new category which ~~day~~ did not exist t-11 days ago (D_{train})

ie., Distribution of data has changed from D_{train} to D_{test}

→ One of the checks that need to be performed is do D_{train} & D_{test} have the same distribution. If they changed, no model can help.

→ If distributions are different, ~~test~~ train error & CV error → low

test error → high.

How to determine same distribution or not :-

$D_n \rightarrow D_{train}\ (x_i, y_i)$ +ve/-ve

$D_n \rightarrow D_{test}\ (x_i, y_i)$ +ve/-ve

$\Downarrow$

$D'_n \rightarrow D'_{train}\ \ y'_i = 1\ ;\ x'_i = \text{concat}(x_i, y_i)$ in D_{train}

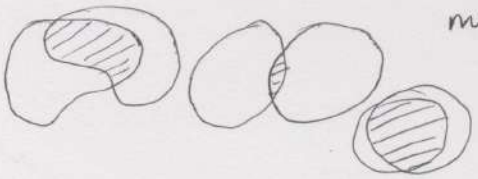
$D'_n \rightarrow D'_{test}\ \ y'_i = 0\ ;\ x'_i = \text{concat}(x_i, y_i)$ in D_{test}

Then use binary classifier on D'_n

$f(x) \rightarrow +1$ or 0

If it is able to seperate, then train & test are different.

more overlap = low accuracy for any classifier = distributions are very similar.



less overlap = medium accuracy = dist are different somewhat

~~Lot of over~~ very little overlap = high accuracy = distributions are very different
⇒ features are not temporally stable. Then we change features/build new features

If case 2 happens, (medium overlap), we remove features that are not stable across train & test. ie, most important features from model are removed ⇒ leftover features end up being less discriminative.

feature importance. Not possible in KNN but possible in others

Impact of Outliers :- (4.6)

→ when k is small, outliers can easily impact the model i.e., more prone to outliers changing the model

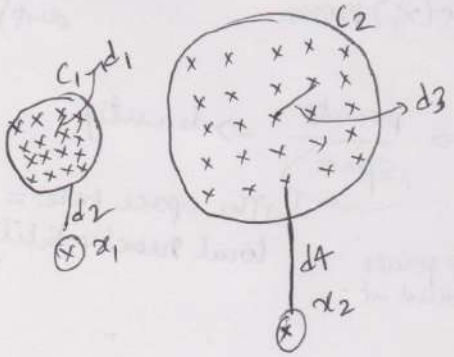
→ After 10 fold CV, if accuracy is
1 fold → 97%.
2 fold → 97%.
3 fold → 97%.
4 fold 97%.
5 fold 95%.
6 fold 92%. . . .
(dropping accuracy)

Always pick highest k value with highest accuracy. Because they are less prone to errors.

→ Local Outlier Factor is a technique inspired by KNN that will help detect outliers

→ Noise → Garbage, useless null information
Outlier → Definite & Accurate but not in same range as other values.
Outliers are useful in medical diagnosis (like MRI)

Local Outlier Factor :- (4.7)



C_1 → very ~~sparse~~ dense clusters
C_2 → sparse clusters
x_1, x_2 → outliers

Simple solution to find Outliers :-
→ take random point from each cluster & the two outliers. Calculate average distance to the k nearest neighbors
if avg distance is high, then it's an outlier.
But this solution won't declare x_1 as outlier as $d_3 \geq d_2$

→ workaround we try to capture the density of the nearest neighbours to the outlier

K-distance (4.8) :-

K-distance = distance to the kth nearest neighbour of x_i from x_j

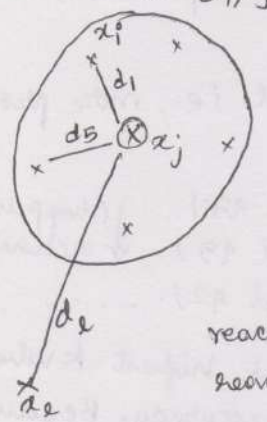
$N(A)$ = neighbourhood of A.

ex:- 5-distance (x_i) = dist to 5th nearest neighbour

$N_{k=3}(x_1)$ = set of the 3 nearest neighbours to K

Reachability Density (4.9) :-

$$\text{reachability density}_{(x_i, x_j)} = \max\left(\text{k-distance}(x_j),\ \text{dist}(x_i, x_j)\right)$$



$N(x_j)_{k=5} = \{x_1, x_2, x_3, x_4, x_5\}$

if $x_i \in N(x_j)$

then reach_dist(x_i, x_j) = k dist(x_j)

else

reach_dist(x_i, x_j) = dist(x_i, x_j)

reach dist$(x_i, x_j) = d_5$

reach dist$(x_i, x_2) = d_2$

→ Tells us within what distance is a point reachable.

It is not commutative i.e., reach_dist$(x_i, x_j) \neq$ reach_dist(x_j, x_i)

Local reachability density (LRD) (4.10) :-

$$\text{Lrd} = \frac{1}{\left(\sum_{x_j \in N(x_i)} \left\{ \frac{\text{reach_dist}(x_i, x_j)}{|N(x_i)|} \right\}\right)}$$

→ Average reachability dist (x_i) from it's neighbours

$|N(x_i)|$ → Number of elements in neighbourhood $(x_i) \approx K$ (could be other than K if multiple points are at same dist)

$\text{lrd}(x_i)$ = inverse of average reachability dist(x_i) from its neighbours.

$$= \frac{|N(x_i)|}{\sum_{x_j \in N(x_i)} (\text{reach_dens}(x_i, x_j)}$$

→ $\frac{\text{points}}{\text{space}}$ ⇒ density

↳ The space here = local reachability.

If $\text{lrd}(x_i)$ is large ⇒ lots of points around it.

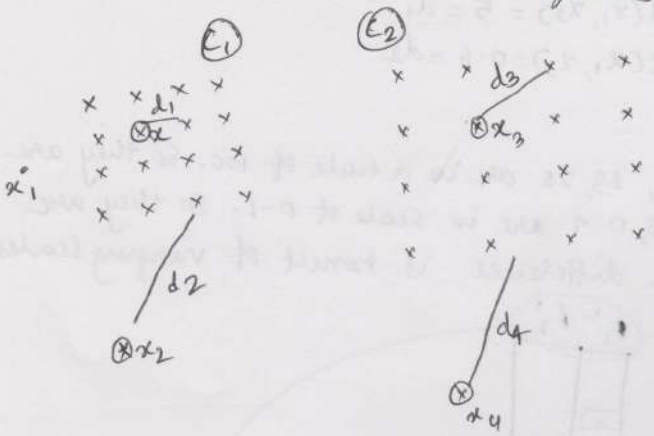
Local Outlier Factor (A) (4.11) :-

$$\text{LOF} = \underbrace{\frac{\sum_{x_j \in N(x_i)} \text{lrd}(x_j)}{|N(x_i)|}}_{\text{avg lrd of all points in } N(x_i)} * \frac{1}{\text{lrd}(x_i)}$$

→ if this is large or if this is small, LOF will be large.

If lof is large, we conclude it to be an outlier

LOF(x_i) is large $\longrightarrow$ outlier $\rightarrow$ if density of x_i is small compared to its neighbours



avg distances = d_1, d_2, d_3, d_4

$d_1 < d_2 < d_3 < d_4$

$$\text{lrd} = \frac{1}{\text{avg. reach dist}}$$

$$= \underbrace{\frac{1}{d_1}}_{\text{lrd}(x_1)} > \underbrace{\frac{1}{d_2}}_{\text{lrd}(x_2)} > \underbrace{\frac{1}{d_3}}_{\text{lrd}(x_3)} > \underbrace{\frac{1}{d_4}}_{\text{lrd}(x_4)}$$

lrd(x_1) $\approx$ lrd($x \in N(x_1)$) $\longrightarrow$ lrd(x_1) is similar to lrd of its neighbours $\rightarrow$ lof is small

lrd(x_2) $<$ lrd($x \in N(x_2)$) $\longrightarrow$ lof is large

lrd(x_3) $\approx$ lrd($x \in N(x_3)$) $\longrightarrow$ density is small but so is the density of rest of its neighbours $\leftarrow$ lof is small

lrd(x_4) $<<$ lrd($x \in N(x_4)$) $\longrightarrow$ lof is large

$\Rightarrow$ lof($x_1 \in C_1$) $\longrightarrow$ small large lof = outlier

lof(x_2) $\longrightarrow$ large

lof($x_3 \in C_2$) $\longrightarrow$ small

lof(x_4) $\longrightarrow$ large

$\rightarrow$ For each point x_i, compute lof(x_i)

pick points with highest lof $\rightarrow$ most of them will be outliers

$\rightarrow$ No need to remove all outliers. If from domain knowledge, we know 5% are outliers, we just remove top 5% lof values. Otherwise it's a subjective decision.

<u>Disadvantages</u> :- ① It's a ratio. So it's hard to determine.

Local Outlier Probability :- Puts lof in range b/w 0 & 1.

Feature bagging :- ensemble approach.

$\rightarrow$ There are many other outlier detection techniques.

$\rightarrow$ sklearn.neighbours.LocalOutlierFactor(n_neighbours, ...)

contamination = what % are outliers.

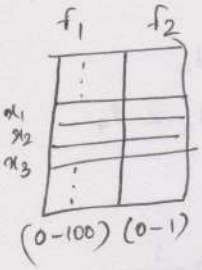
$\rightarrow$ Boxplots are suitable if we are checking univariate outliers ~~in each~~ while LOF are for multivariate outlier detection.

$\rightarrow$ For low latency systems, we don't use LOF, we use techniques like RANSAC.

$\rightarrow$ As part of preprocessing, we can remove all outliers on train data using LOF

$\rightarrow$ Recommended to take range of k values. Elbow method plot used to plot k v/s LOF value plot.

(22) Impact of Scale and Column Standardization (4.12):-

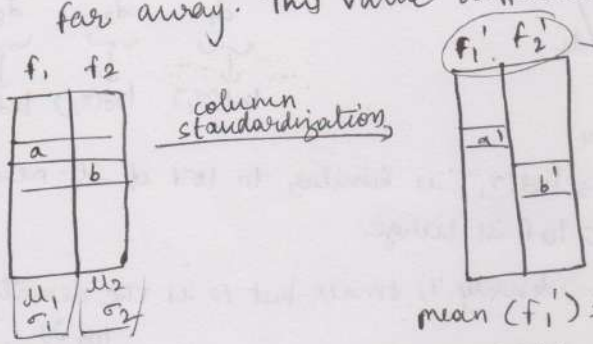


$x_1 = (23, 0.3)$ $\quad d(x_1, x_2) = 5 = d_1$

$x_2 = (28, 0.3)$ $\quad d(x_1, x_3) = 0.6 = d_2$

$x_3 = (23, 0.9)$

even though $d_1 > d_2$, 23, 28 are in a scale of 100. So they are pretty close. But 0.3, 0.9 are in scale of 0-1. So they are far away. This value difference is result of varying scales.



$$a' = \frac{a - \mu_1}{\sigma_1}$$

$$b' = \frac{b - \mu_2}{\sigma_2}$$

$\text{mean}(f_1') = 0 = \text{mean}(f_2')$

$\text{std-dev}(f_1') = 1 = \text{std-dev}(f_2')$

All models & techniques are applied on these → Transformed features.

→ Since KNN works with Euclidean dists, it is imp to perform col stand before performing KNN. Some techniques like decision trees are scale independent.

→ To standardize query point, training mean & std-dev are used.

→ standardScaler() is recommended over MinMax Scaler when we have large outliers in data.

Model Interpretability v/s Blackbox :- (4.13)

ex:- cancer/no cancer.

x (dataset) $\longrightarrow$ f (model) $\longrightarrow$ y_x (class output). But doctors just can't go about with just this 'y_x'. They need reasoning why.

If f returns, feature f_x has a high value & feature f_y has low value, then it's easier for domain experts to come to their own conclusion/justify the prediction.

Such a model is called interpretable model. Models that cannot give such are called blackbox models.

→ KNN is an interpretable model if k & d are small. We could return the nearest k neighbours that are true/false & show that they are similar to the current query point.

→ Important to know whether model is interpretable or not & if it is interpretable, how to interpret it.

Examples of whitebox :- Linear & decision/regression models (transparent & understandable features)

" " Black box :- deep learning, boosting & random forest models (non linear & hard to understand in general)

-> Accuracy $\propto \frac{1}{\text{Interpretability}}$.

Feature Importance & Forward Feature Selection :- (4.14)

$\underbrace{f_1, f_2, f_3, \ldots, f_{10}}_{D_n^{10}}$ $\longrightarrow$ f (model) $\longrightarrow$ if it can tell which features are most important to the task

Feature importance $\longrightarrow$ sort features by importance $\rightarrow$ f_4, f_3, f_5, ⋮

This is useful in understanding a model better.

Thus increasing interpretability.

-> KNN does not give FI by default.

-> Log Regres, Dec Trees can by some mods.

-> Any model can give us imp features. This is called forward feature selection

-> If $d=1000$, we can keep top 100 features & discard rest 900.

$\{f_1, f_2, f_3, \ldots, f_d\}$

only $f_1 \longrightarrow$ [Tr | Test] $f_{(f_1)}$ $\longrightarrow$ accuracy a_1

$f_2 \longrightarrow$ [ ] $\longrightarrow$ accuracy a_2

⋮

$f_n \longrightarrow$ [ ] $\longrightarrow$ " a_n.

Let highest $= f_{10}$.

Now

f_1 & f_{10} $\longrightarrow$ [ ] $\longrightarrow$ accuracy a_1

f_2 & f_{10} $\longrightarrow$ [ ] $\longrightarrow$ " a_2

⋮

Let highest $= f_6$.

=> End of 2nd iteration, $\{f_{10}, f_6\}$ are selected features with highest accuracy.

f_{10}: most useful; f_6: Second most useful

-> This is called forward selection. Picking most important features given the already have some features selected.

(24) Backward Selection :- Selects all features in the begining. Removes a feature in each iteration which results in lowest drop in accuracy.

→ Time Complexity is very high.

→ They are model agnostic.

→ It is sometimes useful to remove highly correlated features before PCA.

→ We should perform both forward & backward selection as both give diff to get the best set of features.

Handling categorical & Numerical features (4.15) :-

→ For model we need numerical values only.

→ Categorical = text.

→ We can't give numbers to each word as words have an inherent order.

→ Alternative is One Hot Encoding :- Creates binary vector of the size (number of distinct elements.

ex:- set = {Black, Brown, Re, Gr, Blue}

Black = | 1 | 0 | 0 | 0 | 0 | . . .

feature is converted to 5 more features.

→ Drawback is when number of distinct elements is large, number of new features = large = sparse & large vectors

→ One technique can be to replace categorical column with nominal.

ex:- country feature in height prediction model can be replaced with average country height - This is called Response Coding.

→ Another technique is to use domain knowledge

ex:- if dist from equator affect height, replace country feature with that.

→ Ordinal features :- features that can be ordered.

ex:- do you like India? V good, good, avg, bad, V bad } → categorical features but ordered. So can be converted to numerical. There is no right numerical value. Have to use domain knowledge.

→ We just toy allot them & choose the best one.

→ Label encoding replaces all categorical features with numbers. Algos like decision trees, random forests work with categorical values just fine & this can be used.

Handling missing value by imputation :- (4.16) (28.72%)

-> could be because of data corruption / collection errors

-> There are ways to featurize missing values.

① Imputation :- three ways (i) mean (ii) median (iii) most freq (mode)

sklearn.preprocessing.Imputer(missing_values='NaN', strategy=' ', ...)

In classification tasks, impute based on class label

if f_3 of x_i is missing & y_i is 1, replace f_3 of x_i with mean of f_3 where $y_i = 1$. This is called mean imputation based on class label.

-> new missing value features :- fill missing values by imputing but also add new feature 0/1 based on if the feature is missing or not. Because missing values could also sometimes be sources of information.

-> Model based imputation :- whatever rows have missing features, we put them in test set & use classification / regression to predict missing features. KNN is used for model based imputation. They take more time than other simple imputers.

-> Sometimes missing features are close to previous data. ex :- temperature of day. ~~We can~~ It is unnecessary to take averages of all temps. In such case we can curve fitting (or) pandas.interpolate(time)

Curse of Dimensionality :- (4.17) :-

-> As dimensionality increases, the # of points to perform good classification (model) increases exponentially.

Hughes Phenomenon :-

if number of datapoints is fixed, performance decreases as dimensionality increases.

It first increases & then decreases.

-> Distance functions (esp Euclidean functions) :- The intuition of distances in 3D is not valid in higher dimensional spaces

let $dist_min = \min_{x_j \neq x_i} (dist(x_i, x_j))$ ⟶ dist to closest neighbors

$dist_max = \max_{x_j \neq x_i} (dist(x_i, x_j))$ ⟶ dist to the farthest neighbors.

$$\frac{dist_max(x_i) - distmin(x_i)}{distmin(x_i)} > 0$$ ⟶ true for 1D, 2D, 3D.

But as dimensionality increases, this becomes closer to 0.

$$\lim_{d \to \infty} \frac{\text{distmax}(x_i) - \text{distmin}(x_i)}{\text{distmin}(x_i)} \longrightarrow 0.$$

⇒ At high dimensions distmax(x_i) ≈ distmin(x_i)

⇒ " " " every pairs of points are equidimensional.

⇒ Euclidean dist does not make logical sense.

Solution :- Use cosine similarity. The impact of higher dimensions is lesser on cosine similarity.

→ high dimensional & dense data → impact of COD is high

high dimensional & sparse data → impact of COD is relatively low

not uniform/random spread of data.

→ Overfitting & Underfitting :- As $d \uparrow$, chances of overfitting ⇑.

Solution :- forward feature selection } → classification (model) oriented.

PCA, tSNE

↓

not classification (model) oriented

Some can use cosine-dist instead of Euc dist.

Some use sparse rep instead of dense representation.

Bias Variance Trade off :- (4.18) :-

→ Generalization error :- error made on future unseen data. Goal of ML is to have less generalization errors

It can be decomposed into sum of three terms

$$\text{Bias}^2 + \text{Variance} + \text{Irreducible error}$$

Bias^2: errors due to simplifying assumptions

high bias = underfitting.

ex:- k=n in KNN when unbalanced data is seen.

Irreducible error: Error that we cannot reduce further for a given mod.

Variance: How much a model changes as training data changes

a decision surface

Small changes in datasets result in large changes in the model. Then it called high variance model aka overfit model

Gen Error = Bias^2 (↓) + Variance (↓) + irreducible error

↓ (no underfit) ↓ (no overfit)

if k=1 → Bias ↓ Var ⇑

as k ↑ → Bias ↑ slightly Var very drastically ↓

as k → n Bias ↑ Var ↓

Goal is to find balance.

Intuitive Understanding of Bias-Variance (4.19) :-

$D \rightarrow D_{train} \rightarrow$ model.
$D \rightarrow D_{test}$.

$x_i \rightarrow$ (model) $\rightarrow \hat{y}_i$
$\in D_{train}$

train error $= \text{diff}(y_i, \hat{y}_i)_{train}$
test error $= \text{diff}(y_i, \hat{y}_i)_{test}$.

high bias / ~~over~~ under fit :-

train error ↑ ⇒ bias ↑
train error ↓ ⇒ bias ↓.

high variance / overfit :-

train error ↓ test error ↑ ⇒ overfit ⇒ variance ↑
(or)
training data changes slightly & model changes a lot

Best & worst case of algorithm (4.21) :-

for KNN, if d is small (<10; not large) → KNN is good.
But as d ↑, curse of dimensionality kicks in, Euclidean dist isn't a good measure
interpretability decreases.
runtime complexity of kd tree / LSH ↑
→ KNN should NOT be used in low latency systems
→ If we can find the right distance measure KNN is good.
ex:- if we are using cosine distance / similarity matrix is given

Performance prediction of models : Accuracy (5.1) :-

$$\text{accuracy} = \frac{\#\text{ of correctly classified}}{\#\text{ of pts in } D_{test}}$$

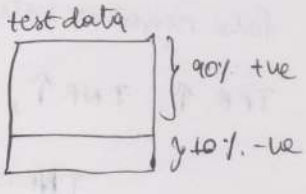
↓ easy to understand

0 (↓ bad) to 1 (↓ good)

→ measures are done only on test data

Drawbacks :-

→ imbalanced dataset :-



dumb model
[return +ve]
↓
90% accuracy

∴ Never use accuracy on ~~dumb model~~ imbalanced dataset.

→ **probability scores :-**

x	y	M_1	M_2	$\hat{y}_1$	$\hat{y}_2$
x_1	1	0.9	0.65	1	1
x_2	1	0.9	0.71	1	1
x_3	0	0.1	0.45	0	0
x_4	0	0.15	0.42	0	0

→ M_1 & M_2 return prob of value being 1.
But both M_1 & M_2 return same $\hat{y}_1, \hat{y}_2$.
Accuracy can't be used here.

(28) → for models that give probabilities, number of outputs depend on # of classes.

Confusion Matrix, & TPR, FPR, FNR, TNR (5.2):-

Confusion matrix:- Does not / Cannot process probability scores

ex:- binary classification

pred ($\hat{y}_i$) ↓ \ actual → (y_i)	0	1
0	a	b
1	c	d

a = # of correct 0's predicted
b = # of 1's predicted as 0
c = # of 0's predicted as 1's
d = # of 1's correctly predicted.

multi class with c classes

	0	1	. . .	c − 1
0				
1				
⋮				
c−1				

principal diagonal elements. These should be high

Rest of the elems are called off diagonal elements. These should be small.

if model is sensible

↑	small
small	↑

→ Binary classification confusion matrix

pred ↓ \ actual →	0	1
0	TN	FN
1	FP	TP
	N	P

N → total # of negatives

P → total # of +ves.

$n = N + P$

TN = true negative FN = false negative
FP = false positive TP = true positive.

true positive rate $= \frac{TP}{P}$

true negative rate $= \frac{TN}{N}$

false positive rate $= \frac{FP}{N}$

false negative rate $= \frac{FN}{P}$

→ Model is said to be good if TPR ↑, TNR ↑, FPR ↓, FNR ↓

ex:- dumb model

pred ↓ \ act →	0	1
0	900	100
1	0	0

$y_0 = 900$
$y_1 = 100$.

$TNR = \frac{900}{900} = 100\%$ ✓

$TPR = \frac{0}{100} = 0\%$ ✗

$FNR = \frac{100}{100} = 100\%$ ✗

$FPR = \frac{0}{900} = 0\%$ ✓

→ accuracy could not help against dumb models. But confusion matrix can

→ catch is accuracy ~~has~~ gives out only 1 number. But CM gives 4

→ which rate is more important is very domain specific.

ex:- in medical appns, we need very high TPR & very low FNR

→ in multiclass problems, we take one v/s rest approach.

ex:- classes = {A, B, C} → TPR A, TPR of B, TPR of C, FPR of A, ...

→ cosine distance is equivalent to Euclidean distance when input vectors are normalized.

Precision and Recall, F1 Score :-

→ Used in information retrieval (Search engines etc, When there is lot of text & you only need some of it)

$$\text{precision} = \frac{TP}{TP+FP}$$ ← what % of really positive points did model predict out of all positively predicted points.

We are not really worried about negative class.

pred \ act	0	1
0	TN	FN
1	FP	TP

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

→ Both precision & recall care about positive class only.

→ We want precision ↑ & recall ↑.

$$\rightarrow \text{F1 score} = \left(2 * \frac{pr * recall}{pr + recall}\right)$$

Based on Harmonic mean → inverting the precision & recall, taking their avg, and inverting it

$$= \frac{2}{\frac{1}{pr} + \frac{1}{re}} = 2 * \frac{1}{\frac{pr+re}{pr \times re}} = 2 \times \frac{pr \times re}{pr + re}$$

→ often used in Kaggle comps.

→ We can adjust the probability threshold (say 0.5 to 0.4) to get more positive predictions thus more recall, but less precision. This is called precision recall tradeoff.

→ We use harmonic mean instead of simple mean as it punishes extreme values more. Harmonic mean of two numbers is never higher than the geometric mean. It also tends towards the least number minimizing the impact of outliers & maximizing the impact of small ones. The f-measure tends to previlege balanced systems

→ G1 score = $\sqrt{\text{precision} * \text{recall}}$. Not very popular.

(30) Reciever Operating characteristic (ROC) curve and AUC :- (5.4) :- (30.24)

→ Designed by electronic engineers to see how well their missiles were working

steps :- ① Sort values based on probability score

x	y	$\hat{y}$ ↓
x_1	1	0.95
x_2	1	0.92
x_3	0	0.80
x_4	1	0.76
x_5	1	0.71

→ tub 1, 0.95 is threshold

→ tub 2, 0.92 is threshold

② put $\tilde{y}$ as 1 if $\hat{y} \geq 0.95$

x	y	$\tilde{y}_{0.95}$	$\hat{y}$	$\tilde{y}_{0.92}$
x_1	1	1	0.95	1
x_2	1	0	0.92	1
x_3	0	0	0.80	0
x_4	1	0	0.76	0
x_5	1	0	0.71	0

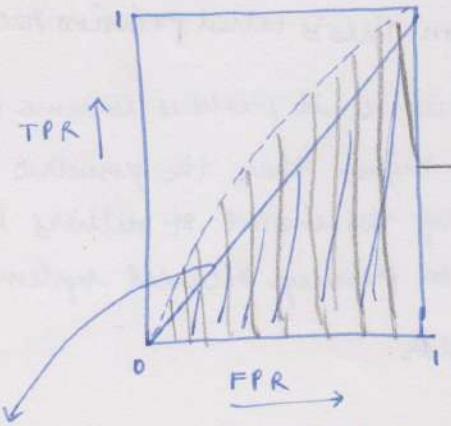
→ we can calculate TPR, FPR corresponding to $\tilde{y}_{0.95}$, $\tilde{y}_{0.92}$, ...

if there are n points, we can have n thresholds

FPR_1, FPR_2, FPR_3, ..., FPR_n

TPR_1 TPR_2 TPR_3 TPR_n

→ If model is sensible the graph of these points will look like



→ This curve is called reciever operating characteristic curve

→ AUC = Area under curve. could lie b/w 0 & 0

0 —— 1

terrible very good

line breaking square into two halves

→ This is typically used for binary classification.

Why use AUC? :-

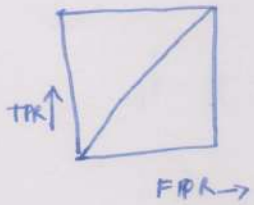
① unbalanced data/ → AUC can be high for a simple/dumb model

② AUC is not dependant on $\hat{y}$. Only ordering of probabilities.

x_1	y	$M_1 \hat{y}_1$	$M_2 \hat{y}_1$
x_1	1	0.95	0.2
x_2	1	0.92	0.2
x_3	0	0.80	0.1
x_4	1	0.76	0.01
x_5	1	0.72	0.69

→ Both M_1 & M_2 have same AUC score

③ AUC of random model = 0.5



④ If a model has very low AUC (0.2 for), just swap 1 to 0 & 0 to 1. Then AUC will become (1 − 0.2). Do this when AUC < 0.5. Happens when there's a mistake in modelling.

→ Sometimes we need one metric to asses how good a model is. AUC is one such metric. It combines both TPR & FPR.

→ AUC = probability that we predict correctly using the model.

→ It helps us find the best threshold. We can put high threshold if we dont want false positives or put very low threshold if we dont want false negatives. In ROC we check if model is able to discriminate b/w positive and negative classes.

Log loss (5.5):-

→ Uses probability scores.

→ if there are n points in test set, $\text{log-loss} = \underbrace{\frac{-1}{n}}_{\text{negative of average}} \sum_{i=1}^{n} \underbrace{\left\{ (\log(P_i) * y_i) + (1-y_i) * \log(1-P_i) \right\}}_{\text{log loss}}$

→ penalizing for small deviations in probability score

→

x	y	$\hat{y}$	
x_1	1	0.9	→ $-\log(0.9) = 0.0457$ → low
x_2	1	0.6	→ $-\log(0.6) = 0.22$
x_3	0	0.1	→ $-\log(0.9) = 0.0457$ → high
x_4	0	0.4	→ $-\log(0.6) = 0.22$

negative average of these vals = log loss

we want as small as possible

→ Easily extended to multi class problems

$$\frac{-1}{n} \sum_{i=1}^{n} \sum_{j=1}^{c} \underbrace{y_{ij}}_{\substack{1 \text{ if } x_i \in \text{class } j \\ 0 \text{ otherwise}}} \log \underbrace{(P_{ij})}_{\text{prob that } x_i \in \text{class } j}$$

c = number of classes.
Multi class has one v/s rest predictions.

→ Intuitively both log loss & AUC scores are trying to measure discriminatory power of positive & negative class.

Root Squared (R^2) (or) Coefficient of Determination :- (5.6) :-

→ Regression performance measure

→ x_i, y_i, $\hat{y}_i$ error $= y_i - \hat{y}_i$

y_i → actual value

$\hat{y}_i$ → value predicted by model

→ SS_{total} = total sum of squares $= \sum_{i=1}^{n} (y_i - \bar{y})^2$ where $\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i$

→ SS_{res} = Sum of squares of residues $= \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (e_i)^2$

residue $= y_i - \hat{y}_i = e_i$

→ $R^2 = \left(1 - \frac{SS_{res}}{SS_{tot}}\right)$

case ①:- $SS_{res} = 0$ (ie., $e_i = 0$) $\Rightarrow R^2 = 1$ (best value)

case ②:- $SS_{res} < SS_{tot} \Rightarrow R^2 = 0$ to 1

0 ↑ bad, 1 ↑ good

case ③:- $SS_{res} = SS_{tot}$

ie., model returns all means

$\Rightarrow R^2 = 0$

case ④:- $SS_{res} > SS_{tot} \Rightarrow R^2 = -1$. Model is worse than a simple mean model

→ Major problem with R^2 is that it will stay the same / increase with addition of more features to the regression problem even if they don't have any relationship with y_i. Adjusted R^2 / $\overline{R^2}$ penalizes for adding features that do not improve the existing model.

Median Absolute Deviation (5.7):-

→ One problem with R^2 is if one e_i is very large it will effect whole R^2 value
R^2 is not very robust to outliers

→ If we think of e_i as a random variable,

$$\text{median}(e_i) = \text{central } \cancel{\text{median}} \text{ value of errors}$$

$$MAD(e_i) = \text{median}\left(|e_i - \text{median}(e_i)|\right) \rightarrow \text{robust measure of std-dev}$$

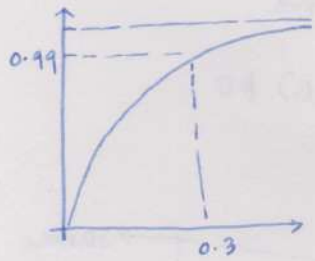
→ if median & MAD are small, ⇒ error is also small.

Distribution of Errors (5.8):-

→ Just like we used statistical measures (mean, median, std-dev, MAD) we can also use probabilistic methods like PDF, CDF

→

most e_i's are small



99% of errors are < 0.3
1% of errors are ≥ 0.3

→ if we have two models, M_1 & M_2, and M_2 CDF is below M_1, we can choose M_1.

→ Mean Absolute Percentage Error

$$M = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y-\hat{y}}{y}\right|$$

But this is prone to divide by zero.

$$\text{Modified MAPE} = \frac{\frac{1}{n}\sum_{i=1}^{n}\left|y-\hat{y}\right|}{\bar{y}} = \frac{1}{n}\text{Residual SS} = \frac{1}{n}SS_{Res}.$$

Conditional Probability (7.1):- (31.32)

→ Naive Bayes classification algo based on basics of probability

→ Conditional prob :-

$$P(A|B) = \text{Prob}(A = a \mid B = b)$$

value

↳ conditioned on

A: Random Variable
B: Random Variable.

ex: Dice roll ~~P(~~ → 36 possibilities

$$P(B \le 5) = \frac{10}{36}$$

$$P(A = 2 \mid B \le 5) = \frac{3}{10}$$ (see wikipedia article if confused)

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

$$\frac{3/36}{10/36} = \frac{3}{10}$$

iff $P(B) \,!= 0$

(34) → We can only apply conditional probability iff A & B are independent.

Two events A, B are said to be independent iff

$$P(A \text{ and } B) = P(A)P(B)$$

→ Correlation and Independence are synonyms in english. But not in probability and statistics.

Independent Events & Multially Exclusive Events (7.2):-

→ Two Events A, B are independent if

$$P(A|B) = P(A) \quad , \quad P(B|A) = P(B)$$

→ $P(A|B) = P(B|A) = 0$, then A & B are multially exclusive.

Bayes Theorem with Examples (7.3):-

→ Very old Theorem.

→ $$\underbrace{P(A|B)}_{\text{posterior probability}} = \frac{\overbrace{P(B|A)}^{\text{likelihood}}\ \overbrace{P(A)}^{\text{prior}}}{\underbrace{P(B)}_{\text{evidence}}} \quad \text{if } P(B) \neq 0$$

proof:- $$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A,B)}{P(B)}$$ → same

We know $P(A \cap B) = P(B \cap A)$

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

$$P(B|A) = \frac{P(B \cap A)}{P(A)} \Rightarrow P(B \cap A) = P(B|A) \cdot P(A)$$

$$\Rightarrow \boxed{P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}}$$

Bayes Theorem Exercises:- (7.4):- (31.5)

→ free method

→ Read notes on probability.

Naive Bayes Algorithm :- (7.5) :- (31.94)

→ n features and k class labels

$$\underbrace{P(C_k|x)}_{\text{posterior}} = \frac{\overbrace{P(C_k)}^{\text{prior}}\ \overbrace{P(x|C_k)}^{\text{likelihood}}}{\underbrace{P(x)}_{\text{evidence}}}$$

posterior → Probability of being class 'k' given 'x'

evidence → Same for all values. It's a constant. Ignoring it

$$P(C_k)\cdot P(x|C_k) = P(C_k \cap x) = P(C_k, x)$$

$$\Rightarrow P(C_k|x) \propto P(C_k, x)$$ → Joint probability model.

x is an n dimensional vector $(x_1, x_2, x_3, \dots x_n) = x$

$$\Rightarrow P(C_k|x) \propto P(C_k, x_1, x_2, x_3, \dots, x_n)$$

Using chain rule

$$P(C_k, x_1, x_2, \dots, x_n) = P(x_1, x_2, x_3, \dots, x_n, C_k)$$
$$= P(x_1|x_2, x_3, \dots, x_n, C_k)\cdot P(x_2, x_3, \dots, x_n, C_k)$$
$$= P(x_1|x_2, x_3, \dots, x_n, C_k)\cdot P(x_2|x_3, x_4, \dots, x_n, C_k)$$
$$\vdots$$

$$P(C_k|x) \propto P(C_k, x) = \underbrace{P(x_1|x_2, x_3, \dots x_n C_k)}\cdot P(x_2|x_3, \dots, C_k) \dots P(x_n|C_k)\cdot P(C_k)$$

But in order to compute this probability we need atleast few points. But this is hard to compute & low chance of finding such points.

Hence we make naive conditional assumptions

Simple independence ~~if~~ $P(A|B) = P(A)$

If $P(A|\underbrace{B, C}_{B \cap C}) = P(A|C)$, A is conditionally independent to B.

Naive assumption is, $P(x_i|x_{i+1}, x_{i+2}, \dots, C_k) = P(x_i|C_k)$

i.e., x_i is conditionally independent to $x_{i+1}, x_{i+2}, \dots, x_{i+n}$

$$\Rightarrow P(C_k|x) \propto P(x_1|C_k)\cdot P(x_2|C_k) \dots P(x_n|C_k)\cdot P(C_k)$$
$$\propto P(C_k)\cdot\prod_{i=1}^{n} P(x_i|C_k)$$

$$\Rightarrow P(C_k|x) = \frac{1}{Z}\cdot P(C_k)\cdot\prod_{i=1}^{n} P(x_i|C_k), \quad Z = P(x) = \sum_k P(C_k)\cdot P(x|C_k).$$

But the class label is obtained by the max prob of $P(C_k|x)$

i.e., $\max(P(C_1|x), P(C_2|x), P(C_3, x), \dots P(C_k|x)$

$$\Rightarrow \hat{y} = \underset{k=\{1,2,\dots k\}}{\operatorname{argmax}}\ P(C_k)\cdot\prod_{i=1}^{n} P(x_i|C_k)$$ → This is called Maximum Aposteriori classifier.

Laplace Smoothing : Done to prevent zero probability in the algorithm
aka
Additive smoothing. ~~2+α~~ 1 added to the numerator & number of dimensions is added to the denominator.

→ Naive Bayes assumes there is no correlation among the features.

Naive Bayes with Example (7.6) :-

→ Naive Bayes has training & testing ~~times~~ unlike KNN.
→ In training phase, we compute likelihoods for all features & P(Class)
→ $TC = O(nd)$, By some optimization, $O(n)$ can be obtained.
Space Complexity $= O(d * c)$
→ We can store all probabilities in a python dictionary.
→ Test time complexity $= d * c$ lookups from dictionary
$= O(d * c)$
→ We can use One hot encoding for text features as Naive Bayes works on basis of probabilities.
→ These are conditional probabilities. Not basic probability. Sum won't be equal to one. (Sum of all classes probabilities)

Naive Bayes on Text Data :- (7.7) :-

→ Naive Bayes one of the first for spam filtering.
→ Text Classification is one of the main popular areas.

→ text → remove stopwords, stemming, n-grams → $\{w_1, w_2, w_3, \dots, w_d\}$ (Binary BoW)

$$P(y=1 \mid text) : \propto P(y=1 \mid \underbrace{w_1, w_2, \dots, w_d}_{features})$$

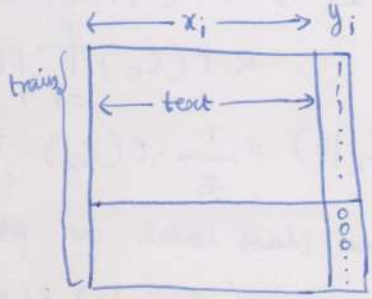
$$\propto \underbrace{P(y=1)}_{\text{class prior}} * \underbrace{P(w_1 \mid y=1) * P(w_2 \mid y=1) \dots * P(w_d \mid y=1)}_{\text{likelihoods}}$$

$$P(y=1 \mid text) \propto P(y=1) * \prod_{i=1}^{d} P(w_i \mid y=1)$$

$$P(y=0 \mid text) \propto P(y=0) * \prod_{i=1}^{d} P(w_i \mid y=0)$$

$$P(w_i \mid y=1) = \qquad w_i = \{w_0, w_1, \dots, w_i, \dots\}$$

$$= \frac{\#\text{ train data points containing } w_i \text{ \& } y=1}{\#\text{ train data points with } y=1}$$



→ For many text classification problems,

{ spam detection, polarity of review } → Naive Bayes is a good baseline (used as a benchmark)

Laplace / Additive Smoothing (7.8):-

→ We encounter words that are not part of training set, while testing the model.

ex:- w' is a word encountered for the first time in testing

$$P(y=1 \mid text_q) = P(y=1 \mid w_1, w_2, w_3, w')$$
$$= P(y=1)\, P(w_1 \mid y=1)\, P(w_2 \mid y=1)\, P(w_3 \mid y=1)\, P(w' \mid y=1)$$

$$P(w' \mid y=1) = \frac{\#\text{ of Training point with } w' \,\&\, y=1}{\#\text{ of points with } y=1} = \frac{0}{n_1}$$

can't just ignore it because ignoring means $P(w' \mid y=1) = 1$ which is wrong. Can't put 0 either whole thing will become zero.

→ workaround for this is Laplace Smoothing. (Not Laplacian Smoothing) aka Additive Smoothing.

$$P(w' \mid y=1) = \frac{0+\alpha}{n_1 + \alpha k}$$

k = # of distinct values w' can take. In this value, w is Bow, so 0 & 1. ∴ $k = 2$.

α = Typically 1

If α is small, we get rid of multiplication by zero problem.

If α is large, after smoothing we get ≈ 0.5 (No bias)

→ We do laplace smoothing for all points irrespective of their presence in training data.

→ As α increases we are moving likelihood probabilities to a uniform distribution.

→ $\alpha = 1$ is also called add-one smoothing.

Log Probabilities for numerical stability (7.9):-

$$P(y=1 \mid text) = P(y=1 \mid w_1 w_2 w_3 \ldots w_n)$$
$$= \underbrace{P(y=1)\, P(w_1 \mid y=1) \ldots P(w_n \mid y=1)}_{\text{We b/w 0 \& 1}}$$

→ will be a very small value if 'n' is large. Leads to numerical stability issues. Python double precision only has 16 significant values. Numerical underflow will happen

To prevent this we take

$\log(P(y=1 \mid text))$. Since log is a monotonic function, it should be okay.

$\Rightarrow \log(P(y=1 \mid w_1 w_2 \ldots w_d)) = \log(P(y=1)) + \sum_{i=1}^{d} \log(P(w_i \mid y=1))$

Since log converts multiplication to addition, it is helpful
exponents to multiplication

Bias Variance Tradeoff (7.10):-

high bias = underfitting.
high variance = overfitting.
→ α in laplace smoothing determines this (Hyperparameter)
→ If n is large, then words that occur rarely will have very low probability values. This will lead to overfitting (high variance)
→ If $\alpha = 0$, small change in D_{train} results in large change in the model. high variance ⇓ overfitting
→ If α = very large, for all i, $P(w_i \mid y=1) \approx P(w_i \mid y=0) \approx \frac{1}{2}$

$$\Downarrow$$

$$P(y_q = 1 \mid x_q) \approx P(y_q = 0 \mid x_q) \approx \frac{1}{2}.$$

Underfitting. If +ve class has more points than -ve class, +ve class is given as the final result.
⇓ (high bias)

→ We find appropriate α by simple cross validation (or) 10-fold cross validation.
→ After certain loss decrement, the loss does not decrease much & seems constant from that point i.e., accuracy can't be improved further. This is called asymptotic accuracy. Naive Bayes achieves this in $\log n$ time.

Feature Importance and Interpretability (23.02.1.) :- (7.11):-

$\forall w_i \rightarrow$

w_i	$P(w_i \mid y=0)$	$P(w_i \mid y=1)$

→ sorted by descending orders of likelihoods i.e., top words have high probability

⇒ w_i with highest prob are important in determining target class.
⇒ $P(w_i \mid y=0)$ → high, w_i is imp feature.

feature importance in NB:- +ve class:- find words (w_i) with highest value of $P(w_i \mid y=1)$
-ve class:- " " " " " $P(w_i \mid y=0)$

→ feature importance can be obtained directly from the model unlike naive Bayes where we used forward feature selection etc.

Interpretability :-

$x_q \longrightarrow y_q = 1$ because x_q contains words w_3, w_6, w_{10} which have high likelihoods

$y_q = 0$ because x_q contains w_4, w_5 which have very low likelihoods.

→ we can use both count based & binary BoW / TFIDF. But if count based is used then we have to use $\underbrace{\text{MultinomialNB}}_{\text{when features are not boolean.}}$ & in case of binary, we have to use $\underbrace{\text{BernoulliNB}}_{\text{when features are boolean}}$.

→ word word word

$\underbrace{P()\ \ P()\ \ P()}_{\text{features.}}$

a	1	p
a	1	p
b	2	q
c	1	p
b	2	a
c		

→ we see how important each category is.

Imbalanced Data (7.12) :-

$$\cancel{P(w_q \mid y=1)}\ P(y=1 \mid w_q) = P(y=1) \prod_{i=1}^{d} P(w_i \mid y=1)$$

$$P(y=0 \mid w_q) = P(y=0) \prod_{i=1}^{d} P(w_i \mid y=0)$$

if data is unbalanced i.e., $n \to n_1 (+),\ n_2 (-)$

$n_1 >> n_2 \Rightarrow P(y=1) >> P(y=0)$

if likelihoods product is same, difference in priors give undue advantage to majority class.

→ To overcome this

① Upsampling / Downsampling

↳ $n_1 \approx n_2 \Rightarrow P(y=1) = P(y=0) = 1/2$

② Drop $P(y=1)$ & $P(y=0)$.

$\Rightarrow P(y=1) = P(y=0) = 1$.

③ Modified NB. Not often used.

→ if $n_1 = 900$, $n_2 = 100$

$P(w_i \mid y=1) = \dfrac{\square}{900}$ → count out of 900 occurances

$P(w_i \mid y=0) = \dfrac{\square}{100}$ → count out of 100 occurances

⇓

large numerator, small numerator

→ Both multiplied with same α during laplace smoothing.

$\Rightarrow \alpha$ impacts more minority class than majority class.

ex:- $\frac{2}{100}$, $\frac{18}{900}$

both are 2%.

laplace smoothing

$\frac{2+10}{100+20}$, $\frac{18+10}{900+20}$

$\Downarrow$

$\frac{12}{120} = 10\%$. $\frac{28}{920} = 3.4\%$.

$\downarrow$

$2\% \rightarrow 10\%$. $2\% \rightarrow 3.4\%$.

Large difference

→ Upsampling / Downsampling can be used.

② tricks:- diff α for +ve & -ve class.

→ SMOTE :- Synthetic Minority Over-Sampling Technique. Basically creates new minority instances synthetically.

Nearmiss :- Undersampling technique.

Outliers (7.13) :-

→ Outliers in test data can be handled with Laplace Smoothing.

→ If word occurs less than 'x' times in training data, ignore that word.

↑ Basic soln for outliers in training data.

Another soln is laplace smoothing.

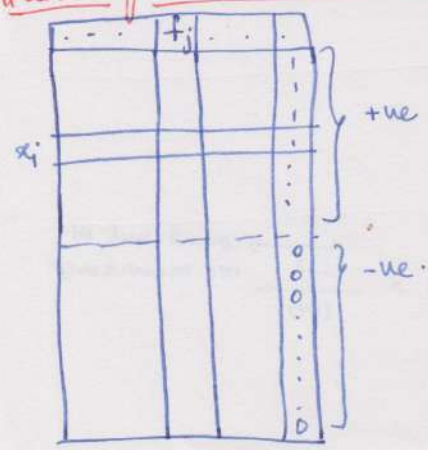
Missing Values (7.14) :-

→ Text data has no concept of Missing Data.

→ For categorical values, NaN itself can be considered as a category.

→ For numerical features, we use standard techniques like imputation.

Handling Numerical Features (7.15) :-

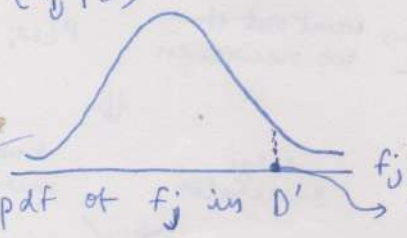


$$P(y_i = 1 \mid x_{i1}, x_{i2} \dots x_{id}) = P(y_i = 1) \prod_{j=1}^{d} P(x_{ij} \mid y_i = 1)$$

Let f_j = numerical.

Let $D' = D^{+ve}_{train}$, $D'' = D^{-ve}_{train}$

$\Rightarrow P(x_{ij} \mid D')$



pdf of f_j in D'

Assume its Gaussian Dist.

x_{ij}. This prob can be obtained from pdf.

This Gaussian Dist PDF can be obtained from $N(\mu, \sigma)$

→ Similarly D'' prob can also be obtained.

→ People <u>assume</u> D' & D'' are Gaussian Distributions with

$$N(\mu_j', \sigma_j') \ \& \ N(\mu_j'', \sigma_j'')$$

This model is called Gaussian Naive Bayes (We assume likelihood prob are Gaussian)

$P(w_1 \mid y=1)$ → Bernouli Naive Bayes.

0 1

→ Multinomial Naive Bayes.

→ PDF of a Gaussian distribution returns the relative likelihood probabilities rather than absolute likelihood value or real absolute likelihood value for any continuous variable is zero.

→ Low likelihood values mean rare model / incorrect model. Likelihood deals with fitting models given known data.

→ If data is combination continuous & categorical,

fit gaussian NB on continuous,

fit multinomial NB on categorical, transform all dataset by taking class assignment probabilities (with predict_proba) method as new features,

np.hstack((multinomial_probs, gaussian probs)) & refit new Gaussian model on new features

Multiclass Classification (7.16) :- (33.45)

NB can do multiclass.

$$\left.\begin{array}{l} P(y_i = 1 \mid W) \\ P(y_i = 2 \mid W) \\ P(y_i = 3 \mid W) \\ \vdots \end{array}\right\} \text{largest prob will be output}$$

→ NB-regression can be used for continuous class labels. Not often used.

Similarity Matrix or Distance Matrix (7.17) :-

⇒ Naive Bayes cannot handle distance Matrix / Similarity Matrix like KNN.

Large Dimensionality (7.18) :-

→ NB can handle large datasets. But we have to use log probabilities to get sensible answer.

(42) Best and Worst Case :- (7.19) :-

→ If the assumption of conditional independence of features is true, then Naive Bayes performs well. Performance deteriorates as this becomes false. When some features are dependant, NB works reasonably well.

→ Text Classification (email spam, review polarity) NB works really well.

→ Extensively used for Categorical. Not used for real valued.

→ Very interpretable. Runtime & Training time Space & time is low.

→ Easily overfit if laplace smoothing isn't done.

Code (7.20) :-

→ out of core Naive Bayes :- When data is very large & does not fit in RAM. Probability of data is calculated in chunks.

→ binarize :- Data has to be in binary form. if val ≥ x, it's 1, else 0.

→ Dataset → train, test

train → vectorizer → fit train → normalize

vectorizer → fit test

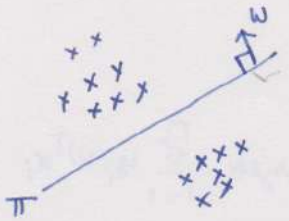
Interview Question :-

→ Multinomial NB does Standardization & Normalization automatically

Logistic Regression :- (8.1) (33.69) :-

→ classification technique.

→ It plane data is linearly seperable / almost linearly seperable



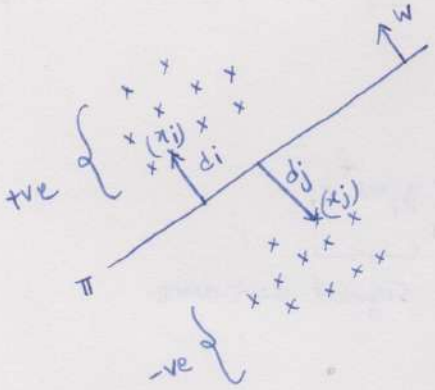
$\pi : (w, b) = w^T x + b$

w → normal to π, b → intercept

if π passes through origin, $b = 0 \Rightarrow \pi = w^T x$

Assumption made by Logistic Regression :- Classes are almost / perfectly linearly seperable.

Given {+ve points, -ve points}, task is to find w & b, ie, find π that best seperates +ve points from -ve points.



Let $y_i \in \{+1, -1\}$ instead of $\{1, 0\}$

$$d_i = \frac{w^T x_i}{\|w\|} = w^T x_i$$

$\|w\| \rightarrow 1$ (unit vector. Normal)

$$d_j = \frac{w^T x_j}{\|w\|} = w^T x_j.$$

+ve points & w pointing towards the same direction.

$\Rightarrow w^T \cdot x_i > 0$

$\Rightarrow w^T \cdot x_j < 0$

if $w^T x_i > 0$, then $y_i = +1$

else if $w^T x_i < 0$, then $y_i = -1$

→ decision surface of LR : plane

→ if $\underbrace{y_i}_{+1} * \underbrace{w^T x_i}_{\text{basis of model.}} > 0$, w is correctly classifying the point

(+) (+)

(−) (−)

both the & -ve points, if $y_i * w^T x_i > 0$, LR is correctly classifying the point.

→ if $y_i * w^T x_i < 0$, w is uncorrectly classifying the point

(+) (−)

(−) (+)

Misclassified point

→ For classifier to be good, we want as many points to have $y_i w^T x_i > 0$.

$$\max_{w} \sum_{i=1}^{n} y_i w^T x_i$$

fixed terms.

variable.

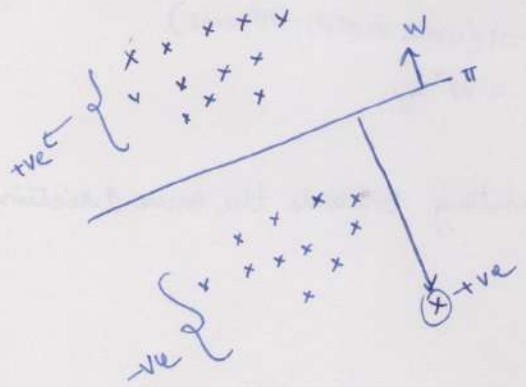
→ change / vary w / find optimal w that maximizes $\sum_{i=1}^{n} y_i w^T x_i$

$$w^* = \underset{w}{\operatorname{argmax}} \left(\sum_{i=1}^{n} y_i w^T x_i \right)$$ ← optimization problem.

optimal w

→ IRL we never know if data is linearly seperable. We apply Logistic Regression & compare it's performance with random model (gives class randomly) and choose it if it performs better.

Squashing & Sigmoid function (8.2):-



$$w^* = \operatorname{argmax} \sum_{i=1}^{n} \underbrace{y_i w^T x_i}_{\text{signed distance.}}$$

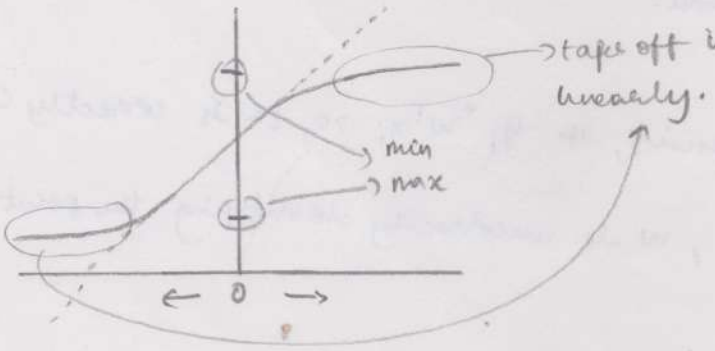
→ Distant outliers reduce the signed distance by a lot. As a result, incorrectly classifying hyperplanes have ~~lower~~ higher signed distances.

One single outlier is impacting model.

max sum of signed dist is not outliers prone.

To minimize this we use squashing.

Squashing :- Idea is if signed dist is small, use it as it is
if signed dist is large, make it small.



→ tapes off instead of letting it increase/decrease linearly.

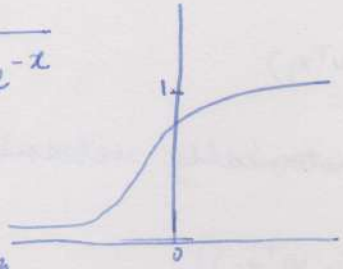
→ Instead of $\underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} y_i w^T x_i$,

$\underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} f(y_i w^T x_i)$, f does not let $w^T x_i$ grow linearly After a certain Value, it keeps that distance.

one such function is called sigmoid function ($\sigma(x)$)

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

max = 1
min = 0
$\sigma(0) = 0.5$



Reason behind choosing sigmoid is, it has a nice probabilistic interpretation & is differentiable

If point lies on hyperplane, ie., $w^T x_i = 0$, $P(y=1) = 0.5$ &
$\sigma(0) = 0.5$

→ instead of maximizing sum of signed distances, we maximize sum of transformed signed distances.

$$\therefore w^* = \underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} \sigma(y_i w^T x_i)$$

$$= \underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} \frac{1}{1+e^{-(y_i w^T x_i)}}$$

less impacted than outliers.

→ sigmoid function basically gives the probability of $P(y=1)$

Optimization function (Q3) :-

Monotonic function :- $g(x)$ and if x increases, $g(x)$ also increases.

if $x_1 < x_2$, then $g(x_1) > g(x_2)$, then $g(x)$ is said to be monotonically increasing function.

ex:- $f(x) = x^2$

$x^* = \underset{x}{\operatorname{argmin}} f(x)$, x^2 is minimum at 0

$\Rightarrow x^* = 0$

$g(x) = \log(x)$

$x' = \underset{x}{\operatorname{argmin}}\, g(f(x)) = \underset{x}{\operatorname{argmin}} \log(x^2)$

$x' = x^*$ since $g(x)$ is monotonically increasing function.

(46)

→ $\underset{x}{\operatorname{argmin}} f(x) = \underset{x}{\operatorname{argmin}}\, g(f(x))$

and

$\underset{x}{\operatorname{argmax}} f(x) = \operatorname{argmax}\, g(f(x))$

if

$g(x)$ is a monotonic function.

→ $w^* = \underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} \frac{1}{1+e^{-(y_i w^T x_i)}}$

$g(x) = \log(x)$ [monotonically increasing function]

$w^* = \underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} \log\left(\sigma\left(y_i w^T x_i\right)\right)$

$w^* = \underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} \log\left(\frac{1}{1+\exp(-y_i w^T x_i)}\right)$

since $\log(1/x) = -\log(x)$

$\Rightarrow w^* = \underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} -\log\left(1+\exp(-y_i w^T x_i)\right)$

$\underset{x}{\operatorname{argmax}} f(x) = \underset{x}{\operatorname{argmin}} -f(x)$

(or)

$\underset{x}{\operatorname{argmin}} -f(x) = \underset{x}{\operatorname{argmax}} f(x)$

$\therefore w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} \log\left(1+\exp\left(\overbrace{-y_i}^{+1 \text{ or } -1} w^T x_i\right)\right)$ } → this formula is derived from geometric approach & ~~log~~ loss function approach.

Alternative formula is

$w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} -y_i \log P_i - (1-y_i) \log(1-P_i)$ → +1 or 0.

$P_i = \sigma(w^T x_i)$.

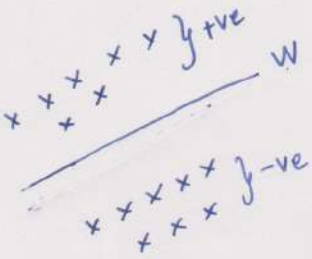
Derived from probabilistic method.

→ log is used to make objective function convex so that it is easy to optimise. convex functions have only one minima. The local minima is same as that of global minima.

→ sigmoid squishes the signed values b/w 0 & 1. This is not the intention. The goal is to control the exponiness at higher values. Log does this. So we make use of montonic property.

Weight Vector (8.4) :- (34.52)

the optimal w^* ~~app~~ obtained from loss function = Weight Vector = W



$$W = \langle \underset{f_1}{w_1}, \underset{f_2}{w_2}, w_3, w_4, \ldots \underset{f_d}{w_d} \rangle$$

$w^* x_q$ = +ve $\longrightarrow$ belongs to positive class

$w^* x_q$ = -ve $\longrightarrow$ belongs to negative class

case ① :- If $\underset{f_i}{w_i}$ = +ve , and if $x_{q_i} \uparrow$

$$\Rightarrow (w_i x_{q_i}) \uparrow$$

$$\Rightarrow \sum_{i=1}^{d} (w_i x_{q_i}) \uparrow$$

$$\Rightarrow \sigma(w^T x_q) \uparrow \longrightarrow \text{monotonically increasing fn.}$$

$$\Rightarrow P(y_q = +1) \uparrow$$

case ② :- if w_i = -ve , and if $x_{q_i} \uparrow$,

$$\Rightarrow (w_i x_{q_i}) \downarrow$$

$$\Rightarrow \sum_{i=1}^{d} (w_i x_{q_i}) \downarrow$$

$$\Rightarrow \sigma(w^T x_q) \downarrow$$

$$\Rightarrow P(y_q = -1) \uparrow$$

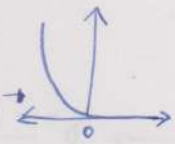
$\therefore$ If weight (w_i) corresponding to feature i ~~increases~~ is positive and gives a query point if its ith feature (x_{q_i}) increases, the probability that it is of +ve class increases.

L2 Regularization (Overfitting & Underfitting) (8.5) :-

$$w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} \log(1 + \exp(-y_i w^T x_i))$$

Let $z_i = y_i w^T x_i$

$$\Rightarrow w^* = \underset{w}{\operatorname{argmin}} \underbrace{\sum_{i=1}^{n} \underbrace{\log(1 + \underbrace{\exp(-z_i)}_{\geq 0 \text{ for all } z_i})}_{\geq 0}}_{\geq 0}$$



→ The minimal possible value is 0. (argmin) and it occurs when $z_i \to \infty \ \forall i$

→ $w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} \log(1+\exp(-z_i))$

if z_i = +ve, $z_i \to +\infty$

then $\exp(-z_i) \to 0$

$\log(1+\exp(-z_i)) \to 0$

→ $z_i = \underbrace{y_i}_{\text{given}} w^T \underbrace{x_i}_{\text{given}}$. For z_i to $\to \infty$, we must modify w in such a way that each $z_i \to +\infty$

If z_i is +ve, then point is correctly classified.

If $z_i \to +\infty$, we get argmin.

→ If we pick w such that

ⓐ All points are correctly classified

ⓑ $z_i \to +\infty$,

then we get best w.

→ There might be outliers. This leads to overfitting (Perfect job with training data & not good with test data)

and $w_i \to +\infty$ / $w_i \to -\infty$

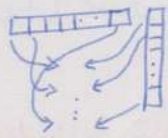
To avoid this we modify optimization problem with <u>regularization</u>.

$$w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} \log\left(1+\left(\exp\left(-y_i w^T x_i\right)\right)\right) + \underbrace{\lambda w^T w}$$

→ We initially made assumption that w is normal. It's not made anymore

$$\lambda \sum_{j=1}^{d} w_j^2 = \lambda \underbrace{\|w\|^2}$$

Square of L_2 norm of w.



$$\Rightarrow w^* = \underset{w}{\operatorname{argmin}} \left[\underbrace{\sum_{i=1}^{n} \log\left(1+\exp\left(-y_i w^T x_i\right)\right)}_{\text{loss term}} + \underbrace{\lambda \|w\|^2}_{\text{regularization term}} \right]$$

This won't let $w_j \to \infty$ / $w_j \to -\infty$ even. If loss term allows it, regularization term won't and it won't lead to minima.

$w_j \to +\infty$ / $-\infty$ ⟶ "0" ⟶ "Very large"

λ → Hyperparameter. if $\lambda = 0$, ⇒ No regularization ⇒ overfit

if λ = Very large ⇒ influence of loss term reduces. ⇒ Training data not being used = underfitting.

→ common theme in many machine learning models

$$= \min\left(\text{loss function over training data} + \text{regularization function}\right)$$

→ right λ found through cross validation.

→ w ignored for optimization purpose from signed distance.

L1 Regularization (8.6) (34.86) :-

$$w^* = \underset{w}{\operatorname{argmin}} \underbrace{\sum_{i=1}^{n} \log\left(1+\exp(-y_i w^T x_i)\right)}_{\text{logistic loss}} + \underbrace{\lambda \|w\|^2}_{L_2 \text{ Regularizer}}$$

are there alternatives to L2 Regularization? Popular alt is L1.

In L1 we use $\|w\|_1$ instead of $\|w\|_2^2$

$$\|w\|_1 = \sum_{i=1}^{d} |w_i|$$

$$\Rightarrow w^* = \underset{w}{\operatorname{argmin}} \left(\text{logistic loss of training data}\right) + \underbrace{\lambda \|w\|_1}_{L_1 \text{ reg.}}$$

→ hyperparameter.

$\|w\|_1$ will prevent $w_i \to -\infty$

λ will prevent $w_i \to \infty$

→ Major advantage L1 has over L2 is <u>sparsity</u>

<u>Sparsity</u>:- $w = \langle w_1, w_2, w_3, \ldots, w_d \rangle$

$\underbrace{\text{Solution to logistic regression}}_{w^*}$ is said to be sparse if many w_i's are zero.

→ From many experiments, we can conclude if we use L1 regularization, all the unimportant or less important features become zero. If L2 regularization is used, less important features become small values but not necessarily zero.

→ Another alternative to both L1 & L2 is <u>elastic net</u>.

$$w^* = \underset{w}{\operatorname{argmin}} \left(\text{logistic loss}\right) + \lambda_1 \|w\|_1 + \lambda_2 \|w\|_2^2$$

two hyperparameters.

→ In cases where there are correlated features, L1 only selects one of them. This is sometimes bad (gene dataset example)

→ For high dimensional cases ($n << d$), L1 can select atmost n features because of the convex optimization problem it tries to solve.

→ Even for $n >> d$ problems, Ridge Regularization (Tikhonov Regularization) has better prediction power than L1.

As a result of above three issues, it is better to choose Elastic net because it is a better feature selector than L1.

(50) LASSO Model = Linear Model with L1 regularization applied.

→ We can use L1 as feature selection as well.

→ LASSO → Dealing with outliers & to get sparse matrix.

RIDGE → To keep correlated features.

Elastic Net → anywhere.

Probabilistic Interpretation of GNB (8.7) (34.96) :-

→ two assumptions are made just like in Naive Bayes

(i) features are real valued → Gaussian distribution.

(ii) $y_i = +1$ or 0 → Bernoulli RV (ex:- coin toss)

→ Logistic Regression = Gaussian Naive Bayes + Bernoulli

$(P(x_i|y_i))$ $(y_i \sim \text{Bernoulli})$

→ In geometric interpretation, $y_i \sim +1/-1$. In probabilistic its $+1/0$.

Probabilistic :- $$w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} -y_i \log p_i - (1-y_i)\log(1-p_i) + \text{reg}$$

$$p_i = \sigma(w^T x_i) = \frac{1}{1+\exp(w^T x_i)}$$

Loss Minimization Interpretation (8.8) (35.14):-

Let $+1$: incorrectly classified points. } In ideal loss function.
0 : correctly classified points. }

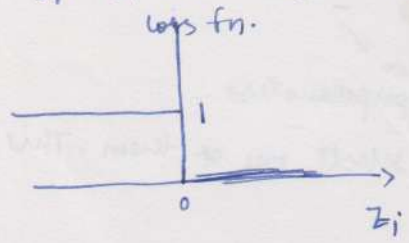
$$w^* = \underset{(w)}{\operatorname{argmin}} \left(\text{number of incorrectly classified points}\right)$$

[∵ We want to minimize number of incorrectly classified points]

Let $z_i = y_i w^T x_i$

if z_i is +ve, ⇒ point is correctly classified ⇒ loss fn = 0

if z_i is −ve, ⇒ point is incorrectly classified → loss fn = 1



→ This type of loss fn is called 0-1 loss function.

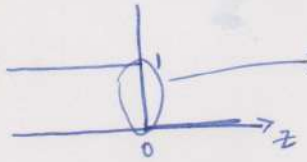
$$w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} \text{0-1 loss}(x_i, y_i, w)$$

$$\text{0-1 loss}(x_i, y_i, w) = \begin{cases} 1 & \text{if } z_i < 0 \\ 0 & \text{if } z_i > 0 \end{cases}$$

$$z_i = x_i w^T y_i$$

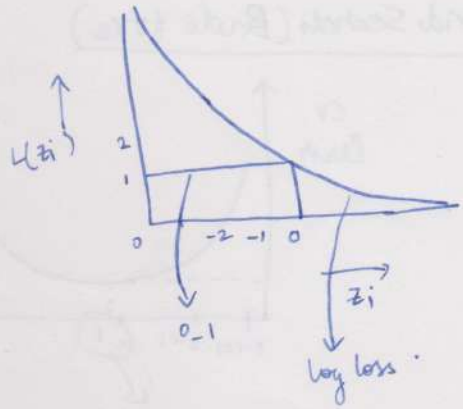
→ We use differentiation to solve optimization problems. So loss functions need to be differentiable.

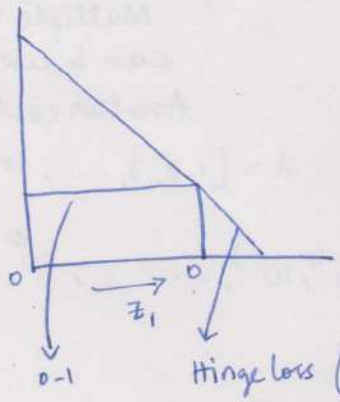
For $f(x)$ to be differentiable, it needs to be continuous.



→ 0_1 loss function is not defined at 0. Not continuous.
∴ Not differentiable.

$\underbrace{\text{logistic loss}}_{\text{differentiable.}} = \log(1+\exp(-z_i))$





Loss Minimization Interpretation

loss fn → log loss → Logistic Regression
Hinge loss → SVM
exponential loss → Ada Boost
Squared loss → Linear Regression.
⋮

→ Loss function is basically a function that is minimized along with the hyperparameters of the model.

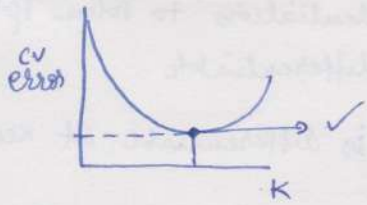
ex:- In ~~linear~~ logistic regression it tries to find balance b/w best hyperplane & good loss value.

Hyperparameter Search (Grid Search & Random Search) (8.9) (35.32):-

λ = hyperparameter $\quad \lambda = 0 \rightarrow$ overfit
$= \infty \rightarrow$ underfit.

How to find best λ?
Similar to K in KNN, ~~&~~ α in NB

(52) → We used CV error for the other models



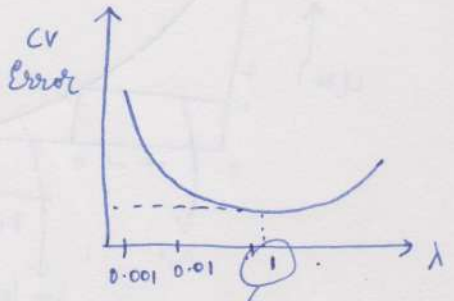
But problem is

K in KNN is integer

λ in LR is real number. So infinite possibilities.

One technique is grid search.

Grid Search (Brute force)



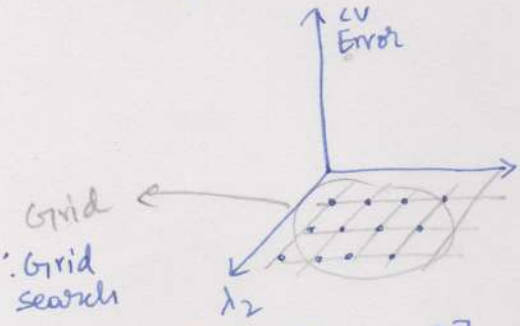
roughly around 1, good λ.

Select $\lambda = [0.001, 0.01, 0.1, 1, \quad]$ (×10, ×10, ...)

One way. Multiple ways can be used. Another could be $\lambda = [1, 2, 3, \ldots, 10]$

→ Typically wide window used to search $\lambda = [10^{-4}, 10^{-3}, \ldots, 10^{4}]$

Elastic Net Hyperparameters :- $\lambda_1 \|W\|_1 + \lambda_2 \|W_2\|^2$



$\lambda_1 = \underbrace{[10^{-3}, 10^{-2}, 10^{-1}, \ldots, 10^{3}]}_{m_1}$

$\underbrace{[10^{-3}, 10^{-2}, \ldots, 10^{3}]}_{m_2}$

→ Each pair is chosen & CV error is calculated.

$m_1 \times m_2$ possibilities are calculated.
combinations.

→ Problem with grid search

λ_1 : 1 hyperparameter : m_1 time traind.

λ_1, λ_2 : 2 " : $m_1 * m_2$

$\lambda_1, \lambda_2, \lambda_3$: 3 " : $m_1 * m_2 * m_3$

⋮

As # hyperparameters increases, # of times model needs to be trained increase exponentially.

Deep learning has lots of hyperparameters unlike logistic regression which has only 2.

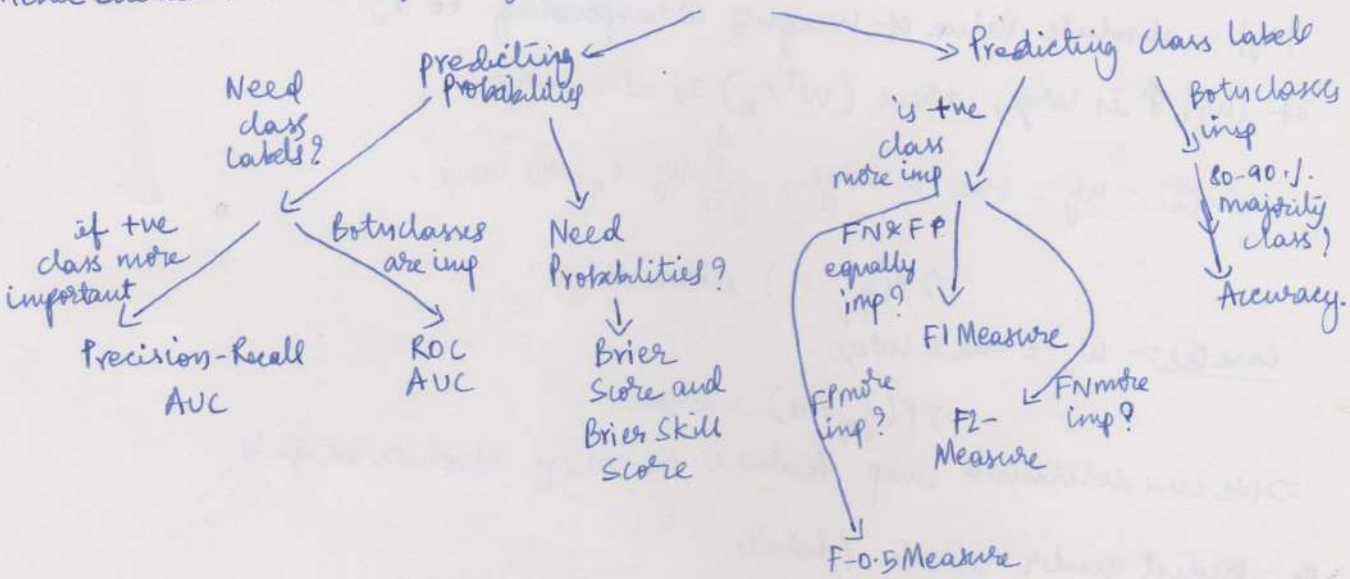
→ To overcome this, we use Random Search.

Random Search: ✱ Let say $\lambda \in [10^{-4}, 10^{4}]$

We randomly pick values in the given interval & try them.

→ Random Search is almost as good as grid search when # of hyperparam is large.

→ Model evaluation techniques & general conditions to use them under



Precision-Recall AUC = area under Curve b/w precision & recall

Column Standardization (8.10) (35.83):-

→ Mandatory to perform column standardization as diff scales in diff features will affect dist from hyperplane.

→ column standardization is aka mean centering & scaling.

→ standardization is used to achieve convergence faster in SGD models. ie, it is required for linear regression, KNN etc, but not required for logistic regression as sigmoid function squashes vals b/w 0 & 1. But sigmoid fn does not make mean 0 & ~~mode~~ variance 1.

→ We basically try to get all means as 0 & all variances as 1 for all features.

(54) Feature importance & Model Interpretability (8.11)

$f_1, f_2, \ldots, f_j, \ldots f_d$

$w \rightarrow w_1, w_2, \ldots w_j, \ldots w_d$

assumption :- All features are independent

if assumption is true, we can get important features.

$|w_j|$ = absolute value of weights corresponding to f_j.

if $|w_j| \uparrow$ is large, then $(w^T x_q)$ is also large.

case ① :- w_j = +ve & large, $\sum_{j=1}^{d} w_j \cdot x_{q_j}$ is large.

$\Rightarrow P(y_q = +1)$ increasy.

case ② :- w_j = -ve & large.

$\Rightarrow P(y_q = -1)$ increases.

$\Rightarrow$ we can determine imp features by using absolute weights.

eg :- predict gender $\rightarrow$ male / female
(+1) (-1)

hl = hair length. as $w_{hl} \uparrow$, $P(y_q = -1) \uparrow$

$\Rightarrow$ from case ② :- w_{hl} will be large negative weight value.

because imp to predict

-1 is female.

ht = height, $w_{ht} \uparrow$, $P(y_q = +1) \uparrow$

$\Rightarrow$ But height is not that diff compared to hair length.

$\therefore$ w_{ht} will have medium positive weight value

Model Interpretability :-

$\rightarrow$ We can use top features (large absolute value) to justify predictions.

like using top 5 features

Colinearity (or) MultiCollinearity (8.12) (36.01):-

$|w_j|$ can be used for feature imp iff features are independent.

<u>collinearity</u> :- f_i, f_j such that $f_i = \alpha f_j + \beta$ then f_i, f_j are collinear.

Multicollinearity :- f_1, f_2, f_3, f_4.

$$f_1 = \alpha_1 + \alpha_2 f_2 + \alpha_3 f_3 + \alpha_4 f_4 \quad , f_1, f_2, f_3, f_4 \text{ are multicollinear}$$

why cant we use $|w_j|$ for collinear?

ex:- $D = \langle x_i, y_i \rangle_{i=1}^{n}$

$$w^* = \underset{f_1, f_2\ f_3}{\langle 1, 2, 3 \rangle} \quad ; \quad x_q = \langle x_{q_1}, x_{q_2}, x_{q_3} \rangle$$

$$w^{*T} x_q = x_{q_1} + 2x_{q_2} + 3x_{q_3}$$

if $f_2 = 2f_1$ ie., f_1 & f_2 are collinear.

$$\Rightarrow w^{*T} x_q = x_{q_1} + 4x_{q_1} + 3x_{q_3} = 5x_{q_1} + 3x_{q_3} = \underbrace{\langle 5, 0, 3 \rangle}_{\tilde{w}^T}$$

w^* & $\tilde{w}$ give same classifiers

But

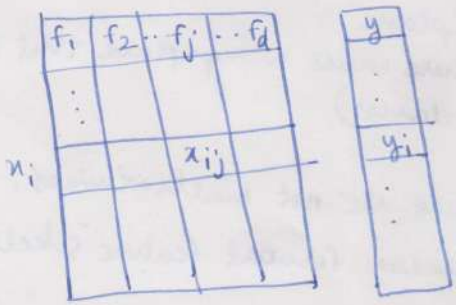
$\langle 1, 2, \underline{3} \rangle$ $\quad$ $\langle \underline{5}, \underline{0}, 3 \rangle$

$\downarrow$ most imp feature $\quad$ $\downarrow$ most imp feature $\quad$ $\rightarrow$ not imp at all.

if features are collinear, weight vecs can change arbitrarily

$\Rightarrow$ ~~feat~~ $|w_j|$ can't be used as feature importance

$\rightarrow$ Pertubation technique can be used to find if features are multicollinear



$\forall\ x_{ij}$ add some error/noise

ie., $x_{ij} + \underbrace{\epsilon}_{N(0, 0.1)}$

random var small variance

Before noise $w^* = \langle w_1, w_2, \ldots w_d \rangle$

After perturbation $\tilde{w} = \langle w_1, w_2, \ldots w_d \rangle$

if w^* & $\tilde{w}$ differs drastically, then features are collinear.

$\Rightarrow |w_j|$ can't be used as feat imp.

→ for collinear features, forward selection can be used.

→ common practice is to identify highly collinear features, include one & eliminate the others. Model will work & feature importance can also be applied.

Train & Runtime complexity (8.13):-

Train LR :- Solving W^*

$O(nd)$

Runtime :- This is the most important metric.

Once we get W^*, we just multiply its' transpose with x_q.

$$W^T x_q > 0 \rightarrow +ve$$
$$< 0 \rightarrow -ve.$$

Space:- $O(d)$

time:- $O(d)$

If d is small, logistic regression is very good for low latency applications.

LR used a lot by internet companies.

If d is large, we can use L1 regularization (w_j's corresponding to less imp features become 0)

$\lambda \uparrow$, sparsity $\uparrow$

In addition to bias-variance tradeoff, we also adjust λ to get fastest LR possible.

latency requirement

$\lambda \uparrow \rightarrow$ bias $\uparrow$, latency $\downarrow$

$\lambda \uparrow \rightarrow$ sparsity $\uparrow$ → latency $\downarrow$

Real World Cases (8.14):-

decision surface :- linear decision surface/hyperplane

assumption :- data is linearly seperable (i.e., there exists a hyperplane that seperates both classes)

(or) almost linearly seperable.

Feature importance/interpretability :- $|w_j|$ if features are not multicollinear.

If multicollinear forward feature selection etc.,

Imbalanced data :- upsampling/downsampling.

Outliers :- less impact because of $\sigma(x)$. Another technique used is

calculate W^* from D_{train}. Calculate distance from π to x_i.

Remove points which are far from π from the dataset.

D'_{train} new dataset. Calculate $\tilde{W}^*$ final solution.

Missing Values :- std imputation techniques (mean, median, mode)

Multiclass :- typically one vs rest is used.
extensions like max { cut models also exist.
soft max classifiers
multinomial LR }

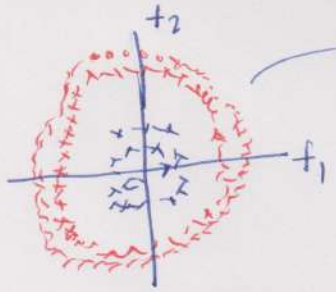
similarity Matrix given :- Regular LR wont take it but Kernel LR will.

Best case :- Almost linearly seperable data
Low latency Requirement.
Very fast to train.

Large dimensionality :- d large => chance data is linearly seperable is high.
It is easy to overfit models in high dimension. So using basic model like LR is beneficial to avoid overfitting.

Non linearly Seperable Data & feature Engineering (P. 15) :-

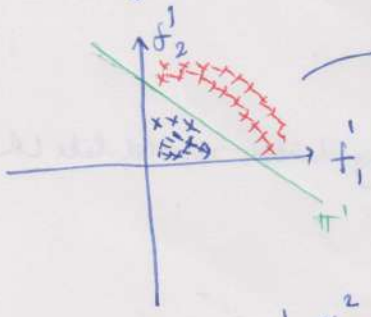
Data is not linearly seperable.



→ drawing a plane cannot seprate +ve & -ve points.

eq of circle :- $f_1^2 + f_2^2 = r$
Not linear in f_1 & f_2
But linear in f_1^2, f_2^2.
∴ New features are f_1^2 & f_2^1

feature transformation



→ LR can be used.

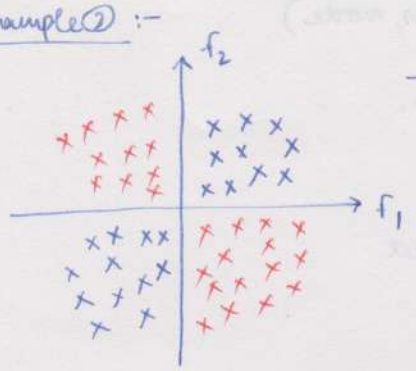
Query :- $x_q = \langle x_{q_1}, x_{q_2} \rangle$
$\downarrow$ FT / FE
$x_q^1 = \langle x_{q_1}^1, x_{q_2}^1 \rangle \xrightarrow[\pi']{LR} y_q$
$\downarrow$
$x_{q_1}^2, x_{q_2}^2$

$f_1^1 = f_1^2$ $\quad x_{i1}^1 = x_{i1}^2$
$f_2^1 = f_2^2$ $\quad \vdots$

→ Most important aspect of applied ML/AI :

1. Feature Engineering.
2. Bias-Variance tradeoff
3. Data Analysis.

example ② :-

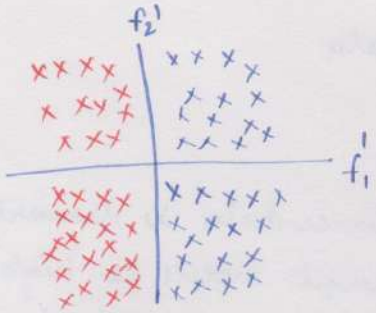


→ XOR dataset

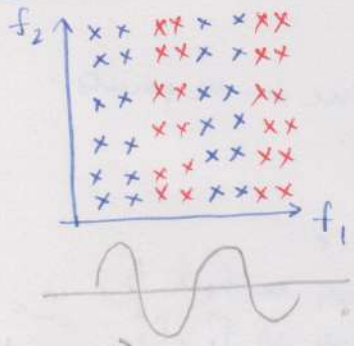
FE

$f_1' = f_1 * f_2$

$f_2' = f_2$



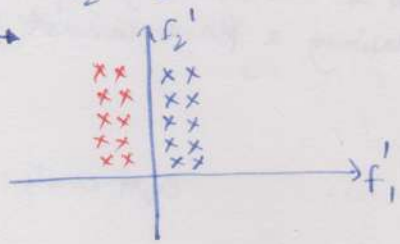
example ③ :-



(sin wave)

$f_1' = \sin(f_1)$

$f_2' = f_2$



→ typical transforms for real valued features

① $\underbrace{f_1 f_2,\ f_1^2,\ f_2^2,\ \ldots}_{\text{polynomial features}}$

② trigonometric features (sin, cos, ...)

③ Boolean features. (OR, AND, XOR)

④ $\log(f_1),\ e^{f_1},\ \ldots$

→ can find if data is linearly seperable or not by training linear model like LR & checking performance.

Code Sample - Logistic Regression (8.16):-

model.coef_ :- weights of every feature in fitted data

model.predict_prob(X) :- prob of every datapoint of X belonging to a particular class

model.decision_function(X) :- how confident is model on about which class every datapoint belongs to.

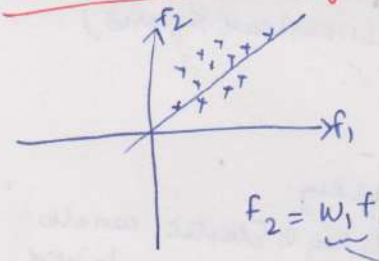
Generalized Linear Model (8.17):-

extension to LR obtained from probabilistic interpretation of it.

GLMs:

- LR :- GNB + Bernoulli
- GNB → Multinomial NB + Multinomial distrib = Multinomial LR aka Softmax classifier → useful for multiclass classfn
- GNB + $P(y|x) \sim N(\mu, \sigma^2)$ = Linear Regression
- GNB + Poisson distribution (count RV useful) = Poisson Regression → for predicting counts

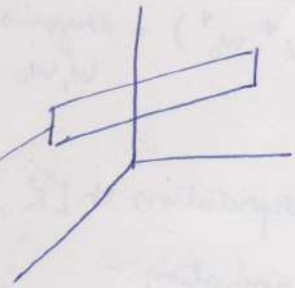
Geometric Intuition of Linear Regression (9.1):-



we try to find a line/plane that fits given data as well as possible

$f_2 = w_1 f_1 + w_0$

objective is to find these two



$y_i = w_1 * x_{i1} + w_2 * x_{i2} + w_0$

$= w^T x_i + w_0$

error for x_i = $y_i - \hat{y}_i$

y_i → actual value

$\hat{y}_i$ → predicted val / point on the line

to find multicollinearity, we can use VIF (variance inflation factor)

$$R_1^2 = \delta_0 + \delta_1 X_{2i} + \delta_2 X_{3i} + \cdots + \delta_{p-1} X_{pi} + \varepsilon$$
$$R_2^2 = \delta_0 + \delta_1 X_{1i} + \delta_2 X_{3i} + \cdots + \delta_{p-1} X_{pi} + \varepsilon$$
$$\vdots$$

$$VIF = \frac{1}{1 - R_j^2}$$ if ≥ 5, then collinear.

→ MAE (Mean Absolute Error) :- Average of errors in set of predictions without considering their directions.

MSE (Mean Squared Error) :- Tells how close regression line is to the set of points.

Mathematical Formulation (9.2) :-

→ errors could be +ve/-ve. Best fit minimizes this error.

$$(w^*, w_0^*) = \underset{w, w_0}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$\hat{y}_i = f(x_i) = w^T x_i + x_0 \longrightarrow \text{Square Loss}$$

$$= \underset{w, w_0}{\operatorname{argmin}} \sum_{i=1}^{n} \{y_i - (w^T x_i + x_0)\}^2$$

→ Linear Regression aka OLS (Ordinary Least Squares), LLS (Linear Least Squares)

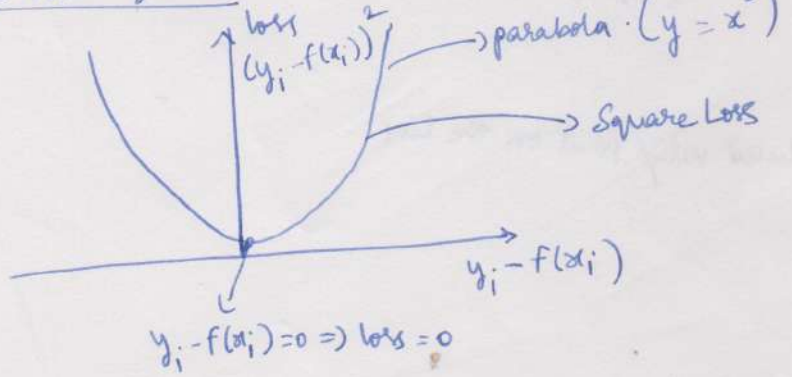
Regularization :-

$$(w^*, w_0^*) = \underset{w, w_0}{\operatorname{argmin}} \sum_{i=1}^{n} \{y_i - (w^T x_i + w_0)\}^2 + \lambda \|w\|_2^2$$

this forces w_i to be small. → L2 Reg. L1 reg & Elastic can also be used

→ prob interpretation of LR, $P(y_i | x_i) = N(\mu, \sigma^2)$

→ Loss-Minimization :-



→ we take square instead of abs, in loss because abs is not differentiable at 0.

Real World cases (9.3):-

Imbalanced data :- upsampling, downsampling. (No imbalanced Data cuz no class label)

feature imp & interpretability :- if ftr are not multicollinear, $|w_j|$ can be used.

feature Engineering & feature transforms :- same as that of logistic regression

regularization :- L_1 : sparsity

$\lambda \uparrow \rightarrow$ sparsity $\uparrow$

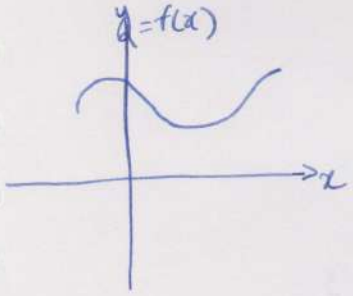
outliers :- remove far away points after calculating w^* & w_0^* & calculate w^*, w_0^* again on new dataset.

This technique is called RANSAC (Random Sampling Consensus)

solving Optimization Problems : Differentiation (10.1):-

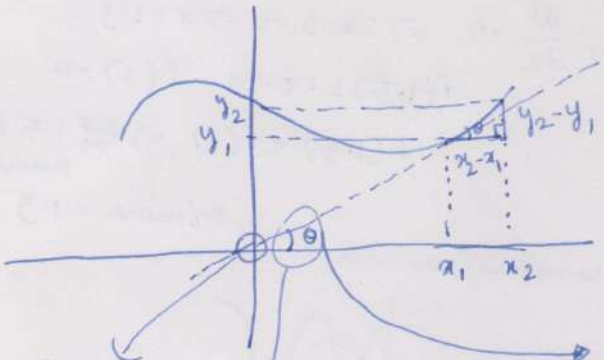
→ differentiation :- single value & vector differentiation is used.

maxima & minima are also used.



$$\frac{dy}{dx} = \frac{df}{dx} = f' = y'$$

$\frac{dy}{dx}$ intuitively means rate of change of y as x changes



$$\frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x} = \tan\theta$$

$$= \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x}$$

doesn't necessarily have to touch origin

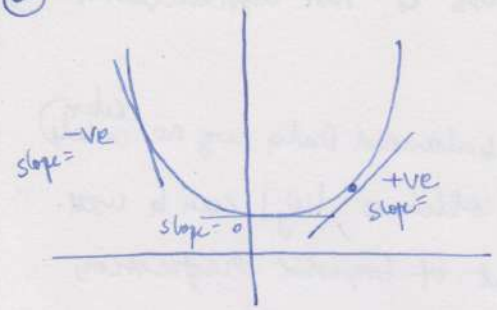
as Δx becomes smaller & smaller it becomes a tangent to $f(x)$

$$\tan(\theta) = \frac{dy}{dx}$$

geometrically

$\frac{dy}{dx}$ = slope of tangent of $f(x)$

$\left[\frac{dy}{dx}\right]_{x_1}$ = slope of tangent to $f(x)$ at x_1



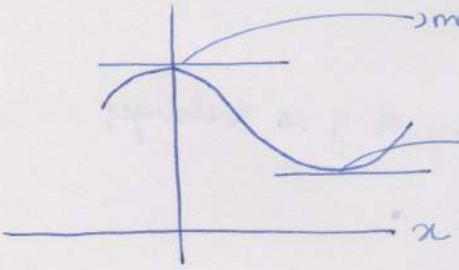
slope = tan(θ)

→ $\frac{d}{dx}(x^n) = nx^{n-1}$ $\quad \frac{d}{dx}(cx^n) = cnx^{n-1}$ $\quad \frac{d}{dx}(f(x)+g(x)) = \frac{d}{dx}(f(x)) + \frac{d}{dx}(g(x))$

$\frac{d}{dx}(c) = 0$ $\quad \frac{d}{dx}(\log(x)) = 1/x$

<u>chain rule</u>:- $\frac{d}{dx}(f(g(x))) = \frac{df}{dg} \cdot \frac{dg}{dx}$ } ML very imp

Maxima & Minima :- (10.2)



→ slope = 0 = $\frac{df}{dx}$

Let $f(x) = x^2 - 3x + 2$

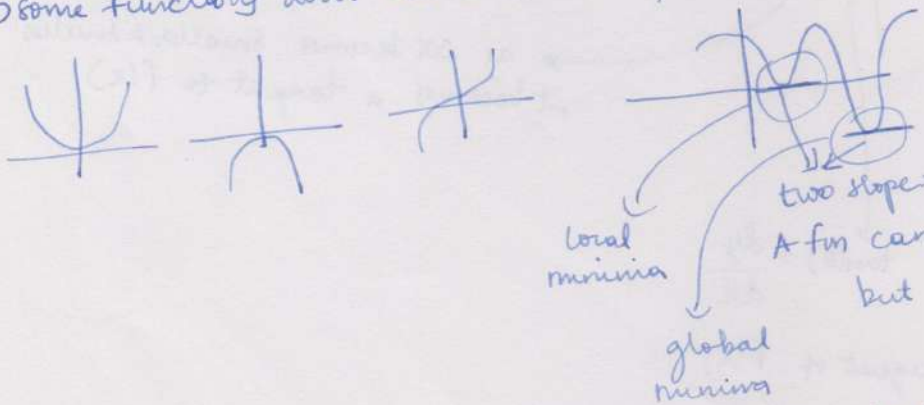
to get minima/maxima → $\frac{df}{dx} = 0 \Rightarrow 2x - 3 = 0 \Rightarrow x = 1.5$

$f(1.5) = -0.25 \quad f(1) = 0$

$f(1.5) < f(1) \Rightarrow$ ~~f(x)~~ 1.5 can't be maxima.

$\therefore$ minima = 1.5

→ Some functions dont have maxima/minima



two slope = 0 = two minima.

A fn can have multiple local minima but only one global maxima

→ putting $\frac{df}{dx} = 0$ is not always the best way to find minima & maxima as equations get tricky/complex. So we use techniques like gradient descent.

Vector Calculus (10.3):-

→ so far we assumed x to be scalar. But in ML it is generally a vector

$$\frac{df}{dx} = \nabla_x f$$

x → vector

∇ → grad (called del) → aka Jacobian.

x → vector.

$$\nabla_x f = \begin{bmatrix} \frac{df}{dx_1} \\ \frac{df}{dx_2} \\ \vdots \\ \frac{df}{dx_d} \end{bmatrix} \longrightarrow \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{bmatrix} \in \mathbb{R}^d$$

→ partial diff is used since we are differentiating partially.

ex:- $f(x) = y = a^T x = \sum_{i=1}^{d} a_i x_i = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n$

$$\nabla_x f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_d \end{bmatrix} = a \qquad \frac{\partial f}{\partial x_1} = \frac{\partial}{\partial x_1}(a x_1) = a$$

$\therefore \nabla_x (a^T x) = a$ $\left(\text{like } \frac{d}{dx}(ax) = a\right)$

Logistic Regression loss fn, $L = \sum_{i=1}^{n} \log\left(1 + \exp(-y_i W^T x_i)\right) + \underbrace{\lambda W^T W}_{\text{similar to } \lambda W^2}$

x_i, y_i → constants in train set.

W → variable.

$$\nabla_W L = \underbrace{\frac{(-y_i x_i)(\exp(-y_i W^T x_i))}{1 + \exp(-y_i W^T x_i)} + 2\lambda W = 0}_{\text{hard to solve as } W \text{ is a vector. So we use Gradient Descent.}}$$

Gradient Descent (10.4):-

→ it is an iterative algorithm.

x_0 → first guess of x^*

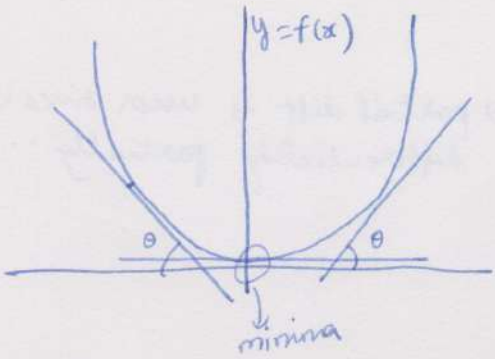
x_1 → iteration 1

$\vdots$

x_k → iteration k.

↳ will be close to x^*

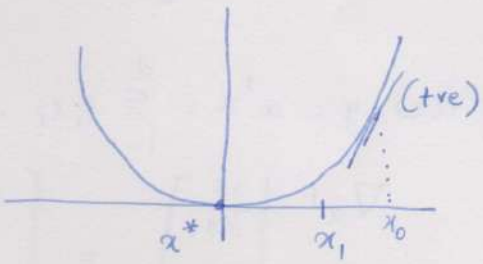
minimizing $f(x)$ = maximizing $(-f(x))$
& vice versa



① slope sign changes at minima.
② as we move closer to x^*, slope reduces.

Gradient Descent :-

① pick an initial point $= x_0$ randomly
initialization point



② $x_1 = x_0 - \gamma * \left[\frac{df}{dx}\right]_{x_0}$ } → update equation

learning rate / stepsize. Let it be 1.

$x_1 = x_0 - 1 * (+ve)$ → since slope at that pt = +ve

$\Rightarrow x_1 < x_0$ i.e., x_1 moves closer to x^*

③ $x_2 = x_1 - \gamma \left[\frac{df}{dx}\right]_{x_1}$

⋮

$x_{i+1} = x_i - \gamma \left[\frac{df}{dx}\right]_{x_i}$

if $(x_{k+1} - x_k)$ is very small, then terminate and $x^* = x_k$

Learning Rate (10.5) :-

→ If learning rate is kept constant, GD will jump b/w the same two points back & forth. This is called oscillation problem. One remedy to this is
→ change / reduce γ with each iteration.
γ becomes a function of iteration number.

Gradient Descent for Linear Regression (10.6) :-

$$L(w) = \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

$$\nabla_w L = \sum_{i=1}^{n} \{2(y_i - w^T x_i)(-x_i)\}$$

① pick a random vector $w_0 = \langle \dots \rangle$

② $w_1 = w_0 - \gamma * \sum_{i=1}^{n} (-2x_i)(y_i - w_0^T x_i)$

③ $w_2 = w_1 - \gamma * \sum_{i=1}^{n} (-2x_i)(y_i - w_1^T x_i)$

if $w_{k+1} - w_k$ is very small $w^* = w_k$

if n is large, computing this is very time consuming.
To overcome this we use stochastic Gradient Descent.

SGD Algorithm (10.7) :-

→ one of the most imp opt algo.

$$w^*_{SGD} :- \quad w_{j+1} = w_j - \gamma * \sum_{i=1}^{k} (-2x_i)(y_i - w_j^T x_i)$$

only difference
$1 \le k \le n$
k = random set of k points.

$w^*_{SGD} = w^*_{GD}$

→ set of k points picked in each iteration are different. Each set is non overlapping.

→ often times $k = 1$
sometimes $k > 1$ & $k << n$, → Batch SGD with batch size $= k$.

→ As k decreases, # of iterations to reach w^* also increases.

→ one epoch = model has seen all of our data once. ie, for gradient descent one iteration = one epoch.

→ Batch Gradient Descent, $k = n$
Stochastic Gradient Descent, $k = 1$
Mini Batch Gradient descent, $1 < k < n/2$

Constrained Optimization & PCA (10.8):-

PCA :- $\underbrace{\max_{U} \frac{1}{n} \sum_{i=1}^{n} (U^T x_i)^2}_{\text{objective function}}$ $\underbrace{\text{such that } U^T U = 1}_{\text{constraint}}$ $\approx \max_{x} f(x)$ such that $g(x) = c$

→ constrained Optimization

→ general constrained opt prob:- $\underbrace{\max_{x} f(x)}_{\text{obj fn}}$ such that $\underbrace{g(x) = c}_{\text{equality constraint}}$ $\underbrace{h(x) \geq d}_{\text{inequality constraint}}$

Lagrangian Multiplier:-

$$\begin{bmatrix} k(x) \leq e \\ \downarrow \\ -k(x) \geq e \end{bmatrix}$$

$\tilde{x} = \max_{x} f(x)$ such that $g(x) = c$ $h(x) \geq d$

$\Downarrow$

$$L(x, \lambda, \mu) = f(x) - \lambda \{g(x) - c\} - \mu \{d - h(x)\}$$

$\lambda, \mu \rightarrow$ Lagrangian Multipliers, $\lambda \geq 0$, $\mu \geq 0$

$$\underbrace{\frac{\partial L}{\partial x} = 0; \frac{\partial L}{\partial \lambda} = 0; \frac{\partial \lambda}{\partial \mu} = 0}_{\text{solving these will give}}$$

$\tilde{x}, \tilde{\lambda}, \tilde{\mu}$

→ PCA :- $\max_{U} \frac{1}{n} \sum_{i=1}^{n} (U^T x_i)^2$ such that $U^T U = 1$

$\Downarrow$

$\max_{U} U^T S U$ such that $U^T U = 1$ & $\underbrace{S = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^T}_{\text{covariance matrix of } X.}$

$$L(U, \lambda) = U^T S U - \lambda (U^T U - 1)$$

$$\frac{\partial L}{\partial U} = 0 \Rightarrow \frac{\partial}{\partial U} (U^T S U - \lambda U^T U - \lambda) = 0$$

$$\Rightarrow SU - \lambda U = 0 \quad \Rightarrow SU = \lambda U$$

→ definition of Eigen value

S: covariance matrix; U: vector.

U is the eigen vector of S & λ is the Eigen value of S.

Logistic Regression Revisited (10.10):-

$$w^* = \underset{w}{\operatorname{argmin}} \, (\text{Log-loss}) + \lambda w^T w$$

$$\left\{ w^* = \underset{w}{\operatorname{argmin}} \, (\text{Log-loss}) \text{ such that } w^T w = 1 \right.$$

constraint opt problem

Lagrange:- $L = (\text{Log-loss}) - \lambda (1 - w^T w)$

$$= \underbrace{(\text{Log loss} - \lambda)} + \lambda w^T w$$

similar

∴ Regularization can be thought of as an equality constraint.

Why L1 creates Sparsity (10.11):-

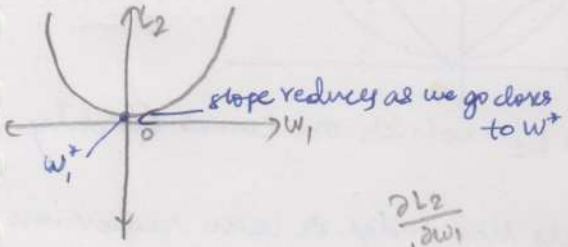
L_2:- $\min_w \text{loss} + \lambda \|w\|_2^2$ → loss is same. λ is const ← L_1:- $\min_w \text{loss} + \lambda \|w\|_1$

$\Rightarrow \min_w \|w\|_2^2$

$\Rightarrow \min_{w_1, w_2, \ldots w_d} (w_1^2 + w_2^2 + \cdots w_d^2)$

from w_1's perspective

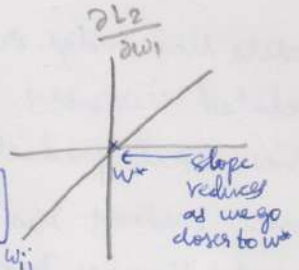
$\Rightarrow \min_{w_1} w_1^2 \longrightarrow L_2(w_1)$



$$\frac{\partial L_2}{\partial w_1} = 2 * w_1$$

$$(w_1)_{j+1} = (w_1)_j - \gamma \left[\frac{\partial L_2}{\partial w_1}\right]_{w_{1j}}$$

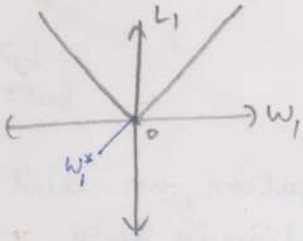
$$= (w_1)_j - \gamma \underline{(2 \times w_{1j})}$$



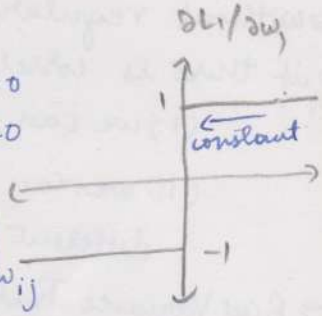
$\min_w \|w\|_1$

$\Rightarrow \min_{w_1, w_2, \ldots w_d} (|w_1| + |w_2| + \cdots + |w_d|)$

$\Rightarrow \min_{w_1} |w_1| \longrightarrow L_1(w_1)$



$$\frac{\partial L_1}{\partial w_1} = \begin{cases} +1 & \text{if } w_1 > 0 \\ -1 & \text{if } w_1 < 0 \end{cases}$$



$$(w_1)_{j+1} = (w_1)_j - \gamma \left[\frac{\partial L_1}{\partial w_1}\right]_{w_{1j}}$$

assume w_{1j} is +ve

$$(w_1)_{j+1} = (w_1)_j - \underline{\gamma (1)}$$

→ The derivative is decreasing as we go closer to w^* in L_2 but its constant in L_1

assume $\gamma = 0.01$, $w_{1j} = 0.05$ in L_2

$\Rightarrow (w_1)_{j+1} = (w_{1j}) - (0.01)(2 * 0.05)$

$= 0.05 - 0.001$

$(w_{1j+1}) = 0.05 - (0.01)(1) = 0.05 - 0.01$
$= 0.04$.

Rate of convergence is faster in L1. L2 doesnot change value of w_1 from one iteration to another that much but L1 is constantly decreasing.

→ SGD more computations than GD. SGD runs multiple epochs to converge, but GD even though uses all points, takes less epochs to converge.

→ Logistic Regression is not called Logistic Classifier because it predicts a continuous value (probability) rather than a discrete value.

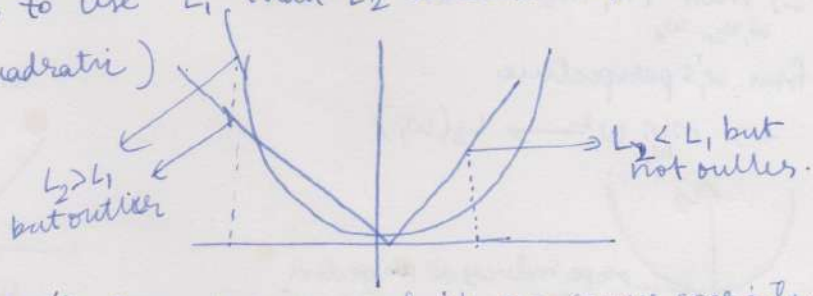
→ correlation :- indicates extent to which two or more variables move together. +ve correlation means two variables increase or decrease together. -ve correlation means if one increases the other decreases.

collinearity :- linear association b/w multiple predictors. Absolute correlation coefficient of > 0.7 among two or more indicates presence of multicollinearity.
Variance Inflation Factor (VIF) of 5 or 10 & above indicates multicollinearity problem.

VIF :- Measure of collinearity among predictor variables. It measures increase in variance of parameters if an additional variable is added.

Tolerance :- Reciprocal of VIF. Ranges b/w 0 & 1. High tolerance implies low multicollinearity & vice versa.

→ in case of outliers it is better to use L_1 than L_2 because L_1's contribution is lesser compared to L_2 (Quadratic)



→ Without regularization, we can't say L_1 or L_2 which one causes sparsity

→ if there is correlation in data
(i) we can use penalized regression models like ridge or lasso regression
(ii) we can add random noise to correlated variables so they become different from one another. But this affects prediction accuracy.

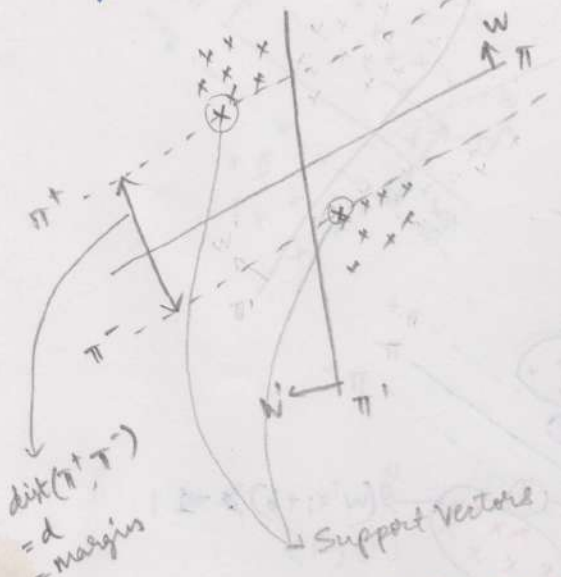
→ Bias Variance Tradeoff :- Predictive models have tradeoff b/w bias (how well a model fits) and variance (how much model changes based on change in inputs). Simpler models are stable (low variance) but don't get close to the truth (high bias). Complex models are prone to overfit (high variance) but expressive enough to get close to the truth (low bias).

→ GD theoretically minimizes the error function better than SGD but SGD converges much faster once dataset becomes large. Means GD is preferable for small datasets & SGD for larger ones.

Support Vector Machine (Geometric Intuition) (1.1):-

→ can do both classification & regression

→ Key idea is to seperate +ve & -ve points as widely as possible.



→ π is better than π'.
π is called margin maximizing hyperplane.

→ π^+ & π^- are hyperplanes parallel to π.
π^+ touches a +ve point
π^- touches a -ve point.
$dist(\pi^+, \pi^-)$ = margin.
The larger the marigin, the wider the π seperates them both.

→ SVM tries to find π that maximizes the marigin.

→ As marigin increases, generalization accuracy (accuracy on unseen datapoints) increases.

→ The points through which π^+ & π^- pass through are called support vectors.

→ π^+ : positive hyperplane, π^- : negative hyperplane

Alternative Geometric Interpretation :-

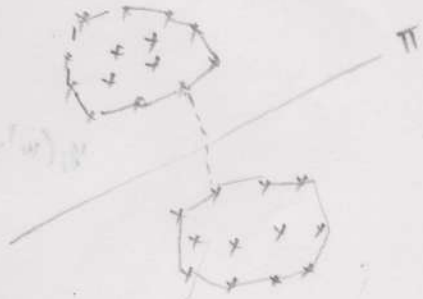
convex hull :-

either all points are on the convex hull or inside the hull & this is the smallest possible shape. It has to be a convex shape ie, shortest path b/w any two points inside should not be outside the convex shape.

step① :- ~~Draw~~ convex hull of +ve & -ve points

step② :- find shortest line connecting both hulls

step③ :- bisect line. New line obtained is margin maximizing hyperplane.



→ SVM minimizes hinge loss whereas logistic regression minimizes logistic loss.
SVM does things geometrically whereas LR does things statistically.

→ SVM focuses on points that are hardest to distinguish using other methods.
This is the reason behind using hinge loss.

(70) Mathematical formulation of SVM (1.2):-

Let $\pi = w^T x + b = 0$, w is $\perp$er to hyperplane

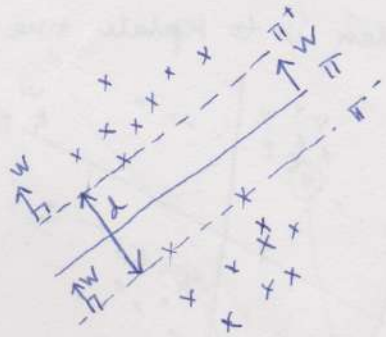
w will also be $\perp$er to π^+ & π^-

w is not necessarily a unit vector.

Let $\pi^+ = w^T x + b = 1$

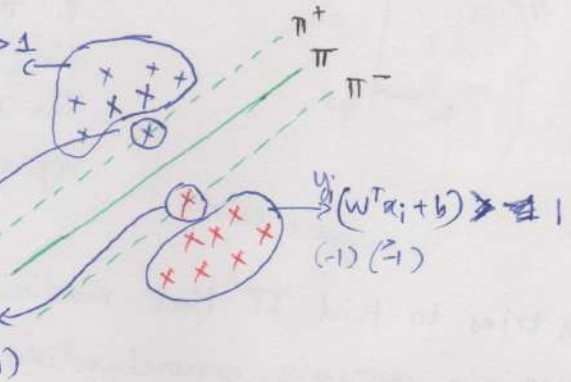
$\pi^- = w^T x + b = -1$

$$d = \frac{2}{\|w\|}$$



→ $\underset{w,b}{\text{argmax}} \frac{2}{\|w\|}$ such that $y_i(w^T x_i + b) = 1$ for SV
< 1 for −ve
> 1 for +ve

n constraints (n points)

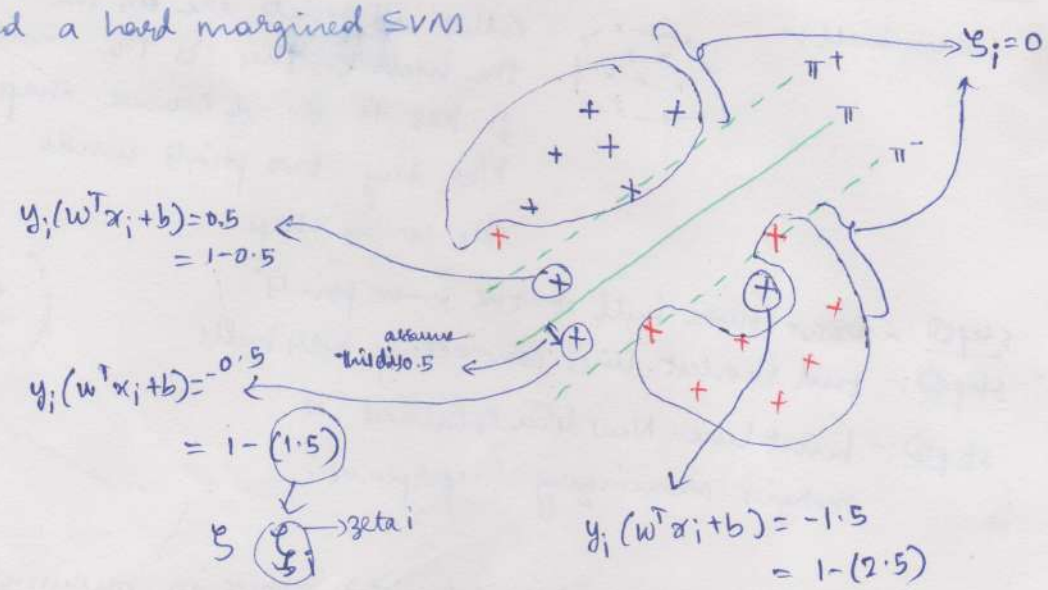


→ $w^*, b^* = \underset{w,b}{\text{argmax}} \frac{2}{\|w\|}$ such that $\forall i,\ y_i(w^T x_i) + b \geq 1$

constraint optimization problem of SVM.

→ if data is not linearly seperable, the optimization problem sometimes cannot be solved.

So, this is called a hard margined SVM



As ξ_i increases, we can know that point is further away from correct hyperplane in the incorrect direction.

→ $x_i \rightarrow \xi_i$

$\xi_i = 0$ if $y_i(w^T x_i + b) \geq 1$ (correctly classified)

$\xi_i > 0$ & if it is equal to some units of distance away from correct hyperplane in the incorrect direction.

$\rightarrow \underset{w,b}{\text{argmax}} = \frac{2}{\|w\|} = \underset{w,b}{\text{argmin}} \frac{\|w\|}{2}$ $\left(\because \max f(x) = \min \frac{1}{f(x)} \right)$

hyperparameter

$\rightarrow$ average dist of misclassified points from correct hyperplane.

$$(w^*, b^*) = \underset{w,b}{\text{argmin}} \frac{\|w\|}{2} + c \cdot \frac{1}{n} \sum_{i=1}^{n} \xi_i$$

such that $y_i (w^T x_i + b) \geq 1 - \xi_i \;\; \forall \; \xi_i \geq 0$ } for correctly classified points.

minimize errors = minimize misclassifications

= minimize $\sum \xi_i$

rewriting the eqn

$$= \underset{w,b}{\text{argmin}} \underbrace{\frac{\|w\|}{2}}_{\text{regularization}} + c \cdot \underbrace{\frac{1}{n} \sum_{i=1}^{n} \xi_i}_{\text{loss (hinge loss)}} \quad \text{such that } y_i (w^T x_i + b) \geq 1 - \xi_i \; \forall \xi_i \geq 0$$

soft margin SVM

hyperparameter. (Always +ve)

As $c \uparrow$, tendency to make mistakes on D_{train} reduces ie., overfit

as $c \downarrow$, tendency to underfit ie., high bias.

$\rightarrow$ if we have 2 equations $ax + by + c_1 = 0$, $ax + by + c_2 = 0$, distance b/w these two lines $= \frac{|c_2 - c_1|}{\sqrt{a^2 + b^2}}$

Here we are using 2 hyperplanes whose equations are

$w^T x + b = 1$ $\quad w^T x + b = -1$

$\Rightarrow w^T x + (b-1) = 0$ $\quad w^T x + (b+1) = 0$

$w^T x = w^1 * x_1 + w_2 * x_2 + \cdots + w_d^* x_d$

So the distance would be $\frac{|\text{difference of vector components}|}{\|w\|}$

$$= \frac{|w^T x + b - 1 - (w^T x + b + 1)|}{\|w\|} = \frac{2}{\|w\|}$$

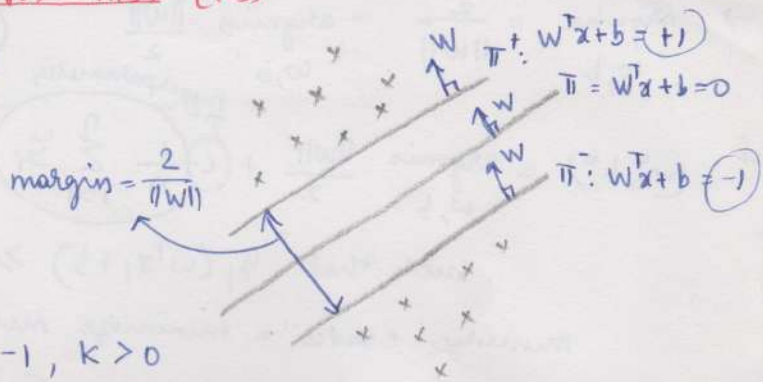
(72) Why take negative & positive values for SVMs :- (1.3)

w is not a unit vector ie, $\|w\| \neq 1$

$w \perp \pi$ is the only condition

Task is to find w^* & b^*

$$w^*, b^* = \underset{w,b}{\operatorname{argmax}} \frac{2}{\|w\|}$$



⇒ Using $+k$ & $-k$ instead of $+1$ & -1, $k > 0$

ie, $\pi^+ : w^T x + b = +k$ $\quad \pi^- : w^T x + b = -k$

$$\Rightarrow \text{margin} = \frac{2k}{\|w\|}$$

since k is a constant,

$$\underset{w,b}{\operatorname{argmax}} \frac{2}{\|w\|} = \underset{w,b}{\operatorname{argmax}} \frac{2k}{\|w\|}$$

⇒ +1 & −1 are taken for convenience.

→ Another interpretation

$$w^T x + b = k$$

$$\Rightarrow \left(\frac{w}{k}\right)^T x + \left(\frac{b}{k}\right) = 1$$

$$\Rightarrow (w')^T x + b' = 1$$

only rule is w is ⊥ to π. We donot care about $\|w\|$. So dividing by k is fine.

→ two lines are said to be parallel if they have same slope & different intercepts. Here they have slopes w, intercepts $b, b+1, b-1$.

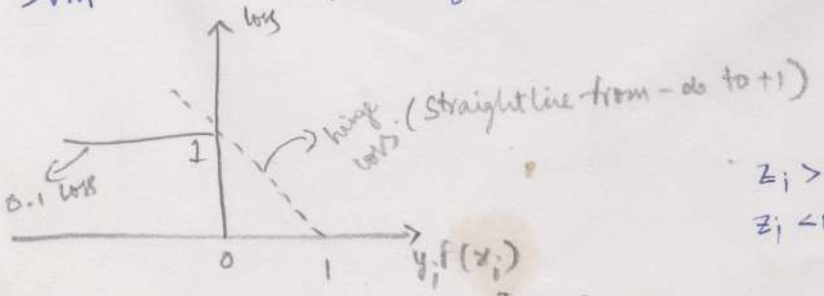
→ w is a perpendicular vector to both planes, $b+1$ & $b-1$ is dot product of vectors π^+ & π^-.

Loss function (Hinge loss) based interpretation :- (1.4)

logistic regression :- logistic loss + regularization

linear " :- linear loss + "

SVM :- hinge loss + "



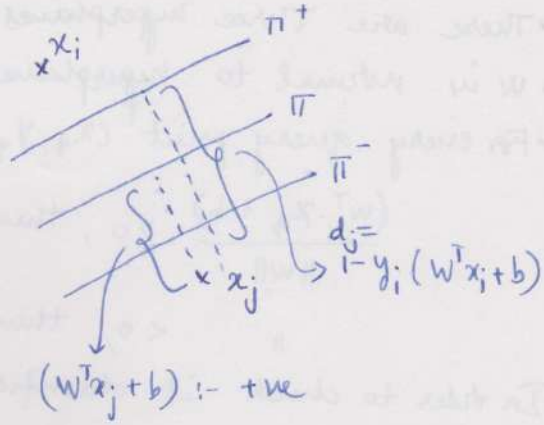
$= y_i(w^T x_i + b) = z_i$

$z_i > 0$:- x_i is correctly classified.

$z_i < 0$:- " " incorrectly "

hinge loss :- $z_i \geq 1 \rightarrow 0$
$z_i < 1 \rightarrow 1 - z_i$ } $\rightarrow$ can also be written as $\max(0, 1 - z_i)$

geometric formulation & loss minimization :-



$\xi_i = 0$ for x_i

x_i & x_j are positive points

d_j = distance of misclassified point from π^+

$d_j = 1 - \underbrace{y_j}_{-ve}\underbrace{(w^T x_i + b)}_{+ve} = 1 - z_j$

ξ_j = dist from x_j to the correct hyperplane (π^+) $= 1 - z_j$

$\xi_j = 1 - z_j$ when x_j is misclassified.

$\Rightarrow$ hingeloss $= \max(0, 1 - z_i) = \xi_i$

soft svm :- $\min_{w,b} \frac{\|w\|}{2} + C \sum_{i=1}^{n} \xi_i$ such that $(1 - y_i(w^T x_i + b)) \leq \xi_i$ & $\xi_i \geq 0$

loss minimization :- $\min_{w,b} \sum_{i=1}^{n} \max(0, 1 - \underbrace{y_i(w^T x_i + b)}_{z_i}) + \lambda \|w\|^2$

$\|w\| \geq 0 \Rightarrow$ minimizing $\frac{\|w\|}{2}$ is same as minimizing $\|w\|^2$

C & λ are hyperparameters. C multiplied with loss & λ " " regularization

$C \uparrow \rightarrow$ overfit $\lambda \uparrow \rightarrow$ underfit

$\xi_i = 0$ for correctly classified points & $\xi_i > 0$ for incorrectly classified points.

$\therefore$ These two are the same.

Summary of SVM:-

- There are three hyperplanes π^-, π, π^+
- w is normal to hyperplane π
- For every query point (x_q, y_q), if

$$\frac{(w^T.x_q + b)}{\|w\|} > 0, \text{ then } (x_q, y_q) \text{ is predicted as positive.}$$

$$\text{"} \quad < 0, \text{ then } (x_q, y_q) \text{ is predicted as negative.}$$

- In order to check if classification is correct or not we use y_q
- If $\dfrac{(y_q^* (w^T x_q)) + b)}{\|w\|} > 0$, then it is correctly classified

$$\text{"} \quad < 0, \text{ then it is misclassified}$$

- Distance of point (x_q, y_q) from hyperplane π (irrespective of whether classifn is correct or not) $= ((w^T.x_q) + b) / \|w\|$.
- If (x_q, y_q) is correctly classified & does not lie within the margin:-
  - if (x_q, y_q) is +ve:-

    dist from (x_q, y_q) to π^+ :- $\dfrac{((w^T.x_q)+b)}{\|w\|} - \dfrac{1}{\|w\|}$

    dist from (x_q, y_q) to π^- :- $\dfrac{((w^T x_q)+b)}{\|w\|} + \dfrac{1}{\|w\|}$
  - if (x_q, y_q) is -ve:-

    dist from (x_q, y_q) to π^- :- $\dfrac{((w^T x_q)+b)}{\|w\|} - \dfrac{1}{\|w\|}$

    dist from (x_q, y_q) to π^+ :- $\dfrac{((w^T x_q)+b)}{\|w\|} + \dfrac{1}{\|w\|}$
- If (x_q, y_q) is misclassified & does not lie in the marigin.
  - if (x_q, y_q) is +ve but classified as -ve,

    dist from (x_q, y_q) to π^+ :- $\dfrac{w^T.x_q + b}{\|w\|} + \dfrac{1}{\|w\|}$

    dist from (x_q, y_q) to π^- :- $\dfrac{w^T x_q + b}{\|w\|} - \dfrac{1}{\|w\|}$
  - if (x_q, y_q) is -ve but classified as +ve:-

    dist from (x_q, y_q) to π^- :- $\dfrac{(w^T.x_q) + b}{\|w\|} + \dfrac{1}{\|w\|}$

    dist from (x_q, y_q) to π^+ :- $\dfrac{w^T.x_q + b}{\|w\|} - \dfrac{1}{\|w\|}$

→ same.
$(+, +) \to \pi^+$
$(-, -) \to \pi^-$
$(+, -) \to \pi^-$
$(-, +) \to \pi^+$
correct classified

Dual formulation of SVM:- (1.5)

→ soft margin SVM is aka primal formulation of SVM.

$$\min_{w,b} \frac{1}{2}\|w\| + C\sum_{i=1}^{n} \xi_i \text{ such that } y_i(w^T x_i + b) \geq 1 - \xi_i \;\; \forall\, \xi_i \geq 0$$

primal can also be written as

$$\underbrace{\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \quad s.t\; C \geq \alpha_i \geq 0 \quad \sum_{i=1}^{n} \alpha_i y_i = 0}_{\text{aka dual form of SVM.}}$$

solving both are equivalent.

→ in dual formulation
- $f(x_q) = \sum_{i=1}^{n} \alpha_i y_i x_i^T x_q + b$
- for every x_i there is a corresponding α_i
- x_i's only occur in the form of $x_i^T x_j$
- $\alpha_i > 0$ for only support vectors.

  $\alpha_i = 0$ for non support vectors

  ⇒ for computing $f(x_q)$ we only need support vectors.

$$\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \underbrace{x_i^T x_j}_{} \quad \text{such that } \alpha_i \geq 0 \;\; \sum_{i=1}^{n} \alpha_i y_i = 0.$$

↳ $x_i \cdot x_j$ = cosine similarity (x_i, x_j) if $\|x_i\| = 1$, $\|x_j\| = 1$.

⇒ $x_i^T x_j$ can be replaced with any similarity b/w (x_i, x_j)

⇒ similarity matrices can be used

⇒ usually $k(x_i, x_j)$ replaces $x_i^T x_j$

↓ kernel function

$$f(x_q) = \sum_{i=1}^{n} \alpha_i y_i \underbrace{x_i^T x_q}_{k(x_i, x_q)} + b$$

α_i is called lagrange multipliers.

Kernel trick (1.6):-

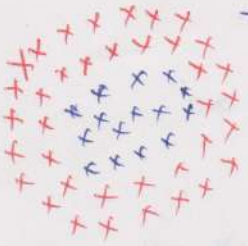
$$f(x_q) = \sum_{i=1}^{n} \alpha_i y_i k(x_i, x_q) + b$$

→ replacing with $k(\,)$ is kernel trick.

$$\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \quad s.t\; C \geq \alpha_i \geq 0 \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

→ Kernel trick is the most important idea in SVM.

→ If $k(\,)$ is not used in the equation, then it is called linear SVM because the concept is similar. Its called kernel SVM if $k(\,)$ is used.



→ log regression :- fail
linear SVM :- fail
log regression + feature transforms :- works
kernel SVM with the right kernel :- works. ie, kernel SVM can seperate non linear datasets. This process is called Kernealization.

→ For some data it is not possible to find the right hyperplane that classifies the data. but there might be one when projected to higher dimensions. This is what kernel tries to do. Find the complex model that could seperate data by moving data into a higher dimensions space.

Polynomial Kernel (1.7) :-

→ Kernalization $K(x_1, x_2) = (x_1^T x_2 + c)^d$

Quadratic Kernel :- $(1 + x_1^T x_2)^2$ $(c = 1, d = 2)$

ex :- let $K(x_1, x_2) = (1 + x_1^T x_2)^2$

if $x_1 = \langle x_{11}, x_{12} \rangle$ $x_2 = \langle x_{21}, x_{22} \rangle$

$$\Rightarrow K(x_1, x_2) = (1 + x_{11}x_{21} + x_{12}x_{22})^2$$

$$= \underbrace{1 + x_{11}^2 x_{21}^2 + x_{12}^2 x_{22}^2 + 2x_{11}x_{21} + 2x_{12}x_{22} + 2x_{11}x_{12}x_{21}x_{22}}_{\text{writing this as product of two vectors}}$$

went from 2D data to 6D data

$$[1, x_{11}^2, \cancel{x_{21}}, x_{12}^2, \cancel{x_{22}}, \sqrt{2}x_{11}, \sqrt{2}x_{12}, \sqrt{2}x_{11}x_{12}] :- x_1'$$

$$[1, \cancel{x_{11}}, x_{21}^2, \cancel{x_{12}}, x_{22}^2, \sqrt{2}x_{21}, \sqrt{2}x_{22}, \sqrt{2}x_{21}x_{22}] :- x_2'$$

$$= (x_1')^T (x_2') \quad [\text{same as feature transformation}]$$

these features alone are enough

Different kernels :-

linear :- $K(x, y) = x^T y$
polynomial :- $K(x, y) = (x^T y + 1)^d$
sigmoid :- $K(x, y) = \tanh(a x^T y + b)$
RBF :- $K(x, y) = \exp(-\gamma \|x - y\|^2)$

→ RBF is better for most problems.

Radial Basis Function (RBF) (1.8):-

→ most popular & general purpose kernel.

$$(x_1, x_2) \Rightarrow K_{RBF}(x_1, x_2) = \exp\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$$

σ : hyper parameter

$\xleftarrow{d_{12}}$

$x_1 \qquad x_2$

$\|x_1 - x_2\|^2 = d_{12}^2$

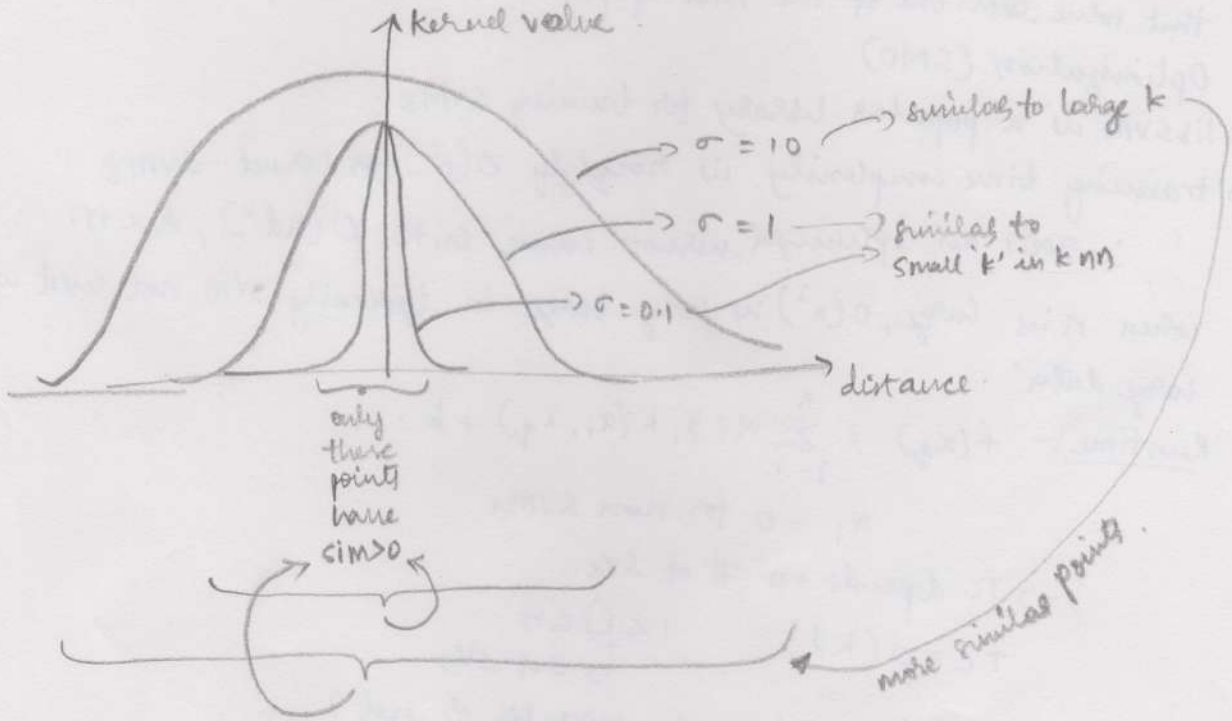
① As $d_{12} \uparrow$, $K(x_1, x_2) \downarrow$

$K(x_1, x_2) > K(x_1, x_3)$

$x_1 \qquad x_2 \qquad x_3$

as distance increases, kernel value falls exponentially.

⇒ kernel :- similarity
distance :- dissimilarity



K-NN and RBF-Kernel :-

$\sigma \uparrow$ (in RBF) $\Rightarrow k \uparrow$ in KNN. In this way SVM is similar to KNN.

→ In KNN we need to store all n points & LSH → $O(nd)$

In RBF, only support vectors

often times $\#SV << n$

RBF-SVM is nice approximation to KNN

→ grid search | random search must be used to get vals of c & σ

(14) Domain specific Kernels :- (1.9)

→ RBF :- general purpose kernel

→ Kernel trick similar to $\underbrace{\text{Feature transformation}}_{\text{domain specific}}$

Similarly, there are domain specific kernels as well

→ Feature transformation is partially replaced by finding the right kernel

→ Linear SVM makes some assumptions unlike kernel SVM. They are

(i) Margin should be as large as possible.

(ii) Support Vectors are the most useful data points as they are the ones most likely to be misclassified.

Train and Runtime Complexities (1.10) :-

→ training can be done with SGD. But there are specialized algorithms that solve SVM. One of the most popular ones is Sequential Minimal Optimization (SMO)

→ libSVM is a popular library for training SVMs.

→ training time complexity is roughly $O(n^2)$ for kernel-SVMs.

in 2007 an optimized version came with $O(nd^2)$, $d < n$.

when n is large, $O(n^2)$ is very large. so typically SVM not used with large data

Runtime :- $f(x_q) = \sum_{i=1}^{n} \alpha_i y_i k(x_i, x_q) + b$.

$\alpha_i = 0$ for non SVMs

⇒ TC depends on # of SVs.

$TC = O(kd)$ $\quad 1 \le k \le n$

↳ # of SVs

→ how to know if data is linearly seperable or not ?

(i) Single Perceptron models :- if hulls overlap, then not linearly seperable.

(ii) SVM :- if residuals are present, data is not linearly seperable.

(iii) data visualization

difference b/w observed & predicted aka errors.

nu-SVM (1.11):-

→ standard SVM :- C-SVM (original formulation, $C \geq 0$)

alternative form :- nu-SVM
↓
new parameters

$0 \leq \nu \leq 1$

$\nu \geq$ fraction of errors → this is useful since $\geq$

$\nu \leq$ fraction of support vectors → this is less useful cuz $\leq$

→ if we want only 10% errors, we can say $\nu = 0.1$
1% error, we can say $\nu = 0.01$

→ ν can be found using grid search

→ if $\nu = 0.01$ ⇒ % of errors = 1%
# of SVs $\geq$ 1% of n

→ for run time complexity, we need fewer SVs.

→ ex :- $n = 100{,}000$, $\nu = 0.01$ ⇒ errors $\leq$ 1%
SVs $\geq$ 1% of n

→ the error it tries to reduce is train error, so extremely low values ~~might~~ lead to overfitting.

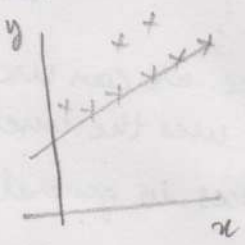
→ nu-SVM computationally more expensive but offers more interpretability

SVM Regression (1.12):-

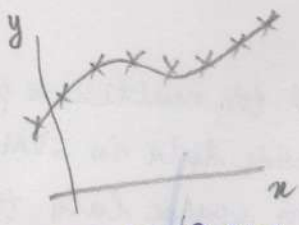
→ $\min_{w,b} \frac{1}{2}\|w\|^2$ such that $y_i - (w^T x_i + b) \leq \epsilon$ $\forall_i$ $(w^T x_i + b) - y_i \leq \epsilon$, $\epsilon \geq 0$.
↳ hyperparameter.

Linear form of SVR. can also be kernalized.

→



linear SVM can only fit straight lines



Kernel SVR can fit curves as well

$\epsilon \downarrow$ ⇒ errors are low on training data ⇒ overfitting

→ two constraints in eqn can be replaced with one $|y_i - \hat{y}_i| \leq \epsilon$, $\epsilon \geq 0$.
$\hat{y}_i = (w^T x_i + b)$

→ we basically try to fit as many points as possible b/w hyperplane π^+ & π^- that are ϵ dist away.

Real world cases (1.13) :-

→ Feature engg & feature transform :- finding the right Kernel. RBF is a defaut option
Kernel ↓ core concept

→ decision surface :- linear SVM :- hyperplane
kernel SVM :- original :- d x_i → non linear surface.
$d \downarrow d'$ x_i' → linear surface

→ If similarity fn/dist fn is given :- kernel trick

→ Interpretability & feature Importance :- no way to get feature importance easily/naturally.
We can always use forward feature selection

→ Outliers :- very little impact. Because it's only support vectors that matters
But RBF with small σ behaves like KNN with small K.

→ Bias Variance Tradeoff :- $C \uparrow \Rightarrow$ overfit (high variance)
$C \downarrow \Rightarrow$ underfit (high bias)

→ Large dimensionality :- good for SVM as RBF tries to do this.

→ Best cases :- appropriate kernel is available
worst case :- train data size is large as TC is high. Bad for internet apps
K is large (# of SVs)
When train data is large, people generally use logistic regr with feature transforms

Code Sample (1.14) :-

→ tolerance = 0.001 means if $(w_i - w_{i+1}) < 0.001$, terminate.

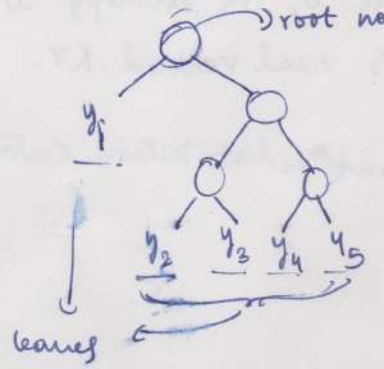
Interview Questions :-

→ Random forest intrinsically suited for multiclass problems. We can use data as it is. We need to encode data in SVM as it uses the concept of distance. SVM performs better on sparse data than trees in general.

Decision Tree Geometric Intuition :- (3.1)

KNN :- Instance Based
NB :- Probabilistic
LR, Log R, SVM } :- Geometric

→ Decision trees :- kind of like if else conditions. Model can be implemented in the form of nested if else conditions.



- highly interpretable

→ The nested conditions divide the plane into regions.
All the hyperplanes are axis parallel.

∴ Decision trees = set of axis parallel hyperplanes

→ Decision tree are non linear because there is no equation ~~b/w~~ that seperates the points.

Building a decision tree : Entropy (3.3) :-

→ Related to entropy from information theory.

→ If RV Y can take $y_1, y_2, y_3, \ldots, y_K$ values

$$\text{entropy of } Y = H(Y) = \sum_{i=1}^{K} P(y_i) \log_b(P(y_i))$$

→ $b = 2$ (or) e

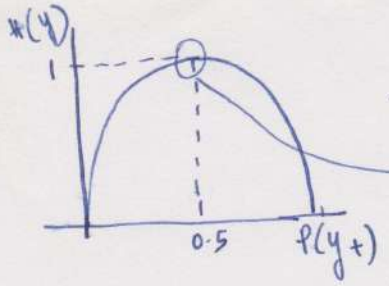
→ Assume Y can take 2 values, $Y \to y_+, y_-$

Properties of entropy :-

Case ① :- $D \to y_+ \to 99\%$, $D \to y_- \to 1\%$ } $H(y) = -0.99 \log(0.99) - 0.01 \log 0.01 = 0.0801$

Case ② :- $D \to y_+ \to 50\%$, $D \to y_- \to 50\%$ } $H(y) = -0.5 \log 0.5 - 0.5 \log 0.5 = 1$

Case ③ :- $D \to y_+ \to 0\%$, $D \to y_- \to 100\%$ } $H(y) = 0$



→ for a 2 class RV

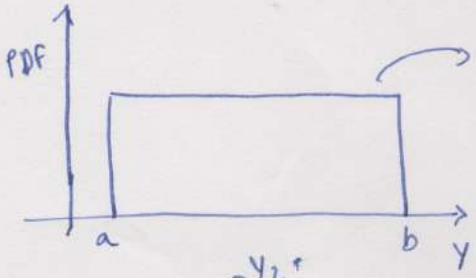
→ max entropy when equal amounts

→ Given a k value RV, if all vals are equally probable, then entropy is maximum.

if y_1 is most probable & rest are 0, the entropy is minimum

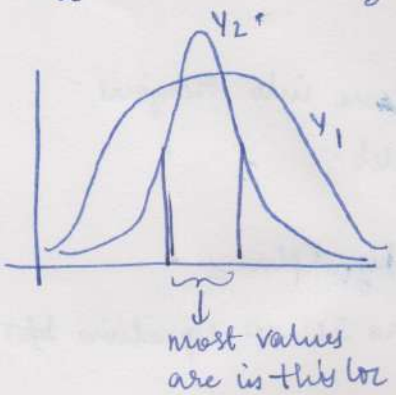
→ entropy can also be calculated for real valued RV.

ex① :-



→ uniform distribution entropy is max

ex② :-



Y_1 → less peaked ⇒ ~~less~~ more spread of values

Y_2 → more peaked ⇒ less spread of values

⇒ $H(Y_2) < H(Y_1)$

→ The more peaked a distribution is, the less it's entropy is

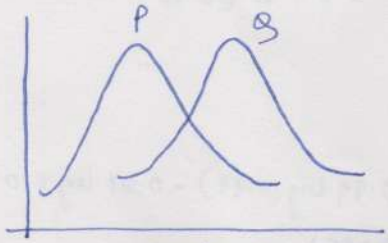
→ entropy can also be used for EDA.

→ For continuous RV, entropy is $H(x) = \log(b-a)$

↳ interval of values.

Kullback Leibler (KL) Divergence :- (3.4)

Let P & Q be two funs



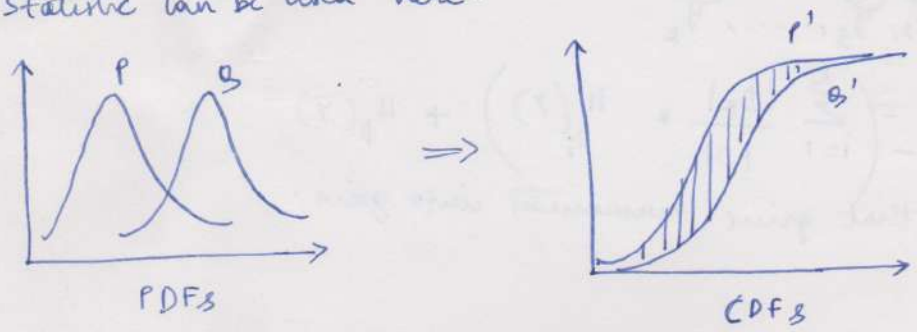
PDF/PMFs of P & Q.

objective is to find dist b/w P & Q.

if P & Q are almost same, dist is small.

if P & Q are very diff, dist is large.

→ KS statistic can be used here.



$$\text{KS statistic} = \sup\{|P'(x) - Q'(x)|\}$$

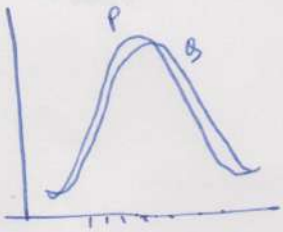
↓ will be small if P & Q are similar

→ while this is good, sup can't be differentiated. So it can't be used in loss functions in optimization based ML problems.

So this is why we use KL divergence. Origins are from information theory & entropy in thermodynamics.

$$D_{KL}(P\|Q) = \sum_x P(x)\log\left(\frac{P(x)}{Q(x)}\right)$$ for discrete random variables.

$$\int_x P(x)\log\left(\frac{P(x)}{Q(x)}\right)$$ for cont random vars



if P & Q are similar, $\frac{P(x)}{Q(x)}$ at point x will be close to 1, ⇒ $\log\left(\frac{P(x)}{Q(x)}\right)$ will be close to 0.

if they are dissimilar, $\log\left(\frac{P(x)}{Q(x)}\right)$ will be large.

→ KL divergence is also called relative entropy

KL divergence is differentiable

→ T-SNE also uses KL divergence to find dist b/w X in l dim & X in d' dim

Information Gain (3.5):-

if D is broken into D_1, D_2, D_3

$$\text{weighted entropy} = \frac{len(D_1)}{len(D)} \times \text{entropy of } D_1 + \frac{len(D_2)}{len(D)} * \text{entropy of } D_2 + \frac{len(D_3)}{len(D)} * \text{entropy of } D_3.$$

information gain, $IG(Y, D) = (\text{weighted entropy}) + \text{entropy of } D \text{ (before breaking)}$

$Y \xrightarrow{\text{variable}} \overset{D_1}{y_1}, \overset{D_2}{y_2}, \overset{D_3}{y_3}, \dots, \overset{D_k}{y_k}$

$$IG(Y, var) = -\left(\sum_{i=1}^{k} \frac{|D_k|}{|D|} * H_{D_i}(Y)\right) + H_D(Y)$$

→ we choose split that gives maximum info gain.

Gini Impurity (3.6):-

→ similar to entropy

$I_G(Y)$ not same as $IG(Y)$

→ $I_G(Y) = 1 - \sum_{i=1}^{k} (P(y_i))^2$ $\quad Y \to y_1, y_2, y_3, \dots, y_k$

<u>case ①</u>:- $P(y_+) = 0.5 \quad P(y_-) = 0.5$

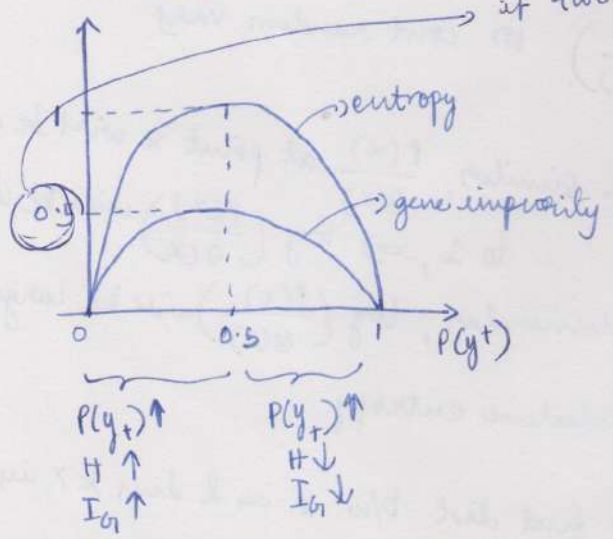
$I_G(Y) = 1 - (0.25 + 0.25) = 0.5$

$H(Y) = 1$

<u>case ②</u>:- $P(y_+) = 1 \quad P(y_-) = 0$

$I_G(Y) = 1 - (1 + 0) = 0$

$H(Y) = 0$



if two class - if three classes it could be b/w [0, 0.67]

⋮

(code sample video comment)

$I_G(Y) \to 1 - \{P(y_+)^2 + P(y_-)^2\}$ ⟶ no log

$H(Y) \to -P(y_+) \log P(y_+) - P(y_-) \log (P(y_-)) \to$ log.

→ computing square easier than computing log.

⇒ I_G is more computationally efficient. Hence used in scikit learn & other implementations.

→ we are calculating the purity of the split of data. If it is pure (only one class) we get value of 0, otherwise not 0.

Constructing a decision tree (3.7) :-

① $D \longrightarrow 9+, 5-$

$H_D(Y) = 0.94$

② D → Outlook

Sunny	Overcast	Rainy
D_1: 2+ 3-	D_2: 4+ 0-	D_3: 2+ 3-

weighted entropy $= 0.69$

Information gain $= 0.94 - 0.69$
$= 0.25$

D → Temperature

H	M	C
D_1: 2+ 2-	D_2: 4+ 2-	D_3: 3+ 1-

weighted entropy =

IG = 0.94 − ____

After calculating all, we have

IG (Y, Outlook) = ____
IG (Y, temp) = ____
IG (Y, Humidity) = ____
⋮
____

} we pick the one with maximum information gain

IG (Y, f) = entropy at parent level − weighted entropy at child level

Assume in this case, it's outlook

D → S: 2+ 3- (D_1) → D_{11}, D_{12}
D → O: 4+ 0- (D_2) → pure node. No need to break further.
D → R: 2+ 3- (D_3) → D_{31}, D_{32}, D_{33}

→ Calculate IG again using the remaining features.

→ After each split,
(i) reached pure node. Stop growing the node.
(ii) can't grow further due to lack of points. (only one point. That point could be a noisy point)
(iii) if we are too deep.

→ As depth of the tree increases, overfitting chances increases.
if depth is small, underfitting chances increases.

→ depth is the hyperparameter in decision trees. CV can be used to find depth.

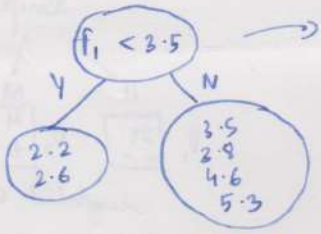
(86) Splitting Numerical features (3.8) :-

→ entropy won't work on numerical features.

→

f	y
2.2	1
2.6	1
3.5	0
3.8	0
4.6	1
5.3	0

Sort values in ascending order



→ method to split numerical vars.

→ $f_1 < T_1 \rightarrow D_1, D_2$

$f_1 < T_2 \rightarrow D_1, D_2$

⋮

$f_1 < T_n \rightarrow D_1, D_2$

$T_1, T_2, \ldots, T_n$ → thresholds

n possible splits.
We pick the one with maximum information gain.

→ max depth can be greater than the number of features

age → gender → age > 40 → gender = M . . .

we dont have to stop here

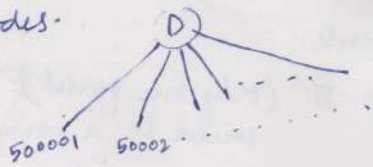
Feature Standardization (3.9):-

→ not a dist based method. No need to perform feature standardization.

→ onehot encoding on categorical features effect accuracy as they make tree grow in one direction.

categorical feature with many possible values (3.10):-

ex:- pincodes.



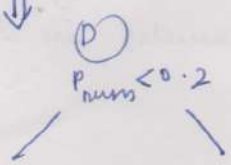
instead

Pincode	Y
500001	0
500001	1
⋮	

replace value with

$$P(Y=1 \mid P_j) \rightarrow \frac{\#\, Y=1 \text{ when Pincode} = P_j}{\#\text{rows}}$$

⇓



→ when we have categorical feature with many elements & it is nominal in nature, we have 3 approaches.

case(i) :- when response variable is categorical and continuous

→ we bin the feature into fewer sub categories.

→ This is known as binning

case(ii) :- when response variable is continuous

→ Replace each category with mean/median

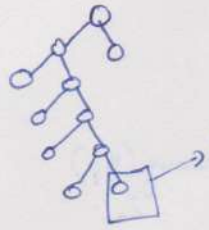
case(iii) :- when response variable is discrete/categorical

→ Replace each x_{ij} with $P(y_i = C \mid F_j = \text{"ABC"})$

→ These are called response variable encoding.

Overfitting & Underfitting (3.11) :-

→ As depth increases, possibility of having very few points at leaf node increases.



overfitting model to noise.

erroneous points/outliers/human corrupted data.

→ depth ↑ interpretability ↓

→ tree of depth 1 = decision stump.

→ using CV to find right depth is good approach.

Train & Runtime Complexity (3.12) :-

Train $\sim O(n(\lg n)d)$ $\quad n$ = # of points, d = dimensions

→ there are algos that help us handling numerical features

$\underbrace{n \lg n}_{\text{sort}}\, d$ → to evaluate the split.

→ Runtime space comp :- after training dec tree, we convert it to nested if else.

⇒ space comp = # of internal nodes + # of leaf nodes.

⇒ $SC = O(\text{nodes})$.

→ people typically have DT of size 3 - 10.

→ runtime SC is reasonable.

→ runtime TC :- maximum depth of any leaf node.

→ DT is good when we have large data, dimensionality is small, low latency is a requirement.

→ Random Forest & Gradient Boost Dec Trees are extremely popular.

Regression Using Decision Tree :- (3.13) :-

Information Gain → classification

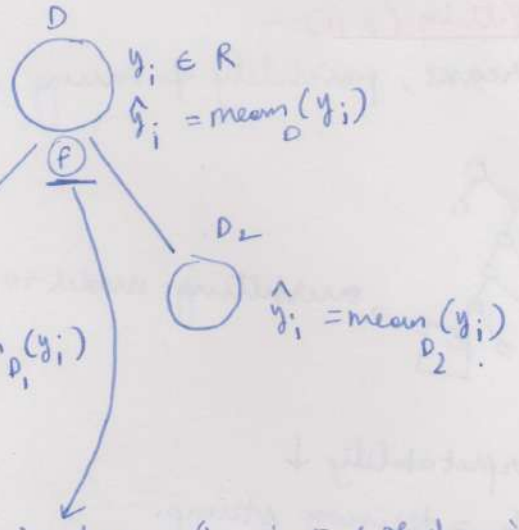
Mean Square Error / Median Absolute Deviation → Regression

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

this is like entropy.

instead of weighted entropy

$w_1^* MSE_{D_1} + w_2^* MSE_{D_2}$



whichever feature reduces this is chosen (ie., instead of choosing feature that reduces entropy, we choose feature that reduces MSE/MAD).

Cases (3.14) :-

imbalanced data :- balance it using upscaling / class weight.
imbalanced data impacts entropy calculation / MSE calculation

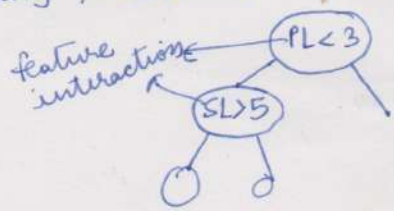
large 'd' :- time comp increases. So if we have categorical features, we must avoid OHE as dimensionality increases.

If similarity matrix is given :- DT won't work. Needs features explicitly.

Multiclass classifn :- DT will work. It will look for majority class at leaf node. No need for one vs rest.

decision surfaces :- non linear axis parallel hyperplanes.

feature interactions :- inbuilt in decision trees.



Outliers :- depth ↑, outliers impact will be more.

Interpretability :- Very interpretable

Feature importance :- for every f_i, sum up (normalized sum) reductions in H or I_G due to that f_i, if lot is being reduced, its an important feature.

sklearn gives feature importance.

Interview Questions & Revision (Decision trees) :-

→ post pruning :- tree is built first and then reduction of branches.

pre pruning :- decision tree is built first and we keep checking if it's overfitting or not.

→ info gain, entropy, MSE are called Attribution Selection Measures.

→ DT is known to work best with non linear. Known to detect non linear interactions

Ensemble Models (5.1) :-

→ group of things.

→ when we have multiple models being used together.

$$\underbrace{M_1, M_2, M_3 \ldots, M_K}_{\text{(combine) into a more powerful model. 'M'.}} \rightarrow \text{base models.}$$

→ Four types mostly used

Bagging (Bootstrapped Aggregation)
Boosting
Stacking
Cascading.

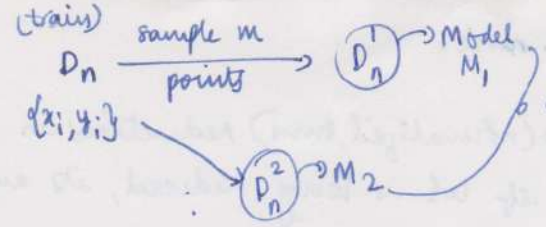
→ Build high performing powerful models.

→ Most kaggle winners use them.

→ Key aspect :- The more different these models $(M_1, M_2, M_3 \ldots M_K)$ are, the better they can be combined.

Bootstrapped Aggregation (Bagging) (5.2) :-

→ popular model :- random forest



→ M_i is built using a different subset of data D_n^i $(m \le n)$ with each having m points.

→ the samples are called bootstrap samples.

→ $M_1, M_2, M_3 \ldots M_k$ are aggregated together to form M.

→ typical aggregation for classification :- majority vote
regression :- mean/medians.

⇒ Bagging can reduce variance in a model without impacting bias.

model error = $\text{Bias}^2 + \text{variance}$

Because, if the points causing high variance are removed, it wont effect the aggregated model that much.

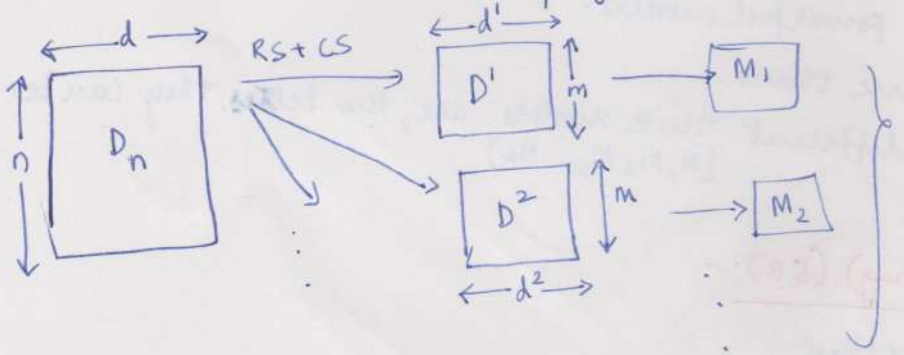
if base model (M_i) → low bias & high variance,
Bagging $(M_i s)$:- low bias, reduced variance.

→ We typically dont hyper param tune base model (M_i)

Random forest and their construction (5.3) :-

→ Random bootstrap sampling

→ RF = Decision Trees + $\underbrace{\text{Bagging}}_{\text{basically row sampling}}$ + $\underbrace{\text{Column Sampling}}_{\text{aka feature bagging}}$.



→ The individual trees are usually of max depth as high variance is reduced after bagging.

→ Points that are in D_i but not in overall D are called out of bag models. (oob). They can be used for CV. oob error = CV error. The other points are called in bag samples.

→ column sampling is done without replacement.

→ Multiple base models can have same set of features but no model will have same feature twice. Hence without replacement.

→ We use OOB error for overall bagged model's hyperparam tuning.

→ hyperparams in sikit learn are

(i) n_estimators :- number of base learners

(ii) max_depth :- maximum depth of the tree

(iii) max_features :- no. of features when looking for best split.

Bias Variance Tradeoff (5.4):-

→ RFs tend to have low bias (since base models have low bias) & low variance.

$$D_n \xrightarrow[C.S]{R.S} D'$$

d, n → $d' < d$, $m < n$

$\frac{d'}{d}$ = column sampling ratio

$\frac{m}{n}$ = row sampling ratio.

column SR ↓ } → low variance model.
row SR ↓ } Means more chance that every model will have different dataset.

Bagging training run time complexity (5.5):-

→ RF with K base learners

train :- $O(n \log n * k)$

→ RF is trivially parallelizable (multiple cores can be used to train multiple base learners at the same time)

Runtime :- $O(\underbrace{depth}_{\text{large because of high variance}} * k)$

space :- $O(\underbrace{DT}_{\text{if else}} * k)$

Code sample (5.6):-

→ bootstrap = True → row sampling will also be done.

Extremely Randomized Trees (5.7):-

ExtraTreesClassifier in sciKit learn.

→ In normal DT, for a real valued feature, we check MEA/MAD for each value by using it as threshold & choose the best one.

→ In Extremely Randomized Trees, we take random sample of vals & check only within that sample.

Random Forest :- Row Sampling + Column Sampling + Aggregation

Extreme Random Forest :- Row Sampling + Column Sampling + Aggregation + Randomization when calculating "y"

→ We are using randomization to reduce variance (RS & CS in RF, RS, CS & y in ERT)

Cases : Random Forest (5.8) :-

RF :- Decision Tree + $\underbrace{\text{RS + CS + agg}}_{\text{to reduce variance}}$.

→ All the cons for DT hold true for RF. (mostly).

→ The bias variance tradeoff of RT & DT is not the same.

→ Feature Importance :- DT :- Overall reduction in entropy or IG because of the feature @ various levels in DT.

RF :- Same as above but insted of various levels we look at various trees.

Boosting Intuition (5.9) :-

Bagging :- high var low bias base models + $\overbrace{\text{randomization (CS, RS) + aggregation}}^{\text{reduce variance}}$

Boosting :- low var high bias base models + $\underbrace{\text{additively combine}}_{\text{reduce bias while keeping variance low}}$

→ example of low var high bias = Shallow decision tree

→ Stage 0 :- D_{train} $(x_i, y_i)_{i=1}^{n}$ ⟶ M_0 $y_p = h_0(x)$ → high bias (large train error)

$y_i - h_0(x_i) = \text{error}$

⇒ for each point, we have, $\{x_i, y_i, \text{error}_i\}_{i=1}^{n}$

Stage 1 :- M_1 ⟵ training data is $\{x_i, \text{error}_i\}_{i=1}^{n}$

$F_1(x)$ = model at the end of stage 1

$= \alpha_0 h_0(x) + \alpha_1 h_1(x)$

Stage 2 :- $\{x_i, \underbrace{\text{error}_i}_{y_i - F_1(x_i)}\}$

$F_2(x)$ = model at the end of stage 2

$= \alpha_0 h_0(x) + \alpha_1 h_1(x) + \alpha_2 h_2(x)$

end of stage k :-

$$\underbrace{F_k(x)}_{\text{this ends up having a low residual error}} = \cancel{\sum_{i=0}^{K} \lambda_i h_i(x)} \quad \underbrace{\sum_{i=0}^{k} \alpha_i \overbrace{h_i(x)}^{\text{trained to fit residual error at the end of previous stage.}}}_{\text{additive weighted model.}}$$

$\Rightarrow$ train error reduces $\Rightarrow$ bias reduces.

→ Popular Boosting techs are Gradient Boosting Decision Trees and $\underbrace{\text{Adaptive boosting}}_{\text{for imagery (face detection)}}$ ~~decision trees~~

Residuals, Loss functions and Gradients (5.10) :-

$$F_k(x) = \sum_{i=0}^{k} \alpha_i h_i(x)$$

residual at the end of stage $k = err_i = y_i - F_k(x)$

→ Lets assume it's a squared loss minimization problem

$$L(y_i, F_k(x_i)) = (y_i - F_k(x_i))^2$$

taking derivative WRT $F(x_i)$

$$\frac{\partial L}{\partial F_k(x_i)} = \frac{\partial L}{\partial z_i} = \frac{\partial}{\partial z_i}(y_i - z_i)^2 = -2(y_i - z_i)$$
$$= -2(y_i - F_k(x_i))$$

$$\Rightarrow \frac{-\partial L}{\partial F_k(x_i)} = 2\underbrace{(y_i - F_k(x_i))}_{\text{residual}}$$

$\Rightarrow$ $\underbrace{\text{negative gradient at the end of stage } k \propto \text{residual}}_{\text{pseudo residual.}}$

→ if we train on pseudo residual instead of residual, we can have a loss function (any loss func that can be differentiable)

→ Random Forest : minimize hinge loss

GBDT : minimize any loss that is differentiable.

→ (base models) → high bias to low bias using combining pseudo residuals

(94) Gradient Boosting (5.11):-

Ip:- $\{x_i, y_i\}_{i=1}^{n}$, differentiable loss function $L(y, F(x))$, $\underbrace{\text{number of iterations}}_{\text{number of base trees etc,}}$ (M)

Steps:- ① $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$ } trying to build the most fluid model as initial model.

Initializing model with a constant value.

② for $m=1$ to m

(i) compute pseudo residuals

$$ps_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i=1, \dots, n.$$

(ii) Fit a base learner to pseudo residuals $\{x_i, r_{im}\}_{i=1}^{n}$

(iii) compute multiplier γ_m by solving

$$\gamma_m = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$$

(iv) Update the model

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

③ Output $F_m(x)$

→ In GBDT, the max depth is generally 1, 2, 3, 4, 5.

→ For a query point (x_q, y_q) :- At the end of the training we have each of the base models $h_m(x)$ and gamma_m's calculated.

The final model for boosting is, $F_m(x) = F_m-1(x) + \gamma^* h_m(x)$.

we can use these and solve it. We shall get corresponding y_q.

Regularization & Shrinkage :- (5.12)

$$F_M(x) = h_0(x) + \sum_{m=1}^{M} \gamma_m h_m(x)$$

→ M = # of base models

# of base models ↑ ⇒ overfit ↑ ⇒ variance ↑ ⇒ bias ↓

→ optimal M can be found using cross validation.

→ We can also use shrinkage. Introduces a learning rate $(0 < v \leq 1)$ to the update rule.

$$F_m(x) = F_{m-1}(x) + v \cdot \gamma_m h_m(x)$$

if v↓, chances of overfitting ↓, variance ↓

→ cross validation :- GridSearch (M, v)

Train and Run Time Complexity (5.13):-

train :- $O(n \lg n d * M)$

not easy to parallelize (linear sequential model) unlike random forest.

runtime :- $O(\underbrace{depth}_{\text{depth is low in GBDT}} * M)$ for one learner.

After calculating $h_m(x)$, just summing them up

$\approx O(depth * n + M) \approx O(depth * M)$

space :- $O(\underbrace{\text{stores each tree}}_{\text{if dt}} + \underbrace{\gamma_m}_{M})$

→ GBDT extensively used in low latency applications

XGBoost :- Boosting + Randomization (5.14):-

deviance loss in sklearn = logistic regression

exponential " " " = Adaboosting

→ sklearn's implementation is slow.

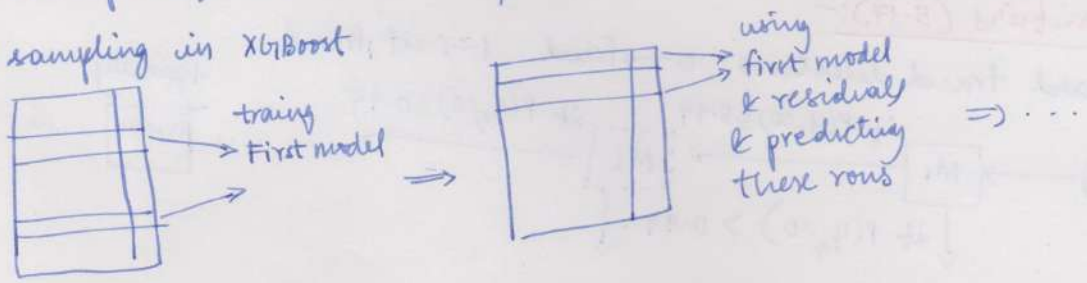
→ XGBoost is the best GBDT implementation. Uses the best of both worlds.

GBDT + $\underbrace{\text{row sampling + column sampling}}_{\text{from random forest}}$.

column-sample-by tree :- select only subset of columns for the whole tree

columnsample-by level :- select only subset of columns for calculating GI / Entropy.

→ row sampling in XGBoost



Adaboost : Geometric Intuition (5.15):-

→ used in image processing (face detection)

→ start by creating a decision stump.

calculate errors.

Increase weight of misclassified points (exponentially) [o/p :- $\alpha_1 h_1$)

create stump again

$\vdots$ $\alpha_2 h_2$

$\vdots$

$\alpha_3 h_3 \ldots$

final $= \alpha_1 h_1 + \alpha_2 h_2 + \cdots$

α is equivalent to γ in GBDT.

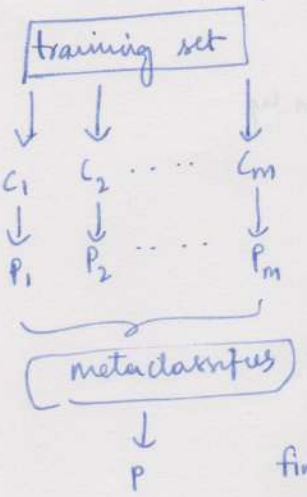
(96) → At every stage we are adapting & fixing errors.

Stacking models (5.16):-

→ mlxtend is a popular choice

from mlxtend import StackingClassifier

→



classification models y

predictions

final predictions

ideally, the more different these are, the better. These are constructed parallelly & with independent features.

→ if there are m base models,

the input of metaclassifier is $\{h_1(x_i), h_2(x_i), \dots, h_m(x_i), y_i\}_{i=1}^{n}$

ie., it's trained on outputs of base classifiers (which already have good bias variance tradeoff)

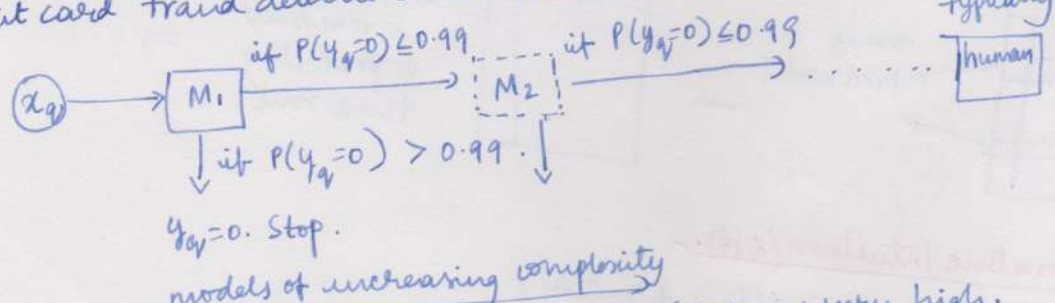
output of metaclassifier $= h'(h_1(x), h_2(x), h_3(x) \dots h_m(x))$

→ stacking is used less often than boosting & bagging because of trainTC.

But for competitions stacking is almost a must.

Cascading classifiers (5.17):-

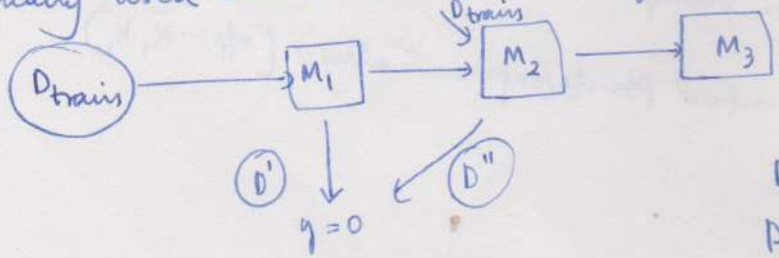
eg:- credit card fraud detection 0 → fraud 1 → not fraud



models of increasing complexity

→ typically used when cost of making mistake is very high.

→



$D_{train} \to M_1$

$D^2_{train} \to M_2$

$D^2_{train} = D_{train} - D'$

$D^3_{train} = D^2_{train} - D''$

(data leakage should be taken care of)

Interview & Reinforcing Questions – Ensembles :–

→ ~~homogeneous~~ heterogeneous :- different base learners

homogeneous :- same type base learners.

Introduction: Featurization & Feature Engineering :– (1.1)

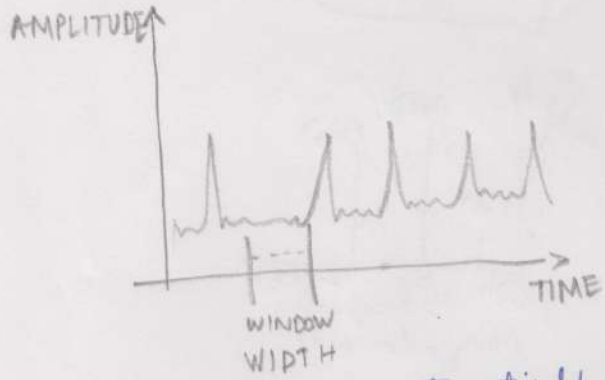
→ Most imp aspect of ML : FE

→ featurization : converting data to numerical.
FE : Using numerical data well in ML.

→ text data : BoW, TFIDF, Avg W2V, TFIDF W2V
categorical data : OHE, Mean Response Rate, domain specific.

→ time series :- heart rate, ecommerce, speech wave patterns, stock prices etc,

Moving window for time series data (1.2):–



→ window width is domain specific.

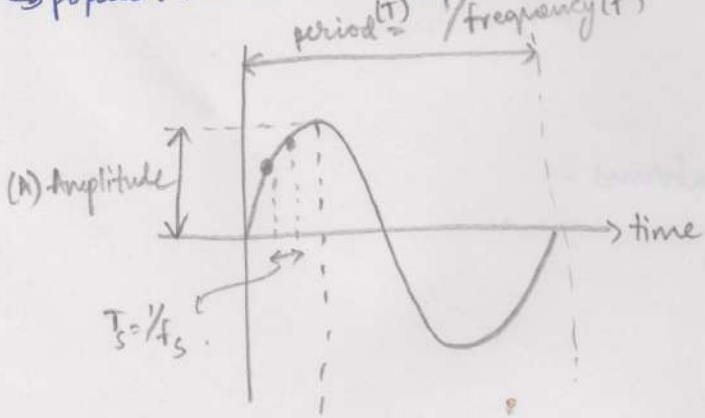
→ take window & calculate
(i) mean, std-dev
(ii) median, quantiles
(iii) max, min, max-min
⋮
} studied in signal processing

→ deep learning can automatically learn correct featurizations instead of us working on it. [SOMETIMES]

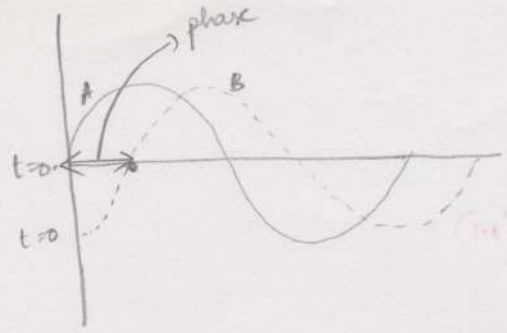
→ window width is a hyperparameter.

Fourier Decomposition (1.3):–

→ popular method to represent time series data.

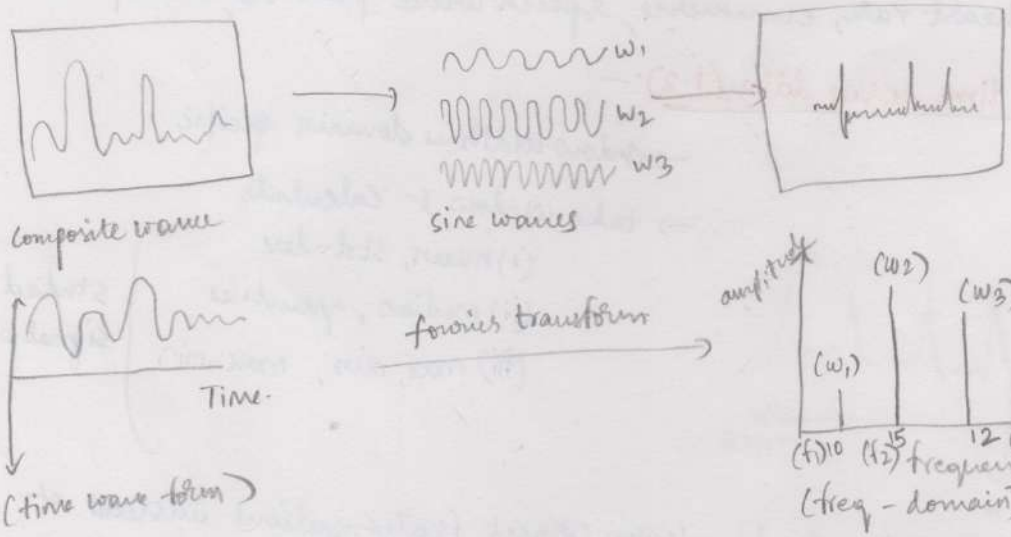


period = time for one cycle/oscillation
→ if it makes one oscillation per sec, freq = 1 Hz.



→ patterns can be seen. ecommerce 12AM less sales → daily patterns.
weekly patterns etc;

fourier transform :- a repeating pattern can be broken down into sum of sine waves.



→ freq domain is useful when we have repeating patterns.

→ We get feature vectors from freq-domain $\left\{ \begin{matrix} f_1, a_1 \\ f_2, a_2 \\ f_3, a_3 \end{matrix} \right\}$

$$\underbrace{\boxed{f_1 \mid f_2 \mid f_3 \mid a_1 \mid a_2 \mid a_3}}_{x_1}$$

fourier representation of time series data.

→ we can use wavelets for irregular waveforms.

Deep Learning features :- (1.4)

→ deep learning : give lots of data. It will almost automatically learn the best featurizations for time series data
These features are called deep learnt features

This is a big reason behind google now, siri, song-recom being better now. They use deep learnt features.

→ Generally people go for deep learning if dataset is minimum 1,00,000.

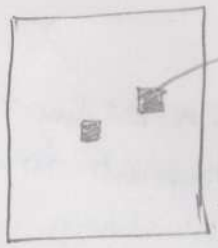
→ if there is enough data, DL will always perform better than ML.

Image Histograms (1.5) :-

→ convolutional neural networks are used a lot now adays.

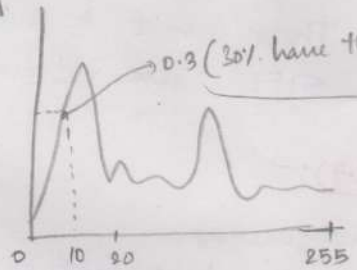
→ two types. (i) color histograms
(ii) edge histograms

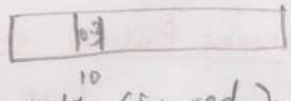
(i) color histogram :-



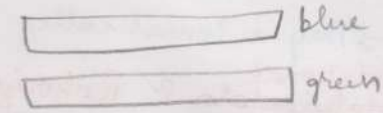
→ each pixel has (R,G,B) coordinates

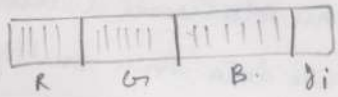
↳ take red





vector (for red).





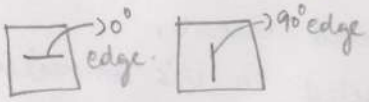
→ if task is to predict sky or not, some val in blue vect is high.

→ we can try to find face using skin color.

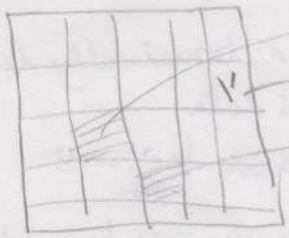
→ color hist cannot detect shapes.

(ii) edge histograms :- edge = regions separating two colors.

→ there are several algorithms in image processing to detect edges.

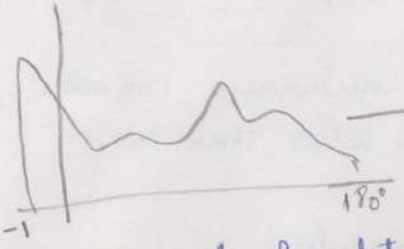


→ check what ° edge is there after breaking down image



→ if edge find angle.
→ multiple edges, use the most powerful edge.
→ no edge. feat − 1.

each region will have edge value & edge angle



→ vector.

→ not the best for face detections. There are algos like haar features to detect faces

Keypoint (SIFT) (1.6):-

SIFT: Scale Invarient Feature Transforms.

→ Useful for object detection.
→ It will find keypoints (edges) and it will create a 128 dim vector for it.
→ It will try to match these keypoints to the keypoints from the database.
→ Scale Invariance because there won't be much change in keypoints even if the image is scaled. SIFT is also rotational invariant.

Deep Learning feature: CNN (1.7):-

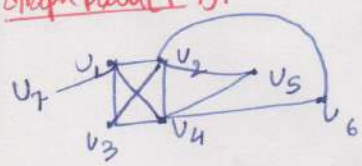
→ CNN beat the best featurizations.
→ They automatically create the best features.

Relational ~~Relatic~~ data & featurizations (1.8):-

from relational databases

→ take data from multiple tables, use domain knowledge to come up with new features.
→ SQL can be used to procure data from relational table.

Graph Data (1.9):-



task: recommend friends to user U_i

→ U_i / U_j → 1/0 1 if connected. else 0.

we can recommend those connected to U_i's connections & not connected to U_i

→ A feature could be number of mutual friends b/w two unconnected people
Another could be # of paths b/w two connections (not just 1 edge paths)

graph theoretic / graph based features

Indicator variables (1.10):-

→ technique of feature engineering

→ used in case studies.

ex:- height is a feature (real valued).

$h > 150 \rightarrow 1$
$h \leq 150 \rightarrow 0$ } problem specific/domain specific. } → indicator variable

ex②:- categorical feature :- country.

```
if (country = India) or (country = USA):
        return 1
else
        return 0.
```

→ widely used in statistical problems.

feature binning (1.11):-

→ logical extensions to indicator variables.

```
→ if h < 120 :
      return 1
  if h < 150 and h ≥ 120
      return 2
  if h < 180 and h ≥ 150
      return 3
  if h > 180
      return 4
```

} multiple thresholds instead of just 1.

→ aka feature bucketing.

→ we can use decision trees to find the right threshold.

→ Binning is a good idea when we expect a discontinuity at the cut points

ex:- boiling point of temp, legal driving age etc.

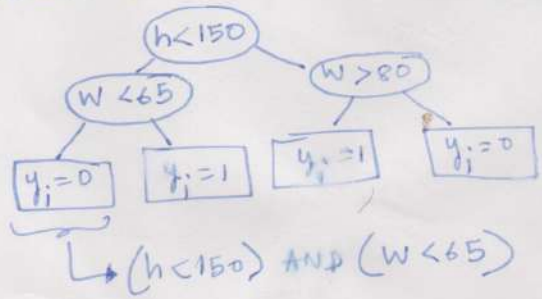
Interaction Variables (1.12):-

task :- height, weight, hair length, eye color → gender.

eg① :- $(h < 150)$ and $(w < 60)$ → logical 2 way interaction feature.

eg② :- $h * w$ → mathematical 2 way interaction feature.

→ Decision trees can be used.



Mathematical Transforms (1.13):-

→ x → single feature

$\log(x)$, e^x, ...

x^2, x^3, x^4, ...

$\sin(x)$, $\cos(x)$, ...

→ best transform is problem specific.

eg:- if x is ~~power law~~ log normal distribution, $\log(x)$ roughly converts it to gaussian.

Boxcox is used to convert power law & log normal into gaussian.

Model Specific featurizations (1.14):-

→ DT's may not work very well when for ex:- there are linear combin-ations of features to get target value. Better to use linear models.

→ for interactions b/w features, it's better to use D.T's as they capture that well.

Feature Orthogonality (1.15):-

→ The more different/orthogonal the features are, the better the model would be.

ex:- $f_1 \xrightarrow{\text{corr}} y$ $\quad f_2 \xrightarrow{\text{corr}} y$ $\quad f_3 \xrightarrow{\text{corr}} y$.

$$\underbrace{f_1 \xrightarrow{\text{highly corr}} f_2/f_3}_{\text{impact of } f_1, f_2, f_3 \text{ would be less.}}$$

$$\underbrace{f_1 \xrightarrow{\text{not corr}} f_2/f_3}_{\text{impact is much better}}$$

→ New features should not be much correlated to existing features.

How?

ex:- $\underbrace{f_1 \quad f_2 \quad f_3}_{X} \quad \underbrace{y}_{Y} \quad \underbrace{\tilde{y}}_{M(X)}$ $\qquad e_i = \tilde{y}_i - y_i$

if new feature f_4 is correlated with e_i,

$\underbrace{f_1 \quad f_2 \quad f_3 \quad f_4}_{X} \quad \underbrace{y}_{Y} \quad \underbrace{\hat{y}}_{M'(X)}$, $M'(X)$ will be closer to y

similar to what's done in GBDT.

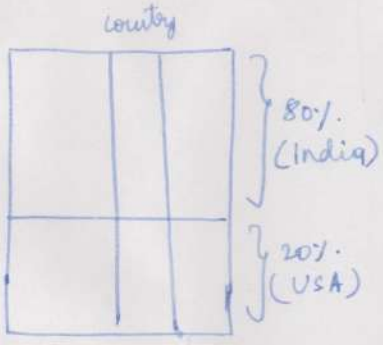
Domain Specific featurizations (1.16):-

→ imp to research existing featurizations made by specialists -

Feature Slicing (1.17):-

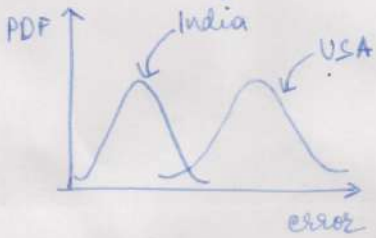
eg:- $\underbrace{h, w, hl, ec, \text{country}}_{x} \longrightarrow \underbrace{\text{gender}}_{y}$.

India USA



Model may perform better on Indians & may perform worse on USA.

After testing



$\longrightarrow D \rightarrow D_{India} \longrightarrow M_1$

$\rightarrow D_{USA} \longrightarrow M_2$

(if there's sufficient data use appropriate model based on country feature.)

Slicing the data, conditions :-

(i) cat 1 & cat 2 must show different results.

(ii) sufficient data.

eg:- Amazon desktop & mobile users.

Callibration of Models (2.1):-

→ given a model, we sometimes need the actual probability $P(y_q = 1/0 \mid x_q)$. In naive bayes, its not the actual probability. It's proportional. The process of getting this is called callibration.

Why do we need ~~logloss~~ callibration? :-

ex:- if 2 class classification :-

$$\underbrace{\text{log loss}}_{} = -\frac{1}{n}\sum_{i=1}^{n} y_i \log P_i + (1-y_i)\log(1-P_i)$$

$P_i = P(y_i = 1 \mid x_i)$

This depends on this value.

→ Predicted probabilities that match the expected distribution of probabilities for each class are referred to as calibrated. It's a must if we want a probabilistic class output ie, $P(y_i \mid x_i)$ as calibration corrects these probabilities.

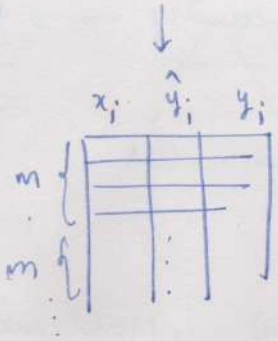
(104) → calibration plots are used in binary classification cases. For multi class, we can convert them into one vs rest

Calibration plots :- (2.2) :-

→ $D_{train} \longrightarrow f$

$D_{cv} \longrightarrow x_i, y_i \longrightarrow f \longrightarrow \hat{y}_i$

$x_i, \hat{y}_i, y_i \qquad y_i, \hat{y}_i \in \{0, 1\}$



sort them in increasing order of $\hat{y}_i$

$\Rightarrow \hat{y}_1 \le \hat{y}_2 \le \hat{y}_3 \cdots$

break into chunks of size m

average $\hat{y}_i$ & y_i for each chunk.

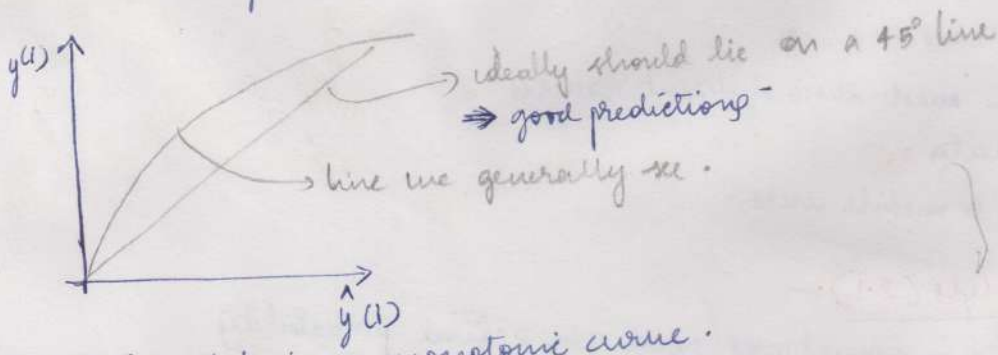
$\Rightarrow \hat{y}^{(1)}, y^{(1)}, \hat{y}^{(2)}, y^{(2)} \cdots$

→ number of chunks $= \dfrac{\text{len}(D_{cv})}{m}$

→ $D_{calibration} = \{\hat{y}^{(i)}, y^{(i)}\}$

$\hat{y}^{(i)}$ → avg $f(x_i)$

$y^{(i)}$ → $P(y_i = 1 \mid x_i)$



→ calibration plot is a monotonic curve.

→ If dataset is imbalanced, we can balance training set but keep cv dataset unbalanced.

→ The idea of calibration is to get calibrated probabilities. Calibration leads to better predictions

→ Brier score loss is a numerical representation of calibration curve. $\dfrac{\text{pred}^2 - \text{org}^2}{m}$

Platt's Calibration / Scaling :- (2.3)

$$D_{calib} = \{\hat{y}^{(i)}, y^{(i)}\}$$

guess this — predict this.

→ idea is to use a modified sigmoid on $(\hat{y}^{(i)})$. See The graph on sklearn page.

$$\rightarrow \quad \underbrace{P(y_q = 1 \mid x_q)}_{y^{(i)}} = \frac{1}{1 + \exp(A \underbrace{f(x_q)}_{\hat{y}^{(i)}} + B)}$$

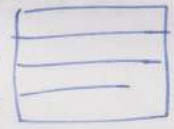
D_{calib} → use this to find best A & B by solving optimization problem

→ calibration / Platt scaling works iff data looks like a sigmoidal function. Fix for this is to use isotonic regression. Isotonic regression is the most used calibration technique.

→ Data → model → on CV data

pred label, orig label

sort by this



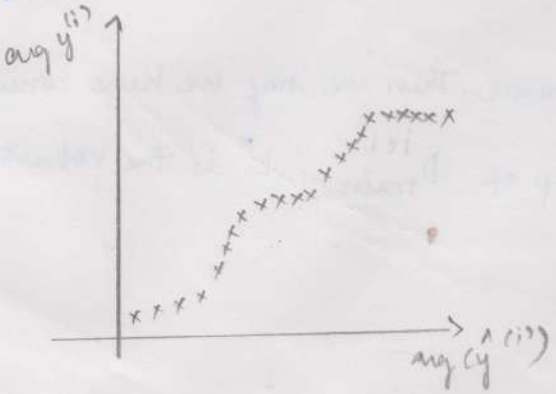
D_{calib} $\{\hat{y}^{(i)}, y^{(i)}\}$ { break into chunks & avg

↓

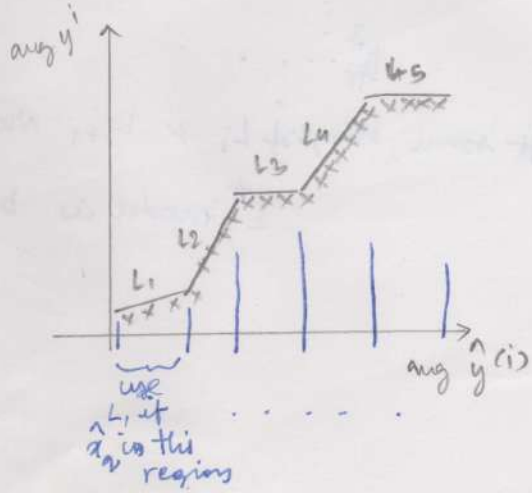
plot. If sigmoid, use platt scaling to get A & B. Else use isotonic regression

Isotonic Regression (2.4) :-

→ Even when calib plot does not look like sigmoid function.



→



→ tries to approximate using bunch of lines.
called piece wise linear models.

→ core idea is to solve multiple optimization problems b/w

$$\underbrace{a_1, a_2}_{L_1}, \underbrace{a_3}_{L_2}, \underbrace{a_4}_{L_3}, \ldots$$

→ looks like extended linear regression.

(m, b_1) $(m_2\ b_2)$ ....

m → slope, b_1 → intercept.

→ Isotonic Regression needs many more points than with platt scaling.
→ If there are less points & not sigmoid, we use isotonic with fewer buckets.
→ whenever we have log loss, we can use calibration to get better results.

RANSAC (Random Sampling Consensus) (2.5):-

→ can we build a robust ML model that is not impacted by outliers.
→ used a lot in image processing & computer vision

Given D_{train},

step①:- D_0 = random sample from D_{train}

D_0 ↓ model (ex:- linear regression) → line (L_0)

step②:- compute outliers based on L_0 on whole D_{train}.

ie., high $(y_i - \hat{y}_i)$

O_0 = {outliers}

step③:- $D'_{train} = D_{train} - O_0$

step④:- find outliers from D'_{train} by building random sample model on D'_{train}

$D^2_{tr} = D'_{train} - O_1$

$\vdots$

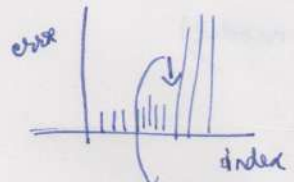
$D^3_{tr} \ldots$

→ At some stage if L_i & L_{i+1} are almost same. Then we say we have converged.

L^* model is built on top of D^{i+1}_{train}. L^* is the robust model

→ RANSAC runs model multiple times & only takes care of extreme outliers. Regularization works within the model and takes care of noisy points as well.

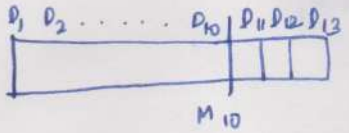
→ one method to find outliers is $[(Q_1 - 1.5 * IQR), (Q_2 - 1.5 * IQR)]$

→ 

inflection point/elbow point - can be used to determine outliers as well.

→ For classification we can use log loss instead of squared loss.

Retraining Models Periodically (2.7) :-



M_{10} on D_{11} → AUC → 0.8

M_{10} on D_{12} → AUC → 0.81

M_{10} on D_{13} → AUC → 0.6 → sharp decrease → Indications to retrain model ~~on~~ till D_{13}.

→ How to determine if we have to retrain:

→ if model performance is dropping.

→ dataset itself is changing. (non stationary data seen in stock market)

→ retrain regularly depending on cost

→ Hyperturning needs to be done if new data is being added.

→ scikit-learn estimators have partial fit option that lets us incrementally train data.

SGD classifiers is used a lot.

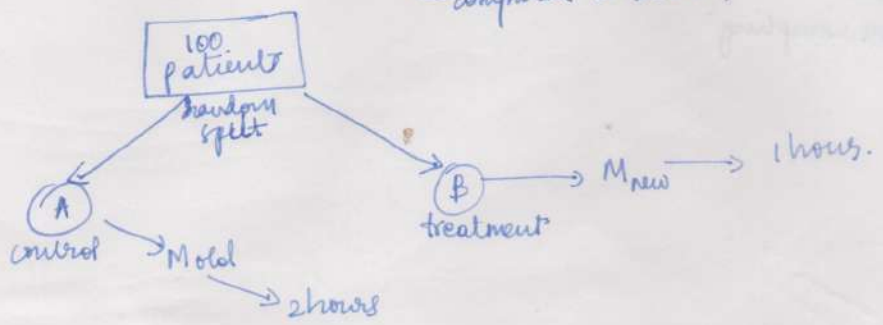
A/B Testing (2.8) :-

→ aka Bucket testing / Split Run / controlled experimenting.

→ imp part of ML & Science.

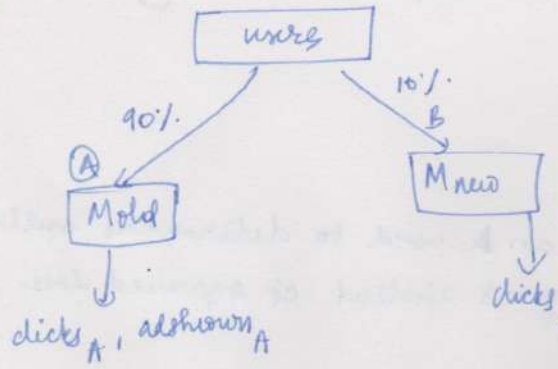
ex:- medicine for cold :- M_{old}, M_{new}, no drug (placeb)

compared to both.



→ use case in machine learning.
Model to show ads.

M_{old}
M_{new}



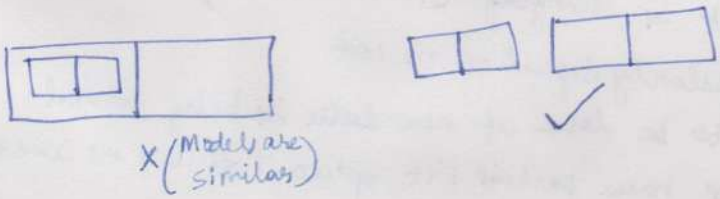
(90% - 10% because can't risk 50% getting bad new model)

$$CTR_A = \frac{clicks_A}{adsshown_A} \qquad CTR_B = \frac{clicks_B}{adsshown_B}$$

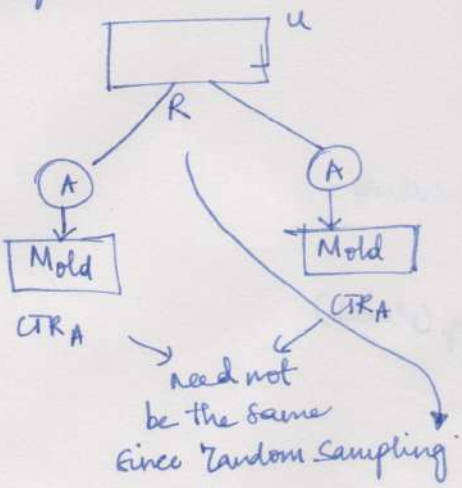
if $CTR_A < CTR_B$ ⇒ model B is better.

→ initially start with 95-5%, then move to 90-10% if model keeps performing better . . .

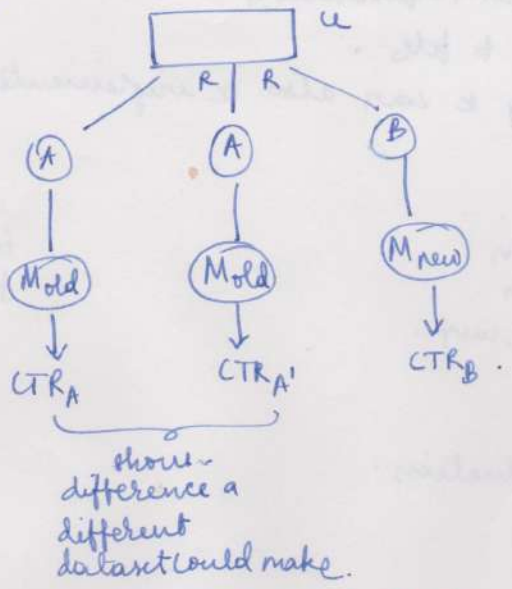
→ Better to use confidence interval instead of directly comparing CTRs.
if confidence interval of CTR_B & CTR_A do not overlap & $CTR_B > CTR_A$, then we can conclude



A-A Testing :-



→ Typically used like this (A-A-B Testing)



Data Science Project Life Cycle (2.9):-

① Understand Business requirements :- define problem, customers.

② Data Aquisition :- ETL (Extract transform & load), tools used : SQL [DB, DW, log-files, Hadoop/Spark]

③ Data Preparation :- Cleaning, preprocessing.

④ Exploratory Data Analysis :- plots, visualization, hypothesis testing, slice & dice data

⑤ Modeling, Evaluation, Interpretation.

⑥ Communicate Results : Clear & Simple, 1-pager, 6-pagers.

⑦ Deployment : Engineering

⑧ Realworld testing :- A-B testing

⑨ Customer / Business buy in

⑩ Operations : retrain models, handle failures, process.

⑪ Optimization :- Improve models, more data, more features, optimize code

Productionization & Deployment of Models (2.10):-

→ Placing models in the real world.

① Using sklearn model persistence :- model is stored to HDD. When needed it can be loaded into RAM.

→ Model can be stored as a string or as a file

→ joblib > pickle for models with large numpy arrays internally (preferred way).

② Custom implementation approach :- when requirements like low latency.

(i) store all parameters of model to file.

ex:- log reg :- store weights & can also be implemented in C/C++.

hash table can be used. can be stored in cache for high throughput use cases.

10-30 x faster than joblib.

Live Session : Prod & Deployment :-

model trained, feature engg, model evaluation.

↓

[M]

↓

productionization (deploy into real world).

life cycle :-

① [raw data] (csv, text file, db, dw, hive, hadoop, spark)

↓

[Process data] (SQL, Python, Java ....)

↓

[featurized Data] → [Model]

ETL Stage, Data Pipeline (extract, transform & load)

Performed by data engineer

→ frequency of ETL pipeline matters. That decides how often model needs to be trained.

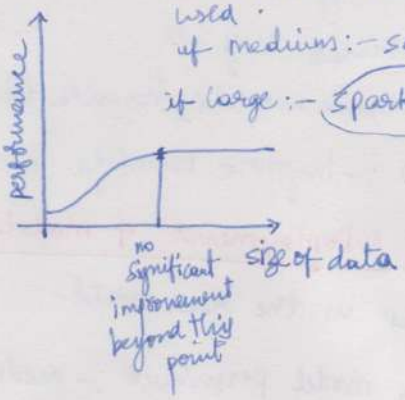
② [Featurized Data] → Train model ex:- LR

(volume of data, (no need to train on all points))

if small dataset & can be fit in RAM :- sklearn, rstudio can be used.

if medium :- sampling, HDD can be used

if large :- SparkML



distributed computing platform

deployment = how well ML model fits in the software.

→ Flask can be used to deploy web API

V-C-dimension (2.16) :-

→ can we tell how powerful a class of models is

theoretical part. statistical ML, computational learning theory.

ex:- linear models, RBF-SVM, boosting algo etc.,

→ practical way is to train on bunch of datasets & measure performance

→ statistical Machine Learning uses VC Dimension to get a numerical value of it.

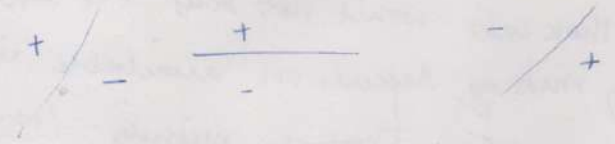
VC = Vapnik Chervonenkis

also created SVM

VC Dimension is not used a lot in applied ML

→ Assume we have a linear model (line, hyperplane)

if there are only 2 points, irrespective of the way they're positioned we can find hyper plane that seperates them (this is called shattering)



if there are 3 points, then also we can shatter (+, −, + X)

⋮

there exists atleast one configuration of 4 points s.t there 4 pts can't be shattered using a linear model.

∴ VC dimension of (linear model) = 3 = max num of points that can be shattered using linear model.

~~The~~ VC dimension: if we can find a set of n points & all 2^n label placements can be classified, VCD = n

→ Radon's theorem can be used to prove.

→ VC Dim (RBF-SVM) = ∞. Does not imply RBF SVM is the best when applied.

Feature Engineering Course:-

→ We can extract number & label from the mixed type (ex:- A12 → A, 12)

→ An Ordinal variable is a categorical variable which values have an intrinsic order.

Variable characteristics:-

→ Missing Data Causes: Lost/doesn't exist/not found/not identified. Missing Data can have significant effect.

→ Missing data is incompatible with sklearn. Missing data imputation may ~~tot~~ distort variable distributions. Affects all ML models

→ 3 MD mechanisms

(i) MCAR : MD completely at Random

(ii) MAR : MD at random

(iii) MNAR : MD not at random

MCAR:- Probability of being missing is same for all observations. No relationship b/w MD & other values within the dataset. Disregarding those cases would not bias the inferences made.

MAR:- Prob of being missing depends on available info.

MNAR:- There is a reason why they are missing. (ex:- no of sports classes among depressed patients)

→ The number of different labels is called cardinality.

→ Cardinality impacts: Uneven dist b/w train & test sets. Overfitting in tree based algos. Operational problems.

→ Some ~~mode~~ labels only appear in train set: Overfitting. Some appear only in test set: model won't know how to interpret them.

→ Cardinality & Overfitting:- Variables with too many labels tend to overfit tree based algos. More labels more noise. Reducing cardinality improves performance.

→ Strings should be encoded as numbers if want to be used in scikit-learn.

Business Objectives & Constraints (1.3)

→ cost of misclassification is very high. We need a $P(Q_1 \text{ similar to } Q_2)$. We can set a threshold.

→ No strict latency concerns.

→ Interpretability is partially important.

Mapping to an ML problem (1.5):-

→ we need 0 to 1 prob. Logloss is the best metric.

KPI (Key performance indicator.

→ if we want probabilities :- logloss.

if classes are balanced :- Accuracy.

if classes are imbalanced :- if we are concerned about TP : precision

" " " " " FN/FP : Recall

F1 score is balance b/w precision & recall.

if we are concerned about both classes ROC AUC.

Mapping to an ML Problem (train-test split) (1.6):-

→ random 70-30 / 80-20 split.

→ timestamp would have been better so that we could do temporal split.

If we sort by timestamp & split, we simulate production.

ML Problem formulation (2.4):-

→ As P_{ij} for ground truth class ↓, Multiclass log loss ↑

$$= \frac{-1}{n} \underbrace{\sum_{i=1}^{n}}_{\text{points}} \underbrace{\sum_{j=1}^{c}}_{\text{classes}} y_{ij} \log P_{ij}$$

$y_i = [0 \mid 0 \mid 1 \mid 0 \mid \cdots \mid 0]$

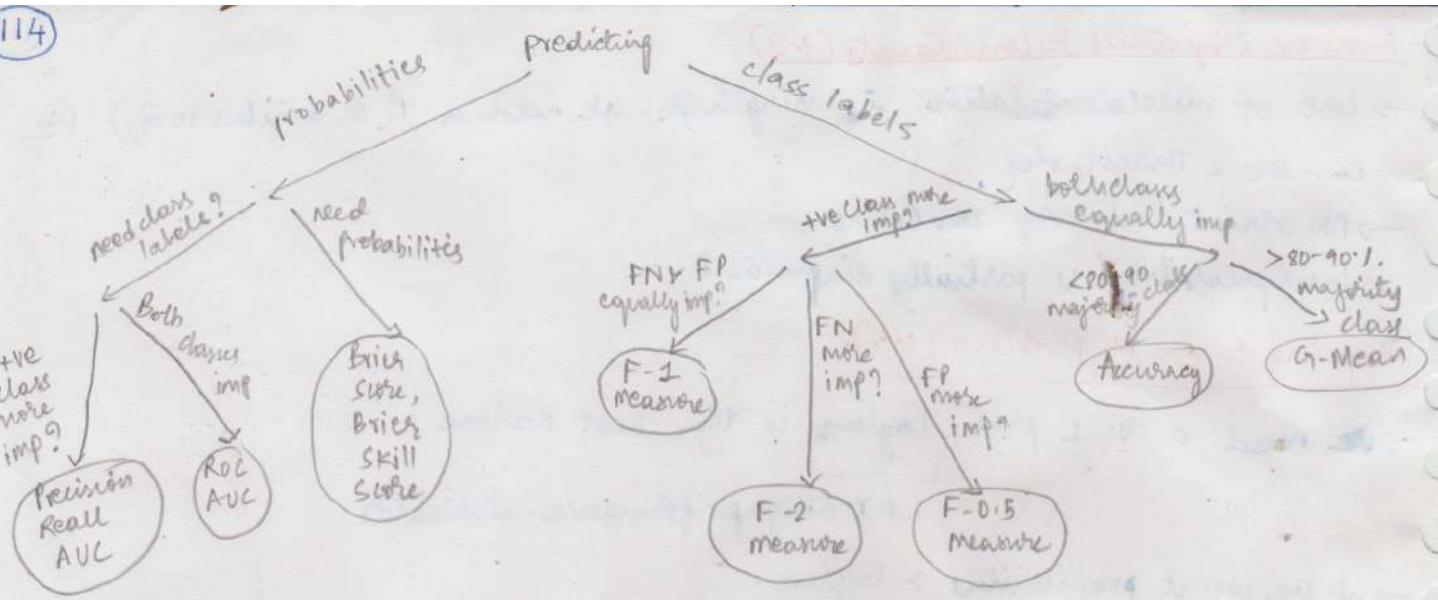
→ Multiclass log loss will penalize even small errors.

→ log loss $\in [0, \infty)$

→ loss can also be used as a performance metric.

→ Whenever we have probabilities under consideration, good to use log loss as it helps penalize errors.

→ We can't use accuracy metrics for optimization because they are not differentiable.



Train - test formulation (2.5) :-

→ temporal nature : changes over time.

→ We can use dummy classifier for random model

One feature univariate analysis (2.8) :-

→ label encoding (assigning a number to a category) is bad because it gives an order to the categories when there's no order. Use when ordering is not important.

Univariate Analysis: Variation (2.10):-

→ having stable feature = model doesn't change much with slight change in dataset.

→ if one hot encoder is used and new categorical variable is encountered in test/CV, model just ignores that feature.

Univariate analysis: Text (2.11):-

→ response coding for text:-

text → $w_1 w_2 w_3 \cdots w_K$

we need $P(y=1 \mid text)$

we use naive bayes approach

$$P(y=1 \mid w_1 w_2 w_3 \cdots) \propto \prod_{i=1}^{K} P(w_i \mid y=1)$$

$$\frac{y=1 \text{ AND } w_i + \alpha}{w_i + 9\alpha}$$

laplace smoothing

$$\rightarrow \log(P(y=1 \mid w_1 w_2 w_3 \cdots w_K)) \propto \sum_{i=1}^{K} \log(P(w_i \mid y=1))$$

because value too small.

⇒ then take exponential of this (antilog)

Data preparation (2.12):-

→ OHE results in high dimensional data (better for LR, SVM etc.)

→ Response coding relatively low number of dimensions (better for KNN, Dtrees, RF etc.)

Baseline Modeling (2.13):-

→ Read comment on vid about lowercase, uppercase gene importance.

Random Forest with One Hot Encoding (2.18):-

→ cv = "prefit" to be used when the data you use for calibration is assumed to not have been used for model training.

Random Forest with Response Coding (2.19):-

→ Random forest with only one tree might overfit the data because it is similar to a decision tree. So to deal with OF increase #estimates

** read comment for feature importance

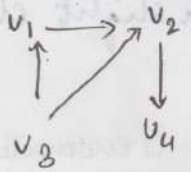
3.2: Overview of Graphs:-

$G = \langle V, E \rangle$

↳ edge $= (v_i, v_j)$

undirected = fb

directed = instagram, twitter etc.,



$v_3 \to v_4$: $v_3 \to v_2 \to v_4$: path of length 2
: $v_3 \to v_1 \to v_2 \to v_4$: path of length 3.

→ This dataset has no cartesian plane significance.

Business constraints & metrics (3.5) :-

→ Precision@topk : $v_i \to \{\underbrace{v_1, v_2, v_3, \ldots v_k}_{\text{how many of topk are correct}}, \ldots v_n\}$

→ Since logloss has no upper bound we haven't used it.

→ v_i & v_j (all other points) are passed it it will tell probabilities.

→ using precision & recall for more interpretability

EDA : Basic Stats :- (3.6)

handshaking lemma:

(i) sum of indegree + outdegree = 2* number of edges.

(ii) sum of indegrees of all nodes = sum of outdegrees of all nodes
= number of edges in directed graph.

Since the sum is same, their average will also be the same.

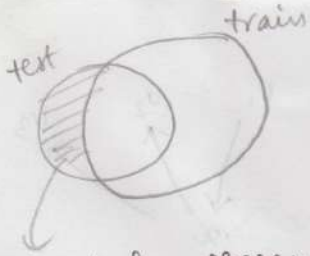
degree = indegree + outdegree.

strongly connected graph :- every vertex reachable from every other vertex.

→ read more about matrix factorization.

EDA : Train & test split (3.9) :-

→ Data is actually temporally changing. we have data from each day in production. So we would have time as well. Since we dont have that here, we use random split.



users only present in test.
⇒ partial cold start problem.

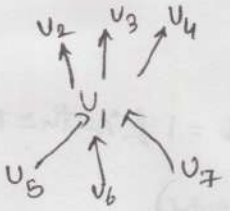
Feature Engineering (3.10) :-

jaccard ~~distance~~ index $= j = \dfrac{|X \cap Y|}{|X \cup Y|}$. More jaccard ~~distance~~ index = more chance of edge b/w X & Y.

jaccard dist = (1 − jaccard index)

→ cosine distance (Otsuka Ochiai coefficient) for sets instead of vectors.

$$= \frac{|X \cap Y|}{\sqrt{|X| * |Y|}}$$ → Same as jaccard. large for closer nodes.



v_1 → ~~follower set~~ aka followee set $= \{v_5, v_6, v_7\}$

follower set $= \{v_2, v_3, v_4\}$

Page Rank (3.11) :-

→ A node is important if it is the destination node for many nodes.

the quality of these nodes also matters.

damping factor in page rank = probability that any step an imaginary surfer will continue clicking.

Connected Components (3.12) :-

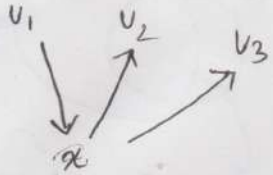
weakly connected component = all components are connected by some path ignoring direction

→ weakly connected components in this dataset = community.

→ If v_i & v_j are in SCC, shortest path helps. If they are not connected but in a WCC, then prob will be a bit higher.

(118) Adamic / Agar Index (3.13) :-

$$A(x, y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\log |N(u)|}$$



$N(x) = \{u_1, u_2, u_3\}$

common elements of x & y with many neighbours less significant than common elements of x & y with less neighbours.

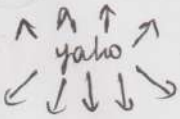
Katz Centrality (3.14) :-

→ Katz Centrality is used to measure relative degree of influence of an actor within a social network. Generalization of Eigen Vector Centrality.

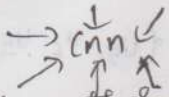
HITS Algorithm (3.15) :-

HITS = Hyperlink Induced Topic Search. aka hubs & authorities

hub = more outlinks ex:- yahoo in old internet days.



authority = more inlinks ex:- cnn, bbc etc.



→ HITS gives a hub score & authority score to each node.

→ its an iterative rule: to begin with all have hub = 1 & auth = 1.

$$auth(p) = \sum_{i=1}^{n} auth(i)$$ (sum of auth of all neighbours)

$$hub(p) = \sum_{i=1}^{n} hub(i)$$ (sum of hub scores of pages that point to it)

Scores are normalized each time to prevent from going to ∞

→ if a person has lots of followers that could mean high auth score.

if a person follows lots of people that could mean high hub score.

Weight features (3.18) :-

probability of these two knowing each other

<

probability of these two knowing each other.

inweight

$$W(v_i) = \frac{1}{\sqrt{1 + |x|}}$$

x = set of vertices linking into v_i

outweight also used.

4.2 : Business Constraints :-

constraints : Latency.

Interpretability not so important.

Relative errors :- predicted : 100, actual = 102

↓

absolute errors = 2, %.age errors = 2%

predicted : 10, actual = 12

↓

absolute error = 2, %.age errors = 20%

this is important.

This %.age error must be kept as low as possible

4.4 : Dask Dataframes :-

Loads small chunks of the dataset at a time into the RAM.

NYC Taxi :-

4.21 : Weighted Moving Average :-

→ Weighted Moving Average seem to carry information from the past and they have information of general trend.

StackOverflow :-

Performance Metrics (5 5) :-

→ Mean FScore = microaveraged F-1 score = $\dfrac{2\, Precision_{micro}\, Recall_{micro}}{Precision_{micro} + Recall_{micro}}$.

$$Precision_{micro} = \frac{\sum_{k \in C} TP_k}{\sum_{k \in C} (TP_k + FP_k)}$$

$$Recall_{micro} = \frac{\sum_{k \in C} TP_k}{\sum_{k \in C} (TP_k + FN_k)}$$

→ weightage of a tag is more if it is repeated more number of times.

→ Macro average f1 score doesnt take the weightage into consideration. It just average f1 score of all classes.

Hamming Loss (5.6) :-

$$\text{Hamming Loss}(\hat{y}_i, y_i) = \frac{1}{|D|}\sum_{i=1}^{|D|} \frac{XOR(\hat{y}_i, y_i)}{|L|}$$

$|D|$ = number of samples.
$|L|$ = number of labels (distinct)

→ contribution to the error is more if difference b/w pred & actual is more.
→ Its the fraction of wrong labels to the total number of labels
→ $XOR(\hat{y}_i, y_i)$ is not a binary string here. Its the number of different predictions

$y_i = 1011101$ $\hat{y}_i = 1001001$ → ~~hamming dist~~ xor $= 2$

Data Modeling (Multi Label Classfn) (5.10) :-

→ to convert multi label to simples probs there are various approaches.

(i) Binary relevance :- each class is binary classfn (extensively used)

x	y_1	y_2	y_3
x_1	0	1	1
x_2	1	1	0
x_3	1	0	1

(ii) Classifier Chains :- each class is passed as training feature to next class training ....

(iii) Label powerset :- unique combinations of tags is considered as one tag. converts multiclass to multi label.

(iv) Adapted Algorithm :- Basically adapts the algo to perform as multi label instead of trying to convert it to multi class.
ex :- multiKNN takes neighbours tags into consideration etc,

→ All these are present in sklearn library (# uses dense matrices)

→ when using binary relevance, we need to make sure base model have low latency. If there are too many labels, we can go with adapted algo or deep learning.

Data Preparation (5.11):-

→ we can take only a subset of features based on frequency.

Sampling Data & tags (5.15):-

→ to increase weightage of title, we can just repeat title multiple times.

Unsupervised Learning :-

What is clustering (1.1):-

classfn & regressions :- $\{x_i, y_i\}$

but in clustering, no y_i. only $\{x_i\}$

$$D = \{x_i\}$$

task: group / cluster similar data points.

↓

problem specific.

generally speaking → (i) points are close to each other and (ii) points in diff clusters are far away.

Three major techniques.

(i) K Means

(ii) Hierarchical Clustering

(iii) DBSCAN algo.

→ TSNE does not come under clustering because it's main task is dimensionality reduction & preserving neighbourhood

→ when clusters are too fine & can't fit new data, its overfitting. when everything falls in same cluster (there is only one) its underfitting.

Unsupervised learning (1.2):-

→ clustering often referred to as unsupervised learning

→ y_i is supervising / helping us in supervised learning algos.

No y_i in clustering.

$P = P_1 \cup P_2$ $\quad |P_1| << |P_2|$ → P is semi supervised.

P_1 → (x_i, y_i), P_2 → (x_i)

ex:- protein sequence, web content classfn, speech analysis.

(122) Applications of clustering (1.3) :-

→ more studied in data mining.

eg ① :- ecommerce :- group similar customers based on purchasing behaviours

eg ② :- image processing, grouping similar pixels. ML algos performed on top of this for object detection.

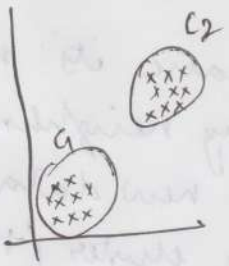
eg ③ :- Amazon food reviews. reviews → +ve, → -ve.

→ Clustering is useful if manual labling is time consuming & expensive

1M reviews
↓
10 k clusters
↓
pick one point from cluster → assign label → ~~10 k for~~ whole cluster gets that label.

Metrics for Clustering (1.4) :-

Ques :- what is a good clustering result?



intra cluster : within a cluster
inter cluster : across/b/w clusters

intra cluster dist is kept small
inter cluster is kept large.

Dunn-Index :- $D = \left\{ \frac{\max_{i,j} d(i,j)}{\max_{k} d'(k)} \right\}$

$d(i,j)$ → dist b/w clusters C_i & C_j → maximal inter cluster dist

should be large ← $\max_{i,j} d(i,j)$

should be small ← $\max_{k} d'(k)$ → inter cluster dist.

if D is high, good cluster.

→ distance b/w centroids of clusters is the standard way.

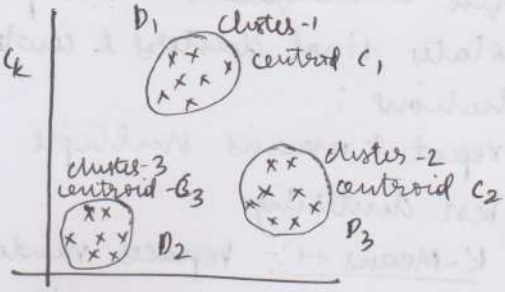
K-Means: geometrical intuition (1.5) :-

→ K = # of clusters

→ K - clusters gives K centroids → $c_1, c_2, c_3, \dots c_K$

K sets → $S_1, S_2, S_3 \dots S_K$

$$c_i = \frac{1}{n} \sum_{x_j \in S_i} x_j \quad \text{(mean point = centroid)}$$



$D_1 \cap D_2 = \emptyset \quad D_1 \cap D_3 = \emptyset$

$D_1 \cup D_2 \cup D_3 = D_n$

Kmeans = centroid based clustering scheme

K-Means: Mathematical formulation (1.6) :-

$\{x_1, x_2, x_3, \dots x_n\} \longrightarrow S_1, S_2, S_3, \dots, S_K$ (K sets)

$c_1, c_2, c_3, \dots c_K$ (K corresponding centroids)

$$\underbrace{\underset{c_1, c_2, \dots, c_K}{\text{arg-min}} \overbrace{\sum_{i=1}^{K}}^{\text{for each cluster}} \overbrace{\sum_{x \in S_i}}^{\text{for all points in cluster}} \|x - c_i\|^2}_{} \quad \text{such that } x \in S_i \text{ and } S_i \cap S_j = \emptyset$$

find c_i such that intra clusles dist is minimized.

The eqn does not say anything about inter clusles dist.

→ This is an NP-Hard problem. So we use approximation algorithms.

The approximated algo is called Lloyd's algorithm.

→ initially we pick random centroids that are updated.

K Means (Lloyd's Algo) (1.7) :-

Step ① :- randomly pick k points from D as $c_1, c_2, \dots, c_K$

Step ② :- for each $x_i \in D$, select nearest c_j ($\text{dist}(x_i, c_j)\ \forall j = 1, 2, \dots K$)

add x_i to set S_j

Step ③ :- recompute centroid stage.

recalculate/update c_js.

$$c_j = \frac{1}{|S_j|} \sum_{x_i \in S_j} x_i$$

Step ④ :- repeat 2 & 3 until convergence.

centroids won't change much. ie, $c_1' - c_1$, $c_2' - c_2$, $c_3' - c_3 \dots$ distances is very small

(124) How to init K-Means (1.8) :-

→ randomized initialization has a problem called initialization sensitivity. which states final clusters & centroids depend on centroids initialized.

→ two solutions :

(i) repeat k means multiple times with diff initializations. Pick best clustering

(ii) K-Means ++ :- replaces random init with smart init

probabilistic approach of picking points.

ⓐ pick first centroid randomly from D

ⓑ $\forall x_i \in D$ create a distribution as follows.

$x_i \longrightarrow dist^2(x_i, \text{nearst centroid})$

$x_1 \longrightarrow d_1$

$x_2 \longrightarrow d_2$

$\vdots$

$x_{n-1} \longrightarrow d_{n-1}$

Then pick a point from these with prob $\propto d_i$

⇒ farther away = more prob of picking

→ choosing probalistically cause outliers affect if we just choose farthest point.

Failure Cases / Limitations (1.9) :-

→ times when clusters are of different

(i) sizes

(ii) densities

(iii) non globular shapes. (non-convex shapes).

How to overcome limitations :-

(i) increase number of clusters. This finds parts of clusters which then need to be put together

K Medoids (1.10) :-

→ After using Kmeans, the centroids may not be interpretable.

ex:- After amazon fine food reviews, the centroid is just a vector (BOW). not interpretable.

→ idea is instead of giving centroid C_j, give point x_j. So we can read review & understand.

i.e., $C_j = x_j \in D$

Partitioning around Medoids (PAM):- Popular algo for k Medoids.

① Init :. Same as k Means ++ ie, probabilistic

② Assignment:- closest medoid

$x_i \in S_j$ if medoid j is closest medoid to x_i. Same step as kmeans.

③ Update/recompute :-

ⓐ swap medoid with non medoid point.

ⓑ if loss decreases keep swap else undo the swap.

$$\text{loss} = \sum_{i=1}^{k} \sum_{x \in S_i} \|x - \underbrace{m_j}_{\text{medoid}}\|^2 \quad \text{(the inter clusters dist)}$$

lots of swaps possible

→ Advantage of k medoids is

→ sim() / dist() matrices can be used. ie, $\sum_{i=1}^{k} \sum_{x \in S_i} dist(x_i, m_j)$

→ more interpretable.

→ kernalization (kernel matrix can be used).

→ another efficient approach is to choose medoid closest to centroid ($O(n)$)

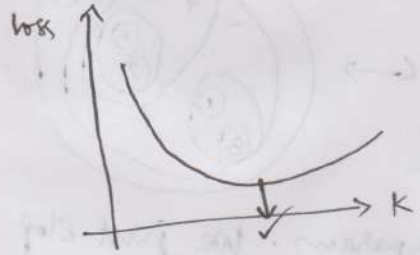
→ k medoids is also an NP hard problem. We use approximation methods to solve them.

Determining the right k (1.11):-

→ domain knowledge.

ex:- if its a good/bad classifn dataset → 2 clusters.
(good bad)

→ in case no domain expert, elbow method aka knee method



→ wrong plot. realistically, it would be like

pick point where reduction in loss starts to drop off. low rate of fall.

(26) Time Complexity (1.13) :-

k-means $= O(nkdi) \approx O(nd)$ if k & i are small. Still linear.

n = # of pts
k = # of clusters
d = dimensions
i = # of iterations

space $= O(nd + kd) \approx O(nd)$

→ K Modes can be used to deal with categorical variables. It is a variation of Kmeans.

→ For a new query point, the time complexity is $O(kd)$

→ Point Biserial Correlation can be used to find correlation b/w continuous & categorical variables. It is similar to pearson's coefficient.

Heirarchical Clustering (2.1) :-

→ Two types: Agglomerative & Divisive
More popular

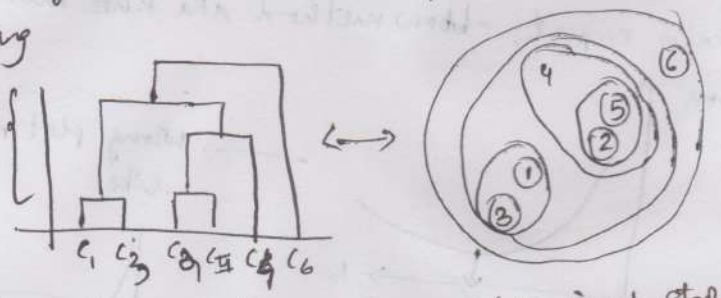
Agglomerative :- starts by assuming each datapoint is a cluster of it's own. Then groups close clusters together iteratively. Grouping is done by some similarity or distance metric.

Divisive :- Works the opposite way. Starts by assuming everything as one big cluster & then divides them. Problem with this is knowing how to divide. Combining easier than dividing.

→ Most imp part of agglomerative is having similarity / distance metric.

Dendrograms :- tree like diagrams that records sequences of merges / splits. Used for agglomerative clustering

y axis is distance / dissimilarity of clusters.



→ # of clusters is not given as hyperparam. We just stop at n clusters stage to get n clusters.

→ we can just continue running to go from n clusters to m clusters unlike Kmeans where we have to re run the model.

→ Heirarchical clustering is computationally expensive.

Agglomerative Clustering (2.2):-

Steps:-

1. compute the proximity Matrix aka distance matrix / similarity Matrix
2. Let each data point be a cluster.
3. repeat:
   merge the two closest clusters
   Update the proximity matrix $\left(P_1 \mid P_2 \mid P_3 \longrightarrow P_1 \mid \underbrace{P_2 \cup P_3}_{\text{form a cluster}} \right)$
   until only one cluster remains.

→ Key op is computing proximity matrix / inter cluster distance.

Proximity Methods: Advantages & Limitations (2.3):-

→ Multiple ways to calculate Inter Cluster Distance.

(i) MIN:- aka Singlelink $sim(C_1, C_2) = \text{MIN } sim(P_i, P_j) \mid P_i \in C_1 \text{ and } P_j \in C_2$
closest similarity b/w two clusters

(ii) MAX:- aka complete $sim(C_1, C_2) = \text{max } sim(P_i, P_j) \mid P_i \in C_1 \text{ and } P_j \in C_2$

(iii) Group Average:- $sim(C_1, C_2) = \dfrac{\sum_{\substack{P_i \in C_1 \\ P_j \in C_2}} sim(P_i, P_j)}{\underbrace{|C_1| * |C_2|}_{\text{Size of clusters}}}$

(iv) Dist b/w centroids:- dist b/w centroids - less popular.

→ These three methods require similarity b/w point. so kernelization can be done.

→ MIN can handle non elliptical shapes as long as dist b/w clusters is not too small.
MIN is sensitive to outliers.

→ In MAX we group clusters which have the smallest distance among maximal distance. MAX is less susceptible to noise & outliers.
MAX tends to break large clusters and merge parts of large clusters with small clusters because part of large clusters is less far away from another clusters than other end of Large clusters. MAX is biased towards globular structures.

→ Group Average compromise b/w single & complete link. Less susceptible to noise and outliers. Biased towards globular structure.

Ward's method: Similar to group average but dist b/w points is squared.

$$dist(C_1, C_2) = \frac{\sum_{\substack{p_i \in C_1 \\ p_j \in C_2}} (dist(p_i, p_j))^2}{|C_1| * |C_2|}$$

Ward's method can be used to initialize K-Means.

→ data must be scaled. Otherwise it leads to misleading clusters.

Time and Space Complexity (2.4):-

Space: $O(n^2)$. We need to store similarity matrix

n = # of datapoints

Time: $O(n^3)$

Atmost n iterations. Each iteration group two clusters ie, update similarity matrix once.

Not very useful when n is large.

Limitations of hierarchical clustering:- (2.5)

→ No math objective that we are solving.

ex:- In K Means we have a loss fn that we can connect to other algos. (Matrix Factorization)

→ MIN, MAX and GROUP have their own disadvantages.

→ Space & Time Complexities are main issues

→ Practical application of hierarchical clustering is to view dendrogram when we don't have an idea of number of clusters.

Density Based Clustering:- (3.1):-

→ seperate into dense regions (clusters) and sparse regions (noise)

DBSCAN: Density Based Spatial Clustering of applications with noise.

How to measure density (3.2):-

→ two params:- MinPts, Eps (hyperparameters)

density at a point P: # points within a hypersphere radius Eps around P.

dense regions:- hypersphere with radius Eps that contains atleast MinPts points.

Core, Border and Noise Points (3.3):-

→ every point in dataset can be defined as core point, border point or noise point.

→ core point: if p has ≥ min points in Eps radius around it.
it always belongs to dense region.

→ border point :- p has < min pts in Eps radius. But p ∈ neighborhood(q)
q is core point.
dist(p, q) ≤ Eps.

→ noise point :- neither core point or border point.

Density Edge and Density Connected points (3.4) :-

→ density edge :- p, q : core points, dist (p, q) ≤ Eps.

→ density connected points :- path exists from p to q with all density edges.
p and q are called density connected points.

DBScan Algorithm (3.5) :-

step ① :- for every point $x_i \in D$, label as either core point, border pt or noise pt.
can be done using range Query

$$S_i = \underbrace{\text{rangeQuery}}_{\text{also seen as k-NN, kd tree.}}(x_i, D, Eps)$$

step ② :- remove all noise points as they dont belong to any cluster.

step ③ :- for each point p, that's not assigned to a cluster,
ⓐ create a new cluster with p.
ⓑ add all points that are densely connected to p into the new cluster.

step ④ :- for each border point, assign it to the nearest core point's cluster.

Hyper parameters : Minpoints and Eps (3.6) :-

→ rule of thumb for minpoints: minpts ≥ $\underbrace{d}_{\text{dimensionality}}+1$
typically minpts ≈ 2*d

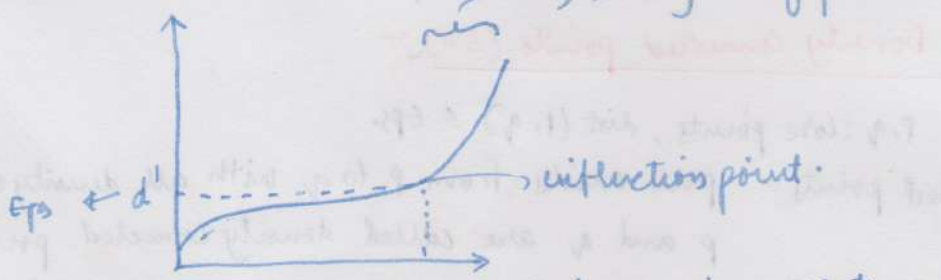
should use larger minpts if dataset is more noisy.
Usually chosen by domain expert.

Eps:- (i) for ex:- let minpts = 4
for each x_i calculate d_i : dist from x_i to 4th NN of x_i

(ii) x_1 d_1
x_2 d_2
x_3 d_3
:
x_n d_n

Sort d_i in increasing order.

(iii) plot them



if d_i is high, chances are x_i is a noisy point.

→ Unlike Kmeans, DBSCAN will figure out the number of clusters.

Advantages & Limitations of DBSCAN (3.7) :-

→ Resistant to noise.
→ Can handle clusters of different shapes very well.
→ Not required to specify number of clusters.
→ only 2 parameters.
→ designed for databases that can accelerate region queries ex:- using an R^* tree

— * —

→ tends to not work very well with varying densities.
→ extremely sensitive to hyper parameters.
→ wont work well with high dimensional data ; curse of dimensionality
→ Quality depends on distance measure.
→ Not entirely deterministic. Border point of two diff clusters could belong to either one based on the order in which data was processed.

Time and Space Complexity (3.8) :-

TC: $O(n \lg n)$ → typical time for range query after using optimized methods like KD tree etc.

SC: $O(n)$ assuming $d << n$.

Problem Formulation : Movie Reviews (4.1) :-

→ Recommends relevant items to a user based on historical data

$$\text{user } U_i \xrightarrow{\text{previous}} I_1, I_7, I_8, I_{10}$$

$$\xrightarrow{\text{model should recommend}} I_j$$

→ Data is not of the form $\{x_i, y_i\}$. instead

$$\begin{array}{c|cccc} & I_1 & I_2 & \cdots & I_j \cdots \quad I_m \} \rightarrow \text{items} \\ \hline U_1 & & & & \\ U_2 & & & & \\ \vdots & & & & \\ U_i & & & A_{ij} & \\ \vdots & & & & \\ U_n & & & & \end{array}$$

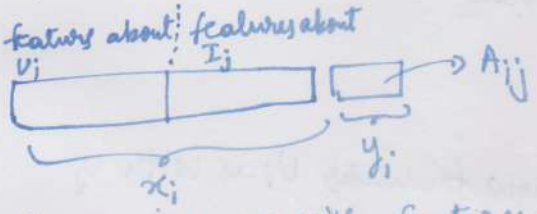
users

A_{ij} → could be rating / boolean val etc ; NaN in case no rating / boolean val.

This matrix is going to be very sparse.

$$\text{sparsity of matrix} = \frac{\#\text{ empty cells}}{\#\text{ total cells}}$$

→ posing as regression prob :



challenge is coming up with features.

⇒ D_{train} : consists of $A_{ij} \neq NULL$

D_{test} : all empty cells of A_{ij}

→ can also be posed as matrix completion problem. (filling up empty values with reasonable values from non empty cells)

→ There is no data imbalance as there is no data labelling.

Content based vs collaborative filtering (4.2) :-

→ two core techniques

(i) content based (ii) collaborative filtering (CF)

→ CF :- U_1 : M_1 , M_2 , M_4

U_2 : M_1 , M_4 , M_3

U_3 : M_1 , $\boxed{?}$ → CF suggests M_4 because U_1 & U_2 liked M_1 & M_4.

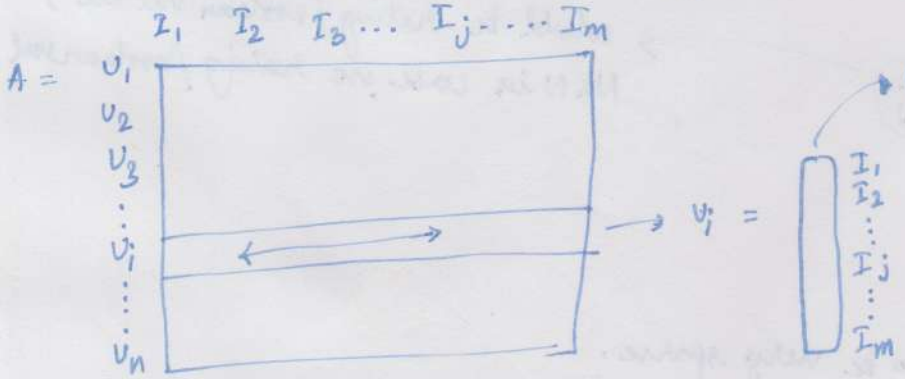
core idea: users who agreed in the past tend to agree in the future.
if this is not true, CF cannot be used.

content based :- acts more like classification/regression problem.
Does not use A_{ij} info.
Instead it finds a way to represent I_j and U_i

ex:- I_j :- action, 90's, english etc., } can be represented as
U_i :- male, 30's, american etc., } boolean vector with y_i as A_{ij}

Similarity Based Algorithms (4.3) :-

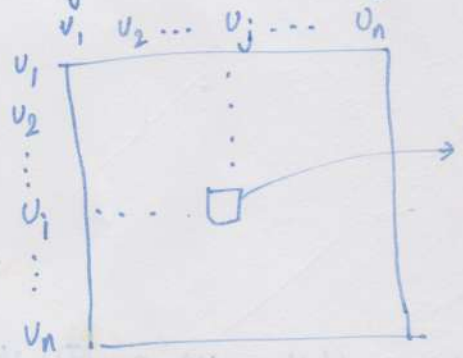
→ two types : user-user based
item-item based → popularized by amazon in 1998...



→ user vector. sparse vector. like bag of words.

$$sim(U_i, U_j) = cosine(U_i, U_j) = \frac{U_i^T U_j}{|U_i||U_j|}$$

$S^U = S^U_{ij}$ = User similarity matrix

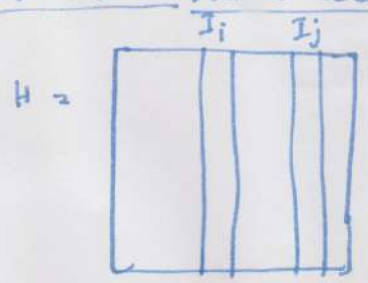


→ how familiar U_i is with U_j

task: recommend new items to U_{10}.
U_{10} is similar to U_1, U_2, U_7. Recommend items liked by U_1, U_2, U_7 that are not yet seen by U_{10}.

→ major prob with user-user sim is preferences change over time. one solution could be to use only recent data. But that makes it more sparser.

→ item-item based recommender system :-



very similar to user-user. Items are used instead.

$sim(I_i, I_j) = cosine(I_i, I_j)$ } we can also use pearson correlation coeff as theyre both similar.

→ Advantage: ratings on a given item do not change significantly after the initial period.

→ preferred approach by ecommerce websites.

→ ex:- U_{10} likes I_1, I_3, I_7

{ } { } { }

prods similar to I_1

repeating elems from these sets are recommended to user.

Rule of thumb :-

→ more users than items (ex: amazon, netflix) item ratings do not change much over time after initial period.

Matrix Factorization: PCA & SVD (4.4) :-

Matrix factorization = Matrix Decomposition.

core idea: decompose matrix into product of other matrices

→ PCA can be interpreted as an example of MF.

→ high dimension → low dimension

X_{nd} → Data Matrix.

$S_{d \times d} = \frac{X^T X}{n-1}$ = co-variance matrix.

→ eigen values: $\lambda_1, \lambda_2, \ldots, \lambda_d$

eigen vectors: $w_1, w_2, w_3 \ldots w_d$

to convert d to d', select top λ's & project onto w_i's.

$$S_{d \times d} = \underbrace{W_{d \times d} \cdot \Lambda_{d \times d} \, W^T_{d \times d}}_{\text{eigen decomposition of } S}$$

$$W_{d\times d} = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ w_1 & w_2 & \cdots & w_d \\ \downarrow & \downarrow & & \downarrow \end{bmatrix} \qquad \Lambda_{d\times d} = \begin{bmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \ddots \; \lambda_d \end{bmatrix}$$

PCA aka Eigen Decomposition of S.

SVD :- Singular Value Decomposition.

PCA can only be performed on square / symmetrical matrix.
SVD can be performed on ~~regular~~ rectangular matrices as well.

$$X_{n\times d} = U_{n\times n} \Sigma_{n\times d} V^T_{d\times d} \longrightarrow \text{SVD}$$

$$\text{cov}(X) = S = W \Lambda W^T \longrightarrow \text{PCA}$$

$$\Sigma = \begin{bmatrix} s_1 & & & \\ & s_2 & & \\ & & \ddots & \\ & & & s_d \end{bmatrix}$$

→ diagonal matrix

→ Singular values of X

$$\frac{s_i^2}{n-1} = \lambda_i$$

U = left singular vectors

V = right singular vectors.

U_i = i^{th} column of U = i^{th} eigen vector of XX^T

V_i = i^{th} column of V = i^{th} eigen vector of $X^TX = S*(n-1)$

Non Negative Matrix Factorization (4.5) :-

aka NMF / NNMF

$$A_{n\times m} = B_{n\times d} (C^T)_{d\times m} \quad \text{such that } B_{ij} \geq 0 \text{ and } C_{ij} \geq 0 \;\forall i,j$$

gives all $A_{ij} \geq 0$ which is why it can be used in recommender systems.

Matrix Factorization and Collaborative filtering (4.6) :-

$$A = \begin{array}{c} \\ u_i \end{array} \begin{array}{c} I_j \\ \begin{bmatrix} & A_{ij} & \end{bmatrix} \end{array}$$

A_{ij} = rating of I_j by U_i

A is very sparse matrix.

suppose $A_{n\times m} = B_{n\times d} C^T_{d\times m}$ $\quad d > 0 \quad d \leq \min(n,m)$

$$B = \begin{array}{c} U_1 \\ U_2 \\ \vdots \\ U_n \end{array} \begin{bmatrix} \text{———} \\ \text{———} \\ \\ \\ \end{bmatrix}$$ → each row is a vector representation of each user.

$$C = \begin{array}{c} I_1 \\ I_2 \\ \vdots \\ I_m \end{array} \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$$

$$A_{ij} = \underbrace{B_i}_{i^{th}\text{ row of } B} * \underbrace{C_j}_{j^{th}\text{ row of } C} = B_i^T * C_j$$

→ goal / optimization problem is to decompose the few non sparse elements of A into B and C. (using matrix factorization)

$$\underset{B,C}{\operatorname{argmin}} \sum_{\substack{i,j \\ \text{such that} \\ A_{ij} \neq 0}} \underbrace{(A_{ij} - B_i^T C_j)^2}_{\text{squared loss. Similar to regression's } (y_i - w^T x_i)^2}$$

We can use techniques like SGD since this not an NP Hard problem.

→ steps

1: solve optimization problem.

2: get B and C.

3: <u>Matrix Completion</u>:- Fill the missing values of A using B and C.

$$A_{ij} = B_i^T C_j$$

$$\hat{A} = B * C^T$$

filled matrix. No empty cells.

We have to ensure $A - \hat{A}$ is very small.

Matrix Factorization for feature Engineering (4.7):-

→ can also be used to create features.

$$A = \begin{array}{c|c} & 1\,2\,3 \cdots I_j \cdots m \\ \hline 1 & \\ 2 & \\ \vdots & \\ u_i & \\ \vdots & \\ n & \end{array} \xrightarrow{\text{After MF}} \begin{array}{l} u_i = \\ I_j = \end{array} \begin{bmatrix} \boxed{}\boxed{}\boxed{}\cdots\boxed{} \\ \boxed{}\boxed{}\boxed{}\cdots\boxed{} \end{bmatrix} \Big\} \text{ d dimensions}$$

$$A_{n \times m} = B_{n \times d} \; C^T_{\cancel{d \times d}\; m \times d}$$

i^{th} row of $B = U_i$

j^{th} row of $C = I_j$

→ If U_i and U_j are similar in their ratings of products/items then $dist(U_i, U_j)$ is small.

Clustering as MF (4.8):-

in K Means, $D = \{x_i\} \xrightarrow{task} \begin{cases} S_1, S_2, \ldots, S_K \rightarrow \text{sets} \\ C_1, C_2, \ldots, C_K \rightarrow \text{cluster centroids} \end{cases}$

such that $\min_{C_i} \sum_{i=1}^{K} \sum_{x \in S_i} \|x - C_i\|^2$

Lets define Z such that $Z_{ij} = \begin{cases} 1 & \text{if } x_j \in S_i \\ 0 & \text{otherwise} \end{cases}$

$$Z = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ \vdots \\ i \\ \vdots \\ K \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & \cdots & j & \cdots & n \end{array} \\ \begin{bmatrix} & & & & & & \\ & & & & \square & & \\ & & & & z_{ij} & & \\ & & & & & & \end{bmatrix}_{K \times n} \end{array}$$

1 if $x_j \in S_i$ formed by C_i

if i^{th} 1, all others vals of column = 0

Now rewriting KNN loss function as

$$\min_{C_i, z_{ij}} \sum_{i=1}^{K} \sum_{j=1}^{n} z_{ij} \|x_j - C_i\|^2$$

$$\rightarrow X = \begin{bmatrix} \uparrow & \uparrow & & \uparrow & & \uparrow \\ x_1 & x_2 & \cdots & x_j & \cdots & x_n \\ \downarrow & \downarrow & & \downarrow & & \downarrow \end{bmatrix}_{d \times n}$$

$$Z \longrightarrow z_{ij} = \begin{cases} 1 & \text{if } x_j \in S_i \\ 0 & \text{otherwise} \end{cases}$$

$$C = \underbrace{\begin{bmatrix} \uparrow & \uparrow & & \uparrow & & \uparrow \\ c_1 & c_2 & \cdots & c_i & \cdots & c_k \\ \downarrow & \downarrow & & \downarrow & & \downarrow \end{bmatrix}_{d \times k}}_{\text{centroids of each cluster}}$$

Now the minimization eqn becomes

$$\underbrace{\min_{C, Z} \|X - CZ\|^2}$$

now this is MF form ie, if X can be factorized into $C * Z$, then we minimize $\min_{C, Z} \|X - CZ\|^2$

→ This means K Means is basically Matrix factorization + column constraints

column constraints = elems should be 0/1 and sum of column elems = 1.

Hyperparameter Tuning (4.9) :-

$A = B_{n \times d} \, C^T_{m \times d}$

d is the hyperparameter.

→ it could be problem specific.

→ systematic approach :-

minimization is $\min_{B,C} \underbrace{\sum_{i,j} (A_{ij} - B_i^T C_j)^2}_{\text{loss}}$

$\Rightarrow \text{error} = \sum_{i,j} (A_{ij} - B_i^T C_j)^2$

as $d \uparrow$, error $\downarrow$

→ inflection point. This is a good point.

errors is reducing very fast till this point.

Netflix prize (4.10) :-

→ Adding L2 regularization to loss func in MF to avoid overfitting

$$L = \min_{q,p} \sum_{(u,i)} (r_{ui} - q_i^T p_u) + \lambda (\|q_i\|^2 + \|p_u\|^2)$$

r_{ui} = rating of u on i (A)

q_i = item vector (B)

p_u = user vector (C)

→ to solve L, we could use SGD ie,

$\frac{\partial L}{\partial p_u}$, $\frac{\partial L}{\partial q_i}$ but it takes lot of time.

→ So we can use Alternating Least Squares (ALS) (similar to SGD)

Step ① :- fix p_u and use gradient descent to find q_i's.

Step ② :- fix q_i and use gradient descent to find p_u's.

Step ③ :- repeat ① & ②.

(138) → ALS is faster for this problem.

→ we add three more terms

$\underbrace{b_u, b_i}_{\text{bias terms for each user \& item.}}$, $\underbrace{\mu}_{\text{mean term}}$ → All are scalars.

mean term. Average rating across all users & items. $\left(\frac{1}{n}\sum_{u,i} r_{ui}\right)$

$$\min_{q,p,b} \sum_{u,i} \left(r_{ui} - \mu - b_u - b_i - p_u^T q_i\right)^2 + \lambda\left(\|q_i\|^2 + \|p_u\|^2 + b_u^2 + b_i^2\right)$$

ex:- Suppose μ of netflix = 3.0

user Joe gave titanic a rating of 3.9

Now it can be broken down into

$$\underbrace{3.0}_{\mu} + \underbrace{0.5}_{b_i} + \underbrace{(-0.3)}_{b_u} + \underbrace{0.7}_{p_u^T q_i}$$

b_i (item bias) titanic item tends to have a higher rating

b_u (user bias) Joe tends to be very critical with his ratings

$p_u^T q_i$ (interaction effect) interaction b/w Titanic and Joe. The actual rating given by Joe to titanic.

→ b_u and b_i can further be made time dependent as user ratings tend to change overtime

Cold Start Problem (4.11):-

→ when new user/item is added the row/column is initially empty.

→ when new users :- top items based on metadata like location, device, age etc., essentially build content based recommender system.

→ when new item :- metadata : category, description.
therefore similar items and add these as new arrival

→ when new items are added, we don't have to run MF from scratch. Instead of initializing with random values, in SGD, initialize available rows & columns with prev iteration's values.

Word Vectors as MF (4.12) :-

→ lots of word vectors existed before word2vec, like LSA, PLSA, LDA, GLOVE

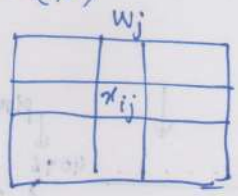
GLOVE: related to w2v

All can be represented as Matrix Factorization

word2vec as SVD :-

Step ① :- Build a cooccurance matrix(X)

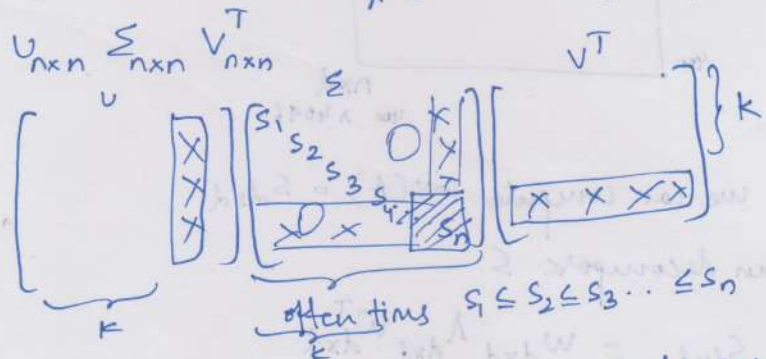
$D = \{\text{text docs}\} \longrightarrow X_{n \times n} =$



x_{ij} = # of times w_j occurs in context of w_i

i.e., # of times w_j is in neighbourhood of w_i (neighbourhood size 2-10 usually)

Step ② :- $X_{n \times n} = U_{n \times n} \Sigma_{n \times n} V^T_{n \times n}$



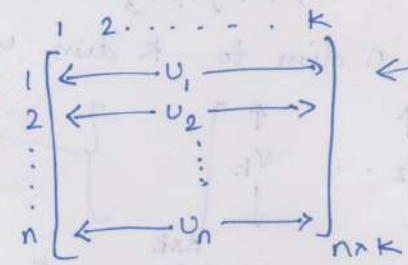
often times $s_1 \le s_2 \le s_3 \ldots \le s_n$

truncated SVD :- instead of taking n cols, we only take k cols & discard the rest

$$\Rightarrow U_{n \times k} \Sigma_{k \times k} V^T_{k \times n} = \hat{U} \hat{\Sigma} \hat{V}^T = \hat{X} \approx X$$

like PCA, we take top k values. k is the hyperparam

U_i = word-vec corresponding to word i



Step ③ :- $\hat{U}$: rows of $\hat{U}$ as $u_1, u_2, \ldots, u_n$ $\quad u_i \in \mathbb{R}^k$

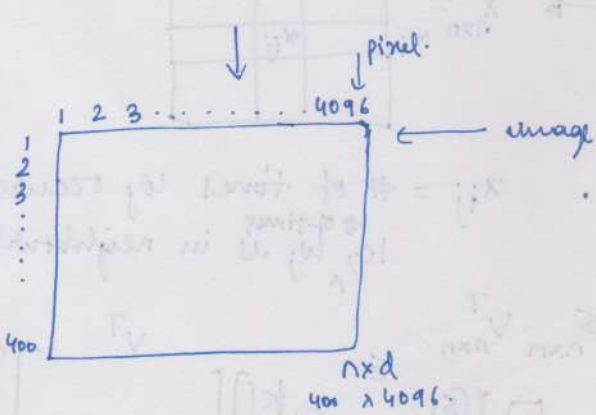
word 1 word 2 word n

(190) Problem is coocurance matrix is very large if # of words = large.
Solution is : instead of using all n words, use most imp words (chosen using tfidf)

Eigen Faces (4.13) :- (PCA on images)

→ early technique for ~~facial~~ face recognition. Nowadays we use CNN.

→ following example:- (from lecture video)



Given A, we can compute $cov(A) = S_{d\times d}$

we can decompose S

$$S_{d\times d} = W_{d\times d} \Lambda_{d\times d} W^T_{d\times d}$$

eigen vectors

$$\Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_d \end{bmatrix}$$

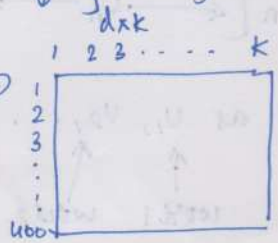
$$\lambda_1 \geq \lambda_2 \geq \lambda_3 \ldots$$

we know to convert n dim to k dim, we select top k eigen vectors

$$W_k = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ v_1 & v_2 & \cdots & v_k \\ \downarrow & \downarrow & & \downarrow \end{bmatrix}_{d\times k}$$

each v_i is a 4096 dim vector from which we can create image. These images are called Eigen faces.
In these images, the white part is what it's trying to capture information in.

$$A_{n\times d} W_{k\,d\times k} = \tilde{A}_{n\times k}$$



Code Example (4.14):-

$$\|X\|^2 = \sum_i \sum_j X_{ij}^2 \longrightarrow \text{Frobenius Norm of } X.$$

→

SVD	PCA
randomized algo	not randomized algo
operates on sparse matrix	does not
faster	slower
more commonly used	less common
better dim reduction	
cov doesn't have to be ~~rectangular~~ ~~symmetric~~ square	cov matrix needs to be square.

sklearn implementation under the hood uses SVD to obtain low dimensional mapping.

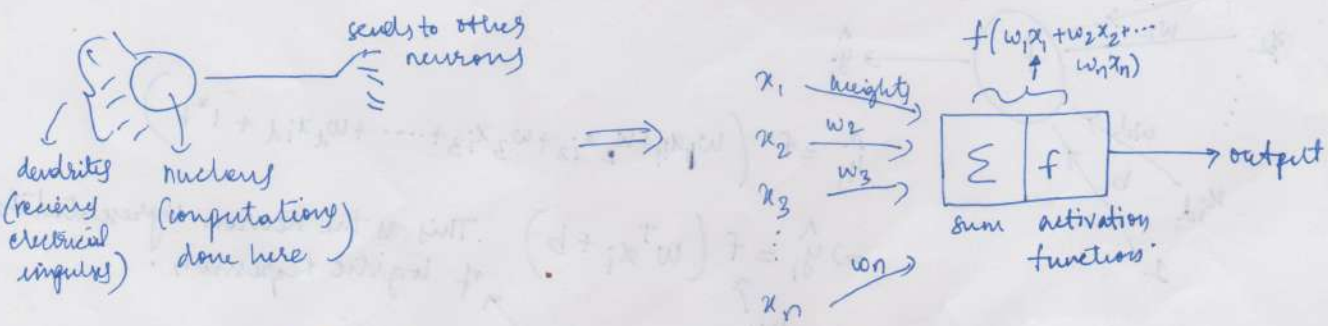
To check performance

: Before deployment : RMSE

: After deployment : Average Precision at K and mean Average Recall at K.

History of Neural Networks & Deep Learning (1.1):-

→ Perceptron : Simplest NN. From 1957. Similar to logistic Regression.



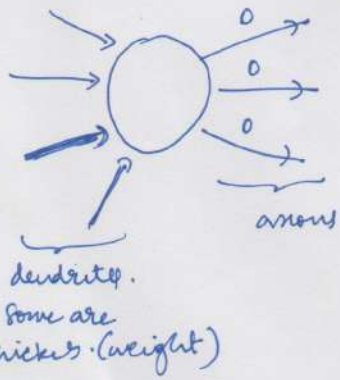
→ loosely inspired by biology.

→ to try & replicate collection of neurons in brain, first successfull attempt by Hinton in 1986 called back propogation (chain rule basically)

→ input layer → $L_1 \to L_2 \to L_3$ → output layer

depth.

How Biological Neurons work (1.2):-



If there's enough input the neuron/cell is activated/fired.

∴ f is called activation function.

$$\text{output, } o = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

activation function; weights; inputs

→ biologically, learning = increasing weights of connections b/w neurons.

Diagrammatic Representation: Logistic Regression v/s Perceptron (1.4):-

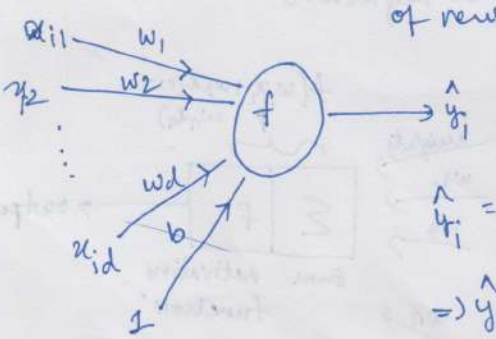
→ In logistic regression, given x_i we predict $\hat{y}_i$

$\hat{y}_i = \text{Sigmoid}(w^T x_i + b)$ we get w & b after training LR.

if $x_i \in \mathbb{R}^d$, then w also $\in \mathbb{R}^d$ & $b \in \mathbb{R}$

$$\hat{y}_i = \text{Sigmoid}\left(\underbrace{\sum_{j=1}^{d} w_j x_{ij}}_{} + b\right) \qquad x_i = [x_{i1}, x_{i2}, \dots x_{id}] \quad w = [w_1, w_2, \dots w_d]$$

this looks similar to functional part of neuron i.e., $f\left(\sum_{i=1}^{n} w_i x_i\right)$



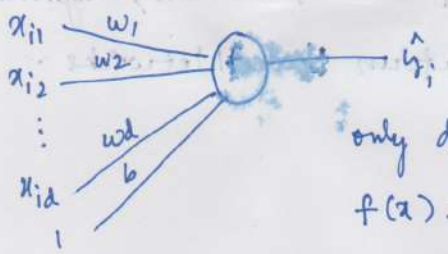
$$\hat{y}_i = f\left(w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} + \dots + w_d x_{id} + 1 * b\right)$$

$$\Rightarrow \hat{y}_i = f\left(w^T x_i + b\right)$$

this could be Sigmoid then

This is the neuron representation of logistic regression.

Perceptron :-



only diff is activating function

$f(x) = 1$ if $W^T x + b > 0$

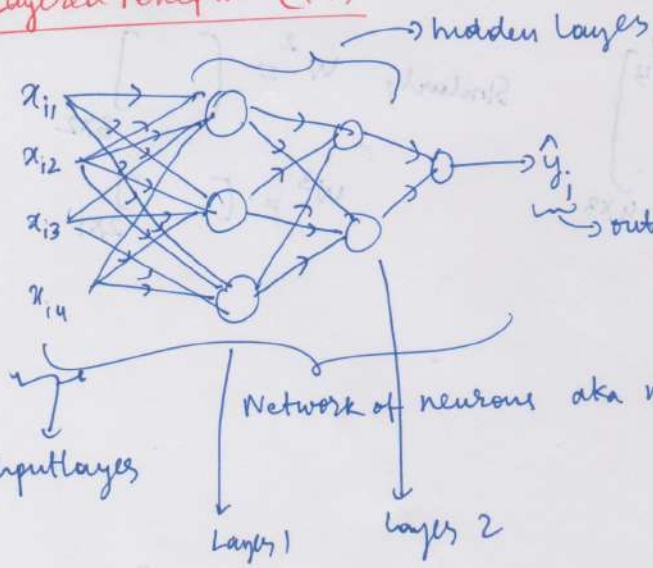
0 otherwise

→ perceptrons can also be seen as a linear classifier.

No squashing fun like in LR.

→ Training done just like in ~~SGD~~ LR using SGD and W are found.

Multi Layered Perceptron (1.5) :-



Why :-

→ Biological Inspiration.

→ By using multi layered structure we can come up with complex mathematical functions. This gives enormous power to models & helps us solve non linear problems.

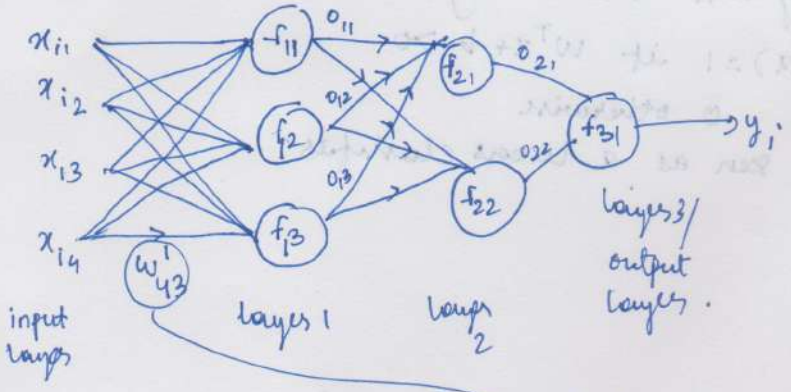
→ MLP is graphical way of functional composition ($f(g(x))$, etc.)

→ They also overfit very easily

→ In MLP, we generally use three functions ; Multiplication, Addition, Sigmoid.

→ On dense tabular data, RF/GBMachines will outperform Neural Networks.

Notation (1.6) :-



$D = \{x_i, y_i\}$ $x_i \in \mathbb{R}^4$ $y_i \in \mathbb{R}$

w^1_{43} → to layer 1, from i/p, to i/p

fully connected NN/MLP = we make all possible connections from one layer neurons to another layer neurons.

all weights from i/p layer to layer 1 can be represented as

$$W^1 = \begin{bmatrix} w^1_{11} & w^1_{12} & w^1_{13} \\ & \vdots & \\ & & \end{bmatrix}_{4\times 3}$$

Similarly $W^2 = [\quad]_{3\times 2}$

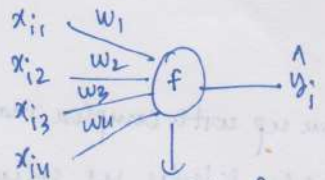
$W^3 = [\quad]_{2\times 1}$

Training a Single Neuron Model (1.7) :-

train = find best weights using D_{train}

→ In ~~logistic~~ linear Regression.

$$\hat{y}_i = \sum_{j=1}^{d} w_j x_{ij} \qquad \hat{y}_i = W^T x_i$$



input to f $= \sum_{j=1}^{4} w_j x_{ij}$ } $\Rightarrow f(z) = z$. This is an identity function

output of f = "

optimization func of LR =

$$\min_{w_i} \sum_{i=1}^{n} (y_i - w^T x_i)^2 + \|w\|^2$$

step ①:- defining loss function

let L = loss across all data & L_i = loss of one point. i.e. $L = \sum_{i=1}^{n} L_i$

$$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + reg.$$

$$L_i = (y_i - \hat{y}_i)^2 \qquad \hat{y}_i = w^T x_i$$ if $y_i - \hat{y}_i$ is more we penalize more.

step ②:-
optimization function:-

$$\min_{w_i} \sum_{i=1}^{n} (y_i - f(w^T x_i))^2 + reg$$

if f = Identity → linear
Sigmoid → logistic
thresholding function → Perceptron.

$$w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - f(w^T x_i))^2 + reg.$$

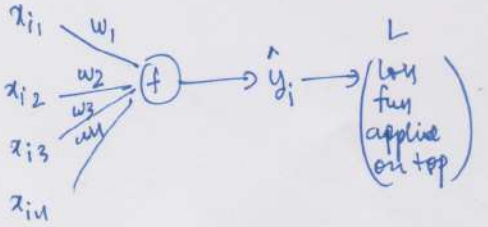
step ③:- solve the optimization problem.

ⓐ initialization of w_i's randomly.

ⓑ $$\nabla_w L = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \vdots \\ \frac{\partial L}{\partial w_d} \end{bmatrix} \qquad x_i \in \mathbb{R}^d, \; w \in \mathbb{R}^d$$

ⓒ $$w_{new} = w_{old} - \underbrace{\eta}_{\text{learning rate}} [\nabla_w L]_{w_{old}}$$

$$(w_i)_{new} = (w_i)_{old} - \eta \left[\frac{\partial L}{\partial w_1}\right]_{at (w_1)_{old}}.$$ in for $i = 1$ to k epochs.



we want to compute $\nabla_w L$

$$= \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots \frac{\partial L}{\partial w_d}\right]^T$$

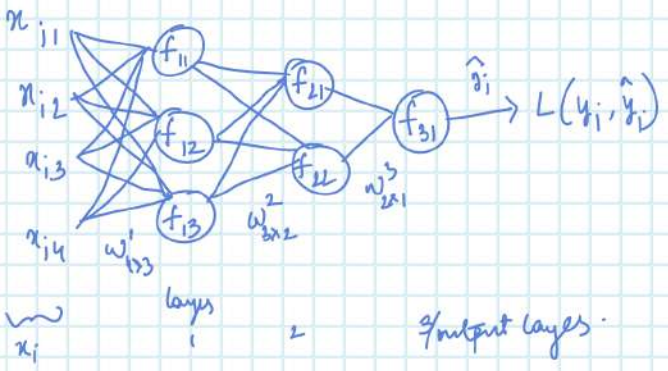
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial w_1}$$

$$\frac{\partial L}{\partial f} = -\sum_{i=1}^{n} 2(y_i - f(w^T x_i)) \qquad \frac{\partial f}{\partial w_1} = x_i$$

$$\hookrightarrow \frac{\partial L}{\partial w_1} = \cancel{-2x_i(y_i - \hat{y}_i)} \quad \sum_{i=1}^{n} -2x_i(y_i - \hat{y}_i)$$

1.8 Training an MLP (Chain Rule) :-

$D = \{x_i, y_i\}$ $x_i \in \mathbb{R}^4$, $y_i \in \mathbb{R}$ (example), loss is squared loss in regression.



→ By the end we need to determine $w^1_{4\times3}$, $w^2_{3\times2}$, $w^3_{2\times1}$ = 20 weights (this is the training in MLP)

<u>step ①</u> :-

$$L = \underbrace{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}_{\text{squared loss}} + \text{reg}$$

→ L_2 reg equivalent would be $\sum_{i,j,k} \left(\| w^k_{ij} \| \right)^2$

L_1 reg equivalent would be $\sum_{i,j,k} \left(| w^k_{ij} | \right)$

$$L_i = (y_i - \hat{y}_i)^2$$

$$L = \sum_{i=1}^{n} L_i + \text{reg}$$

<u>step ②</u> :- optimization problem :-

$$\min_{w^1, w^2, w^3} L$$ → squared loss & regularization.

$$= \min_{w^k_{ij}} L$$

<u>step ③</u> :- SGD (or) GD

ⓐ initialization of w^k_{ij} randomly (there are lot of techniques to initialize them)

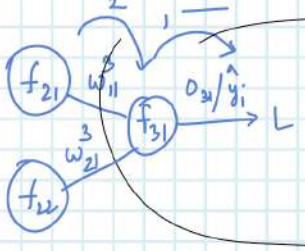
ⓑ $$(w^k_{ij})_{new} = (w^k_{ij})_{old} - \eta \frac{\partial L}{\partial w^k_{ij}}$$

η → learning rate

$\frac{\partial L}{\partial w^k_{ij}}$ → most important part

ⓒ perform updates till convergence.

ex :- we are trying to compute $\frac{\partial L}{\partial w^3_{11}}$. ($w^3_{11}$ directly impacts $O_{31} / \hat{y}_i$ which impacts L)

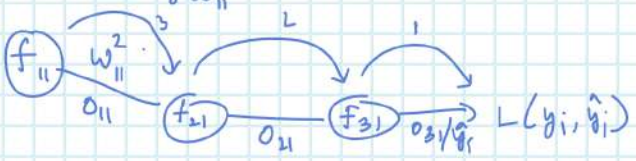


$$\frac{\partial L}{\partial w^3_{11}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w^3_{11}}$$

$$\frac{\partial L}{\partial w^3_{21}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w^3_{21}}$$

chain rule. w_3 is solved now.

→ Now we calculate $\frac{\partial L}{\partial w_{11}^2}$. w_{11}^2 impacts O_{11}, O_{21} & $O_{31}/\hat{y}_i$

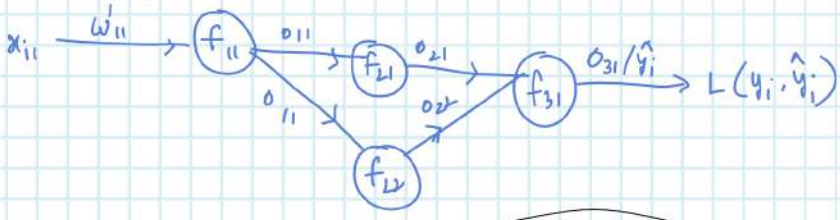


$$\Rightarrow \frac{\partial L}{\partial w_{11}^2} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial w_{11}^2}$$

Similarly

$$\frac{\partial L}{\partial w_{21}^2} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial w_{21}^2}$$

→ Calculating $\frac{\partial L}{\partial w_{11}^1}$



In calculus,

$x \to f(x), g(x) \to h(f(x), g(x))$, then $\frac{\partial h}{\partial x} = \frac{\partial h}{\partial f} \cdot \frac{\partial f}{\partial x} + \frac{\partial h}{\partial g} \cdot \frac{\partial g}{\partial x}$

We have

$x \to f(x), g(x) \to h(f(x), g(x)) \to K(x)$.

$$\frac{\partial K}{\partial x} = \frac{\partial K}{\partial h} \left(\frac{\partial h}{\partial x}\right)$$

$$\Rightarrow \frac{\partial K}{\partial x} = \frac{\partial h}{\partial K}\left[\frac{\partial h}{\partial f} \cdot \frac{\partial f}{\partial x} + \frac{\partial h}{\partial g} \cdot \frac{\partial g}{\partial x}\right]$$

$$\rightarrow \frac{\partial L}{\partial w_{11}^1} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w_{11}^1}$$

$$= \frac{\partial L}{\partial O_{31}} * \left[\frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial O_{11}} \cdot \frac{\partial O_{11}}{\partial w_{11}^1} + \frac{\partial O_{31}}{\partial O_{22}} \cdot \frac{\partial O_{22}}{\partial O_{11}} \cdot \frac{\partial O_{11}}{\partial w_{11}^1}\right]$$

Training an MLP: Memoization (1.1):-

→ As seen in Dynamic Programming

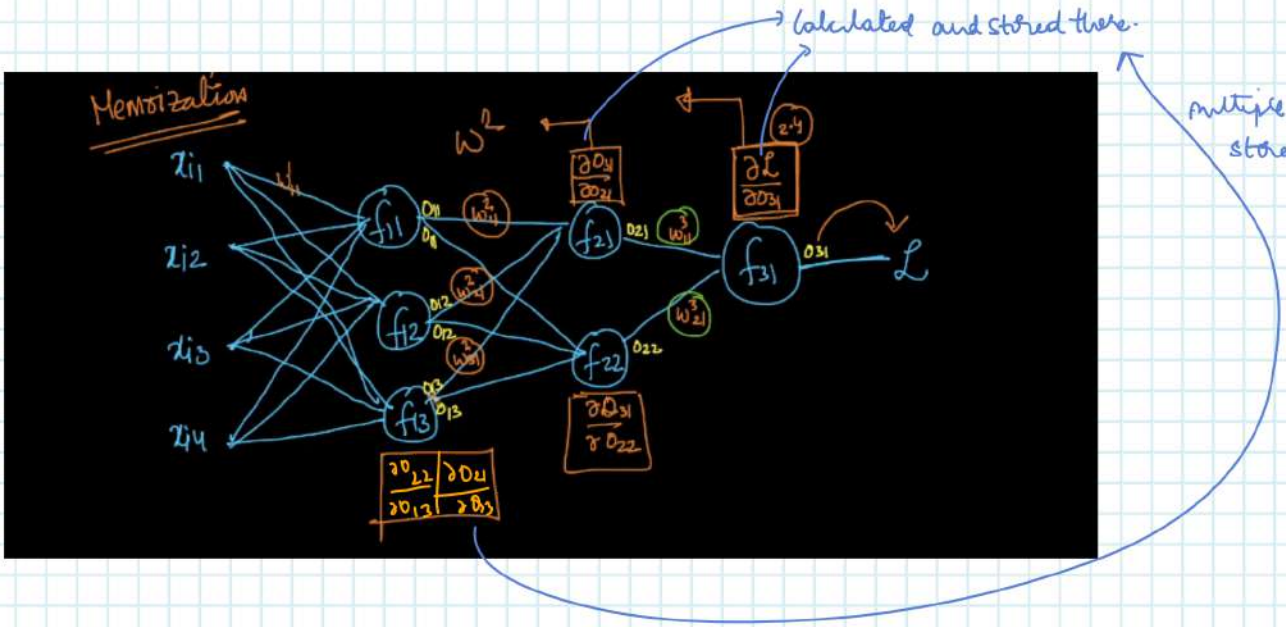
→ $\frac{\partial L}{\partial w_{11}^3} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w_{11}^3}$

$\frac{\partial L}{\partial w_{21}^3} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w_{21}^3}$

→ same. Core idea is to compute and reuse.

→ if there is any operation performed repeatedly, compute it once & reuse it.

→ takes slightly more memory though.



→ Chain Rule + Memoization = Backpropagation; with some optimization.

Backpropagation (1.10):-

Given $D = \{x_i, y_i\}$

Step ①:- Initialize w^k_{ij}

Step ②:- for each x_i in D, pass x_i forward through the network. The end result would be $\hat{y}_i \longrightarrow L(y_i, \hat{y}_i)$

This step is called forward propagation.

Then compute the $L(\hat{y}_i, y_i)$

Compute all derivatives using chain rule and memoization.

Update weights from end of the network to the start $\longrightarrow$ $(w^3_{11})_{new} = (w^3_{11})_{old} - \eta\left(\frac{dL}{dw^3_{11}}\right)$

Backward propagation. We are using the loss to update the weights.

$$\left\{\begin{array}{l} \vdots \\ (w^2_{11})_{new} = \cdots \\ \vdots \\ (w^1_{11})_{new} = \cdots \end{array}\right.$$

Step ③:- Repeat step ② till convergence. ie, $(w^k_{ij})_{new} \approx (w^k_{ij})_{old}$

epoch = Passed all the datapoints to the neural network once. We do multiple epochs in Real World.

Backpropagation = Multi epoch training methodology that uses chain rule & memoization to update weights.

→ Backpropagation works iff activation functions are differentiable. If it is easily differentiable, then training of NN can be sped up.

→ Instead of sending single point in backprop we can send a subset $S \subseteq D$, (Similar to mini batch SGD)

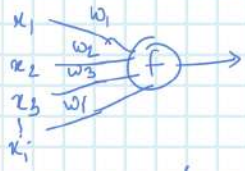
$|S|$ = mini batch size. (typically powers of 2 like 64, 128, ...)

Keeping all datapoints in RAM & computing gradients is hard.

∴ Mini batch SGD is most popular.

Activation Functions (1.11) :-

→ In 1980 & 1990's, sigmoid and tanh were the most popular.



let $z = \sum_{i=1}^{n} w_i x_i = w^T x$

if f is sigmoid, then $f(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$ (This is also used in LR squashing)

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

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

Derivative of $\sigma(z)$ is expressed in terms of $\sigma(z)$.
⇒ Can be helpful in backpropagation.

→ $a = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

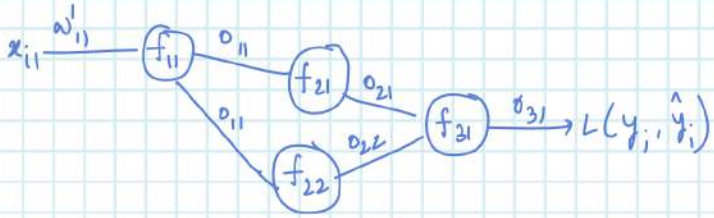
$$\frac{d\tanh}{dz} = 1 - \tanh^2(z)$$

→ ReLU are more popular activation functions these days.

Vanishing Gradient Problem :-

→ Most imp reason NN lost hype back in 90's

→



$$\frac{\partial L}{\partial w^1_{11}} = \frac{\partial L}{\partial o_{31}} \left[\frac{\partial o_{31}}{\partial o_{21}} \cdot \frac{\partial o_{21}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w^1_{11}} + \frac{\partial o_{31}}{\partial o_{22}} \cdot \frac{\partial o_{22}}{\partial o_{11}} \cdot \frac{\partial o_{11}}{\partial w^1_{11}} \right]$$

$0 \le \quad < 1$

$$\frac{\partial o_{31}}{\partial o_{21}} = \frac{\partial f_{31}}{\partial o_{21}} \quad \text{since } o_{31} = f(\text{something})$$

All f's are sigmoids and lie b/w 0 & 1 and the derivatives are being multiplied. Results in much smaller value.

As a result, in

$$(w^1_{11})_{new} = (w^1_{11})_{old} - \eta \left(\frac{\partial L}{\partial w^1_{11}} \right)$$

$(w^1_{11})_{new}$ & $(w^1_{11})_{old}$ will be very close.

This problem is called vanishing gradient.

$\frac{\partial L}{\partial w^k_{ij}}$ is small → vanishing gradient

is very large → exploding gradient (happens when activation function's result > 1)

Exploding Gradient using Sigmoid function:-

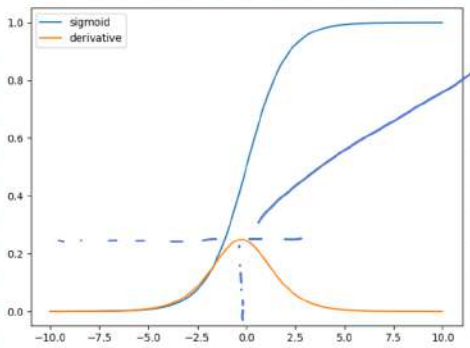
$$\text{input } (x) \xrightarrow[w_1, b_1]{} \overset{z_1}{\sigma} \xrightarrow[w_2, b_2]{o_1} \overset{z_2}{\sigma} \xrightarrow[w_3, b_3]{o_2} \overset{z_3}{\sigma} \xrightarrow[w_4, b_4]{o_3} \overset{z_4}{\sigma} \xrightarrow{o_4} L$$

$o_1 = \sigma(z_1)$ $\quad o_2 = \sigma(z_2)$ $\quad o_3 = \sigma(z_3)$

$z_1 = w_1 x + b_1$ $\quad z_2 = w_2 o_1 + b_2$ $\quad z_3 = w_3 o_2 + b_2$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{o_4} \cdot \underbrace{\frac{\partial o_4}{\partial o_3} \cdot \frac{\partial o_3}{\partial o_2} \cdot \frac{\partial o_2}{\partial o_1}} \cdot \frac{\partial o_1}{\partial w_1}$$

if activation func such that each > 1, it leads to exploding gradient.
But sigmoid max is (0.25)



take $\frac{\partial o_2}{\partial o_1} = \frac{\partial o_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial o_1}$

$$= \frac{\partial \sigma(z)}{\partial z_2} \cdot \frac{\partial (w_2 o_1 + b_2)}{\partial o_1}$$

$$= \sigma'(z_2) * \underbrace{w_2}_{} \quad \text{where } \sigma'(z) = \frac{\partial \sigma(z)}{\partial z}$$

weights are initialized randomly.
If it is a large enough value, it leads to exploding gradient.

Bias Variance Tradeoff (1.13):-

→ # layers ↑ → more weights → higher chance of overfit → high variance.

1 layer → aka log reg → higher chance of underfit → high bias.

→ MLP generally tends to have high variance since no one chooses one layer.

→ to reduce variance we use regularization term along with loss in optimization

$$L = \sum_{i=1}^{n} \text{loss}(y, \hat{y}_i) + \sum_{i,j,k} (w_{ij}^k)^2$$

$$= \sum_{i=1}^{n} \text{loss} + \lambda L_2$$

larger λ ⇒ lesser overfit

using L_1 reg leads to sparsity (connections b/w neurons of various layers dont exist)

Deep MultiLayered Perceptrons : 1980's to 2010s (2.1) :-

→ Upto 2010 people were using NN with 2-3 layers. Because of Vanishing Gradient & lack of data. Also no computing power.

→ From 2010, we started having lots of labelled data. We also started having GPUs. New ideas & algos as well.

→ Approach was also changed. Instead of going theory first, experiments were performed more which took lot less time than coming up with new theory.

Dropout Layers & Regularization (2.2):-

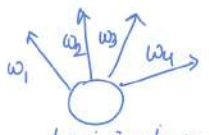
→ One of the major issues of Deep NN is overfitting.
We use regularization to overcome this.

→ In random forest, we get many trees. But we only select subset of trees & subset of features.
We use randomization of features to create regularization.
Core idea of dropout layer is to use this philosophy for MLP's

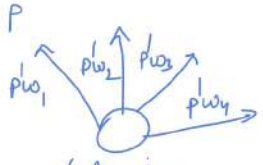
→ In each iteration randomly remove some neurons. As a result, some incoming & outgoing connections will be lost.
dropout rate $0 \le p \le 1$. At any iteration, in each layer $p*100$ % of neurons shall be inactive.

→ Dropout is very similar to having random subset of features in Random Forest.

→ During testing we multiply w with p



training time, neurons are present with probability p



testing time, multiply weights with p.

→ When we are overfitting with many weights & few data points, dropout rate should be large.

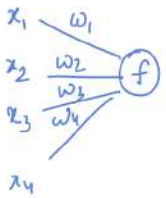
→ Dropouts can also be thought of as a layer that takes inputs from one layer but does not send all of them to the next layer.

Rectified Linear Units (ReLU) (2.3):-

→ One of the main problems with classical NN is vanishing gradient problem.

→ ReLU is the most common activation function for MLP these days.

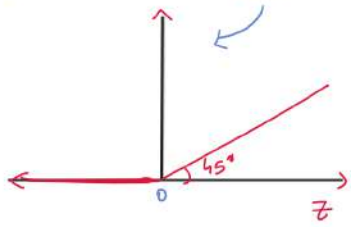
→



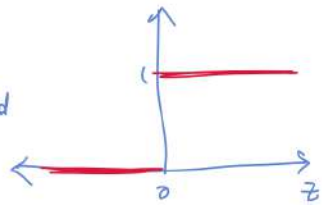
input of $f = W^T X$

$f(z) = (z^+)$ → denotes of ReLU.

$$f(z) = \max(0, z) = \begin{cases} 0 & \text{if } z \le 0 \\ z & \text{otherwise} \end{cases}$$



→ when differentiated



↳ Not differentiable at 0
Since it's not a smooth function.

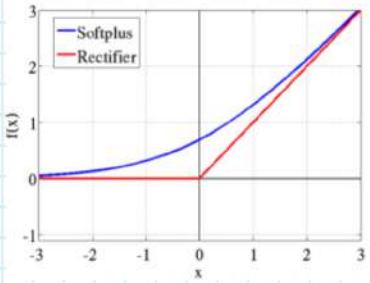
→ $\frac{\partial f_{ReLU}}{\partial z} \in \{0, 1\}$ ⟹ As a result no vanishing gradient, no exploding gradient.

→ But this 0 could cause dead activations

→ ReLU converges faster than tanh/sigmoid because there's no vanishing gradient/exploding gradient.

→ ReLU is sometimes also called simply as rectifier.

→ softplus function is like an approximation of ReLU



$$f(x) = \log(1+\exp(x))$$

$$\frac{\partial f}{\partial x} = \frac{\exp(x)}{(1+\exp(x))} \longrightarrow \text{logistic function.}$$

→ Variations of ReLU :-

Noisy ReLUs :- $f(x) = \max(0, x+Y)$, $Y \sim N(0, \sigma(x))$

Leaky ReLUs :- $z = W^T x$, if weights are −ve,
then z = −ve
if z = −ve, $\partial f(z) = 0$
⇒ the product of this derivative with others in chain rule = 0

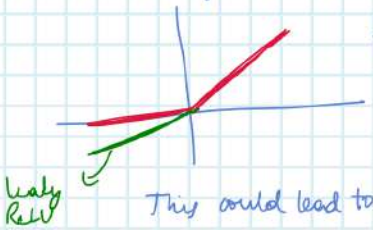
$$\Rightarrow W_{new} = W_{old} - \eta \underbrace{\left(\frac{\partial f}{\partial W}\right)}_{0}$$

⇒ $W_{new} = W_{old}$. weights are not updated.

This state is called dead activation state.

We must always keep track of this when using ReLU

Leaky ReLU fixes this



$$f(x) = \begin{cases} x & \text{if } x \le 0 \\ 0.01x & \text{otherwise} \end{cases} \qquad \frac{\partial f}{\partial z} = \begin{cases} 1 & ; z \text{ +ve} \\ a & ; z \text{ −ve} \end{cases}$$

→ it's the hyperparameter typical $0 \le a \le 1$

This could lead to vanishing gradient sometimes.

Weight Initialization (2.4):-

→ In SGD, we initialized randomly (uniform (or) Gaussian) ie, $W_{ij}^k \sim N(0, \sigma)$

→ Problem of Symmetry :- when all weights are initialized to 0/1/... this is bad.

if f is also same, then

(i) All neurons compute the same thing.

(ii) Same gradient updates

$z = W^T x$ ($W^T \to 0$) ⇒ $z = 0$
if ReLU, whole thing becomes zero.

This is called problem of symmetry.

We want stuff to be asymmetric because we want each neuron of layer to learn something different

→ if we initialize to large −ve values, then $f(z) = 0$ if ReLU is used. if sigmoid or tanh is used, leads to vanishing gradient problem.

idea ① :- → weights should be small
→ not all zero
→ good variance

We can do $W_{ij}^k \sim N(0, \sigma)$ σ is small.

This is called Gaussian/Normal initialization

As a result we get good σ and there will be both +ve & −ve weights.

for f, fan_in = 4
fan_out = 2

→ After lots of experiments, people concluded, weights should be based on fan_in, fan_out.

idea ②:- Uniform Initialization :-

$$W_{ij}^{K} \sim \text{Uniform dist}\left[\frac{1}{\sqrt{fan_in}}, \frac{1}{\sqrt{fan_out}}\right]$$ for any neuron

This works well for sigmoid functions.

idea ③:- Xavier/Glorot :-

two variations.

ⓐ Xavier/Glorot Normal :-

$$w_{ij}^{K} \sim N(0, \sigma_{ij}) \qquad \sigma_{ij} = \sqrt{\frac{2}{fan_in + fan_out}}$$

ⓑ Xavier/Glorot Uniform :-

$$w_{ij}^{K} \sim U\left[\frac{-\sqrt{6}}{\sqrt{fan_in + fan_out}}, \frac{+\sqrt{6}}{\sqrt{fan_in + fan_out}}\right]$$

} we use fanin & fanout in this. In previous one we only used fan_in.

works well for sigmoid too.

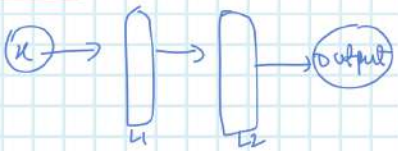
idea ④:- He-Init :-

ⓐ $w_{ij}^{K} \sim N(0, \sigma) \quad \sigma = \sqrt{\frac{2}{fan_in}}$

ⓑ $w_{ij}^{K} \sim U\left[-\sqrt{\frac{6}{fan_in}}, +\sqrt{\frac{6}{fan_in}}\right]$

w well for ReLU.

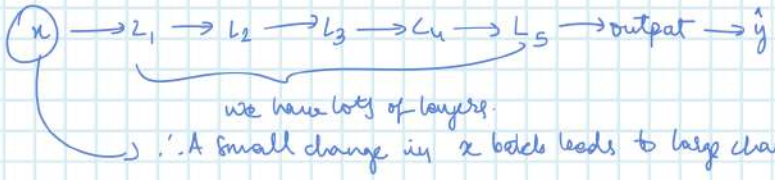
Batch Normalization (2-5) :



$$\tilde{x}_i = \left[\frac{x - \mu}{\sigma}\right]$$

σ = std-dev
μ = mean

→ In minibatch SGD we only send batch of points



we have lots of layers.

→ ∴ A small change in x batch leads to large change in w.

batch 1
batch 2
⋮

→ L_1 wont see much difference.
But L_6 will because of chain rule before it.
This is called internal covariance shift.

→ In Batch Normalization, we add a new layer that normalizes the data of that batch only.

→ γ & β are to be learned in backpropagation

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \qquad \boxed{\tilde{x}_i = \gamma \hat{x}_i + \beta}$$

ϵ → small value.

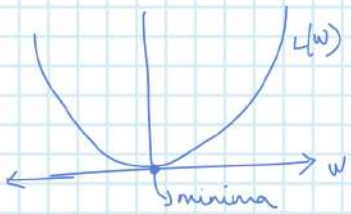
→ Batch Normalization is differentiable.

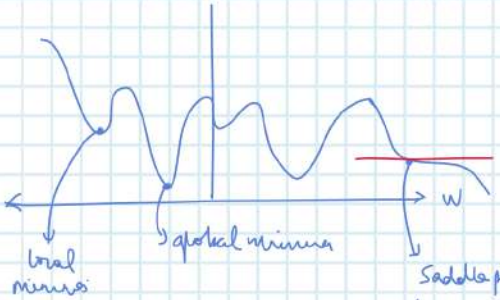
→ During testing, we take the mean & variance of the whole dataset. □

→ Generally, if we use higher learning rate with no normalization, it tends to overshoot minima & convergence is much slower.

Optimizers: Hill Descent analogy in 2D :- (2.6)

→ Simple SGD in deep learning cannot solve the optimization function properly.





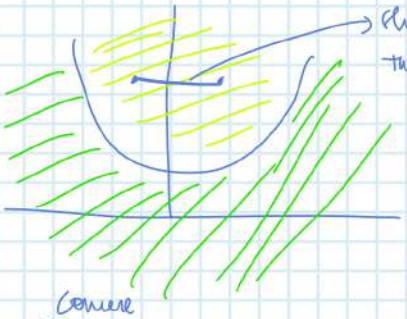
Saddle point
tangent at saddle point ||el to x axis

at minima & maxima and saddle point, $\frac{dL}{dw} = 0$

$$w_{new} = w_{old} - \eta \frac{dL}{dw}$$

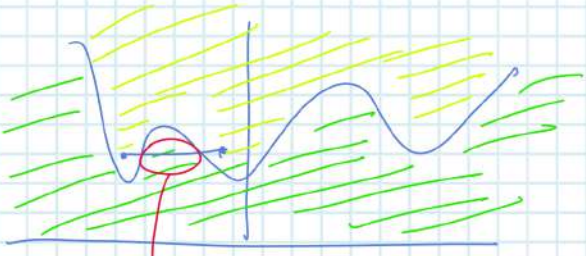
Simple SGD could get stuck at minima maxima or saddle point.

→ Convex functions :-



→ Shortest dist b/w any two points of the region. If all points on this line lie in the same region it's a convex function

Convex



don't lie in the same region ∴ Not convex

→ Convex funcs only have one minima & 1 maxima.
local minima = global minima.
non convex have multiple local minima & 1 global minima.

→ In log reg, lin reg & SVM: all are convex funcs.

→ In deep learning: non convex functions. ie, we have local minima, saddle points etc.,
Depending on w initialization we could end up with local minima.

rajat_chaudhary 47 Votes

Hi,the way I see this, it is upto us to choose which ever loss function we can choose for our algorithm. So is that one of the reason we use squared loss in linear regression, as the squared loss is a convex function, hence no issue of getting stuck at local minima or local maxima or a saddle point. just wanted to confirm my understanding here.

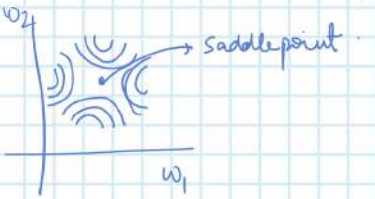
Reply Mar 03, 2019 08:20 AM

Applied_AI

Good question especially for the interviews. You are asked why to use the squared loss and power 4 or modulus. One of the major reason is squared loss upon derivative produces a single value of parameter set and hence gives one unique solution while others don't. However it is not true for one value of the parameter, the loss would be minimal. Loss can be minimal for multiple values also. Hence you can't tell that squared loss would only give you the least solution. Just because you have multiple minima doesn't mean that they need to be different in values. Honestly in Square and power 4 both generates minimal solution but the squared loss gives one solution while power 4 gives 3 solutions.

Reply Mar 03, 2019 10:36 AM

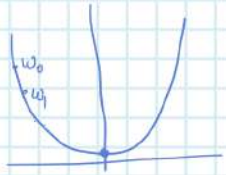
contour plot of a saddle point :-



Simple mini batch SGD will get stuck at saddle point.

SGD Recap (2.8) :-

$$(w_{ij}^k)_{new} = (w_{ij}^k)_{old} - \eta \left[\frac{\partial L}{\partial w_{ij}^k}\right]_{(w_{ij}^k)_{old}}$$

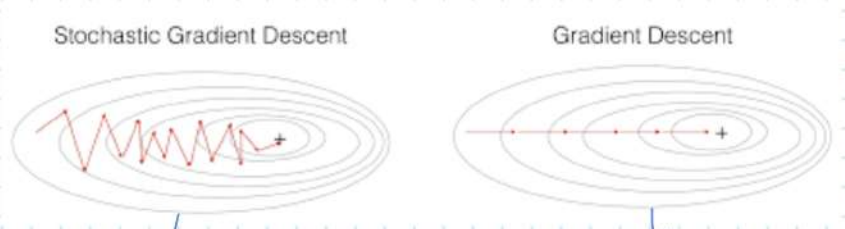


$$\Downarrow$$

$$w_t = w_{t-1} - \eta \left[\frac{\partial L}{\partial w}\right]_{w_{t-1}}$$

→ $\frac{\partial L}{\partial w}$ when calculated using all points → Gradient Descent (computationally expensive)
using one random point in each iteration → SGD
using a random subset of points each itr → MiniBatch SGD (most common in DL)

MiniBatch SGD is erroneous, noisy & an approximation of GD
ie) SGD with lots of iterations ≈ GD
We are estimating the gradient in SGD. So the updates in each iteration are more noisy in SGD than in GD.



Noisy SGD path.
More epochs but faster.

GD path.
Less epochs More time.

Batch SGD with Momentum (2.9) :-

→ Assume we are calculating values

at $t=$	1	2	3
	a_1	a_2	a_3
	v_1	v_2	v_3

Let γ be a value b/w 0 & 1

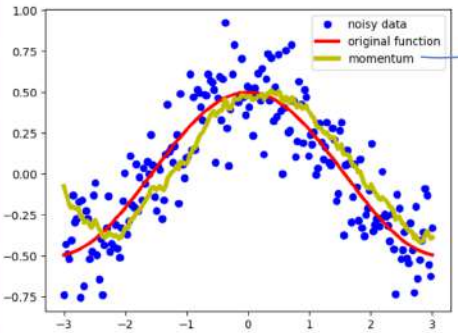
$v_1 = a_1$
$v_2 = \gamma v_1 + a_2$
$v_3 = \gamma v_2 + a_3 = \gamma(\gamma v_1 + a_2) + a_3$

if $\gamma = 1$	$\gamma = 0$	$\gamma = 0.5$
$v_1 = a_1$	$v_1 = a_1$	$v_1 = a_1$
$v_2 = a_1 + a_2$	$v_2 = a_2$	$v_2 = 0.5a_1 + a_2$
$v_3 = a_1 + a_2 + a_3$	$v_3 = a_3$	$v_3 = \underbrace{0.25a_1 + 0.5a_2 + a_3}$

most weightage to a_3
less to a_2
lesser to a_1 ...

$\Rightarrow v_t = \gamma v_{t-1} + a_t$

v_t at any time is called a denoised value. These are called exponentially weighted averages.

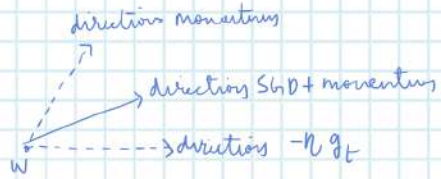


→ obtained after exponentially weighted average. Close approximation to the original function

→ taking this concept & applying it to mini batch SGD :-

$$w_t = w_{t-1} - \eta \left(\frac{\partial L}{\partial w}\right)_{w_{t-1}}$$

Let $g_t = \left(\frac{\partial L}{\partial w}\right)_{w_{t-1}} \Rightarrow w_t = w_{t-1} - \eta g_t$



Applying exponential weighing.

$$v_t = \gamma v_{t-1} - \eta g_t$$ → momentum

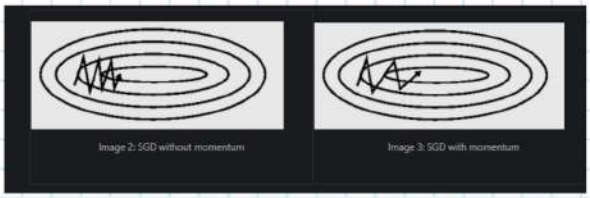
$$w_t = w_{t-1} - v_t$$

In practice typical value of $\gamma = 0.9$

denoising with exponential weighing = SGD + momentum

With momentum, SGD starts taking larger steps towards minima.

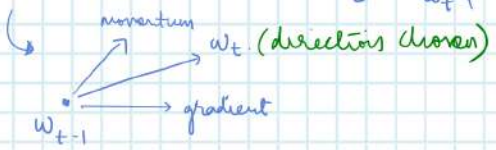
∴ SGD + momentum speeds up convergence.



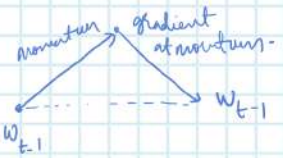
→ As we are moving closer to minima SGD points in the best direction Therefore we use exponential average & not simple average.

Nesterow Accelerated Gradient (2.10) :-

in SGD + momentum $g_t = \left[\frac{\partial L}{\partial w}\right]_{w_{t-1}}$



But in NAG :-



$$w_t = w_{t-1} - (\gamma v_{t-1} + \eta g')$$

$$g' = \left(\frac{\partial L}{\partial w}\right)_{w_{t-1} - \gamma v_{t-1}}$$ = gradient calculated at momentum.

→ if we ever skip minima in an iteration, NAG will take lesser steps till convergence than SGD with momentum.

Adagrad (2.11) :-

→ In SGD & SGD + Momentum, learning rate (η) is constant.

features → sparse, → dense } same learning rate for both is bad.

→ Adagrad = Adaptive Learning rate.

SGD :- $w_t = w_{t-1} - \eta g_t$ → same for all weights.

Adagrad :- $w_t = w_{t-1} - \eta'_t g_t$ η' is different for each weight at each iteration

$$\eta'_t = \frac{\eta}{\sqrt{\alpha_t + \epsilon}}$$

ϵ is a small value to prevent 0 division errors.

η is a normal learning rate value.

$\alpha_{t-1} = \sum_{i=1}^{t-1} g_i^2 \longrightarrow$ always +ve $\Rightarrow \alpha$ is always +ve.

$\alpha_t > \alpha_{t-1} > \alpha_{t-2} \cdots$

as $t \uparrow$, $\alpha_t \uparrow$, $\eta'_t \downarrow$

as iterations increase, learning rate decreases adaptively.

<u>Advantages</u> :-

→ No need of manually tuning η

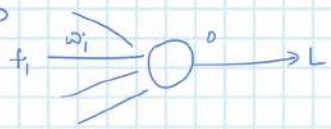
→ if there are some sparse & some dense features, adagrad automatically chooses the best learning rate.

<u>Disadvantage</u> :-

→ As $t \uparrow$, α_t could become very large & η' becomes very small $\Rightarrow w_t \approx w_{t-1}$ leads to slower convergence.

→ to avoid getting stuck at saddle points, add small value ϵ to optimization problem.

→



f_1 = input to neuron with weight i

if f_1 is sparse $\Rightarrow$ mostly NAs

in mini batch SGD, lot of batches contain mostly NAs & only few valid values.

ex:- let loss be squared loss

$$\Rightarrow \left(\frac{\partial L}{\partial w}\right)_{w_{t-1}} = \sum_{i=1}^{n} \underbrace{(-2x_i)}(y_i - w_{t-1}x_i)$$

↳ will be 0 for sparse features.

$\Rightarrow \left(\frac{\partial L}{\partial w}\right)_{w_{t-1}}$ will be much smaller for sparse than dense features.

∴ Adagrad will help with this.

Optimizers : AdaDelta and RMSProp (2.12) :-

→ In adagrad :- if α_{t-1} becomes very large, η becomes low, convergence become slower.

$$\eta'_t = \frac{\eta}{\sqrt{\alpha_{t-1} + \epsilon}} \qquad \alpha = \sum_{i=1}^{t-1} g_i^2$$

→ In AdaDelta, we replace avg with exponentially decaying average so as to avoid small η values & avoid slow convergence.

$$\Rightarrow \eta'_t = \frac{\eta}{\sqrt{eda_t + \epsilon}}$$

$$eda_t = \gamma\, eda_{t-1} + (1-\gamma) g_t^2$$

if $\gamma = 0.95$, then $eda_t = 0.95\, eda_{t-1} + 0.05\, g_t^2 = 0.05\, g_t^2 + \left[0.95\{0.05\, g_{t-1}^2 + 0.95\, eda_{t-2}\}\right]$

$$= 0.05\, g_t^2 + (0.95 * 0.05) g_{t-1}^2 + (0.95)^2 eda_{t-2}^2$$

We are controlling the growth of α

Adam (2.13) :-

→ In AdaDelta, we are taking the eda of g_t^2.

In Adam, we take eda of g_t.

Adam = Adaptive Moment Estimation.

In statistics : mean is aka 1st order moment (Adam more like 1st order)

variance is aka 2nd order moment (AdaDelta more like 2nd order)

mean at time t, $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$ $\quad 0 \le \beta_1 \le 1$ $\longrightarrow$ eda g_t

$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$ $\quad 0 \le \beta_2 \le 1$ $\longrightarrow$ eda g_t^2

β_1 is typically 0.9

β_2 is typically 0.99

α is typically 0.001

$$\hat{m}_t = \frac{m_t}{1-(\beta_1)_t} \qquad \hat{v}_t = \frac{v_t}{1-(\beta_2)_t}$$

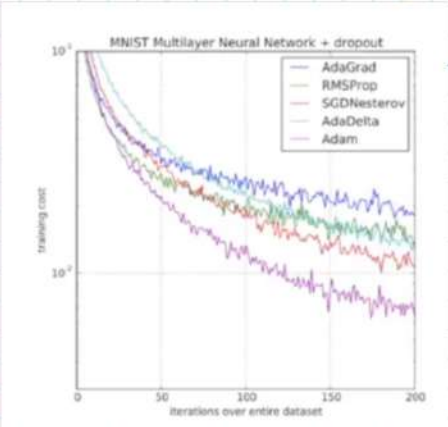
$$w_t = w_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{v_t} + \epsilon}$$

very similar to ada delta

if $\beta_1 = 0 \Rightarrow$ Adadelta

$\beta_1 = \beta_2 = 0 \Rightarrow$ not exactly adadelta but similar.

→ Adam has all the advantages as that of Ada Delta.



→ Adam was the fastest.

→ Adam has the problem of overshooting the minima.

NADAM = NAG + Adam + RMSProp.

→ As of 2021 AdaBound an Adam Variant that employs dynamic bounds on learning rates to achieve gradual & smooth transitions to SGD, is faster.

When to choose which algorithm (2.14):-

Mini Batch SGD — Momentum & NAG — AdaGrad — AdaDelta & RMSProp — Adam.

- Mini Batch SGD: good for small neural networks. Can't get out of saddle points.
- Momentum & NAG: work well in most cases. But slow.
- AdaGrad, AdaDelta & RMSProp: good for sparse data.
- Adam: Best and fastest.

Gradient checking & Clipping (2.15):-

→ good habit to monitor weights & gradients for each epoch, weight & layer.

→ helps detect vanishing gradient / exploding gradient.

→ Solution to exploding gradient = clipping:-

$w =$ [ ] [ ] [ ] . . . [ ] [ ]

w_{11} w_{12} w_{13} $\qquad$ w_{1n}

$G =$ [ ] [ ] [ ] . . . [ ] [ ]

$\frac{\partial L}{\partial w_{11}}$ $\frac{\partial L}{\partial w_{12}}$ $\qquad$ $\frac{\partial L}{\partial w_{1n}}$

L_2 Norm clipping $\longleftarrow$ $G_{new} = \frac{G}{\|G\|_2} * \tau$ $\qquad$ τ = threshold

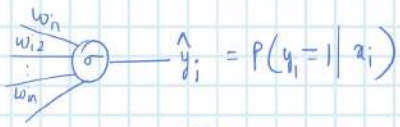
$\|G\|_2 \longrightarrow L_2$ Norm

All values of G_{new} will be $< \tau$

Softmax and Cross Entropy for Multi Class Classification (2.16) :-

→ Logistic Regression on its own works only for binary classification.
for multiclass → one v/s rest.
Instead of one v/s rest ; log reg for multi class = softmax.

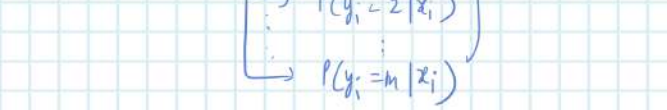
→ Logistic Regression from neural net perspective



$$z = x_i^T w$$

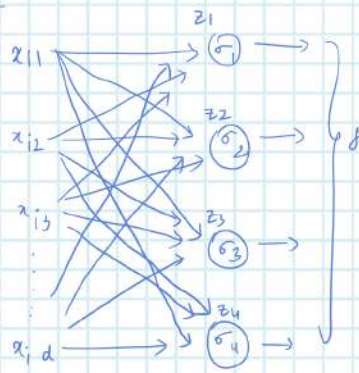
$$P(y_i = 1 | x_i) = \hat{y}_i = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{e^z+1}$$

→ $D = \{x_i, y_i\}$ $\quad y_i \in \{0, 1, 2, \dots m\}$



Soft-max :-

let m=4



$\underbrace{z_1, z_2, \dots z_m}_{\text{inputs to neurons.}}$

$$z_1 = \sum_{j=1}^{d} x_{ij} w_{j1} \qquad z_2 = \sum_{j=1}^{d} x_{ij} w_{j2} \quad \dots$$

$$\sigma_1(z_1) = \frac{e^{z_1}}{\sum_{i=1}^{m} e^{z_i}} \qquad \sigma_2(z_2) = \frac{e^{z_2}}{\sum_{i=1}^{m} e^{z_i}}$$

$$\sigma_1(z_1) + \sigma_2(z_2) + \dots = \frac{e^{z_1}}{\sum_{i=1}^{m} e^{z_i}} + \frac{e^{z_2}}{\sum_{i=1}^{m} e^{z_i}} + \dots + \frac{e^{z_m}}{\sum_{i=1}^{m} e^{z_i}} = \frac{\sum_{i=1}^{m} e^{z_i}}{\sum_{i=1}^{m} e^{z_i}} = 1 \quad \text{Sum} = 1$$

→ softmax minimizes multi class log loss aka cross-entropy

$$-\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} y_{ij} \log p_{ij}$$

$y_{ij} = 1$ if $x_{ij} \in y_{ij}$, 0 otherwise $\qquad p_{ij}$ = prob of belonging to y_{ij}.

How to train MLP (2.17) :-

① Preprocess : Data Normalization.

② weight initialization : (i) Xavier / Glorot → Sigmoid / tanh
(ii) He → ReLU
(iii) Gaussian → Small σ

③ Choose activation function :- ReLU as of now

④ Perform Batch Normalization (especially for deep MLP (later layers))
Dropout is compulsory.

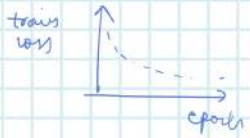
⑤ Optimizer : Adam (best one rn)

⑥ hyperparameters : lots of hyperparams in DL (Hyperc if you have GPU)

⑦ loss fn → 2 class → log loss
K class → multiclass log loss
regression → square loss

⑧ Monitor Gradients - Clipping

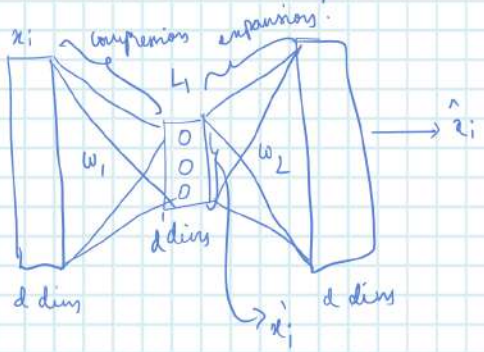
⑨ plot :



⑩ Avoid overfitting - Easy to overfit in deep learning because lots of weights.

Autoencoder :- (2.18)

→ Neural network which can perform dim-red.

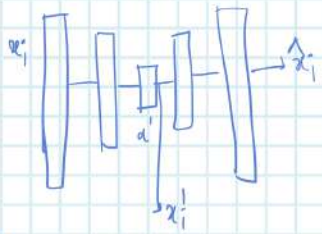


if the d' outputs from L_1 ie, $\hat{x}_i$ accurately represent x_i then x_i' encodes all of the info of x_i

$$L(x_i, \hat{x}_i) = \|x_i - \hat{x}_i\|^2$$

if $x_i == \hat{x}_i$, $L = 0$.

→ just like MLP, we can have deep autoencoders ie, multiple hidden layers



Denoising Autoencoder :-

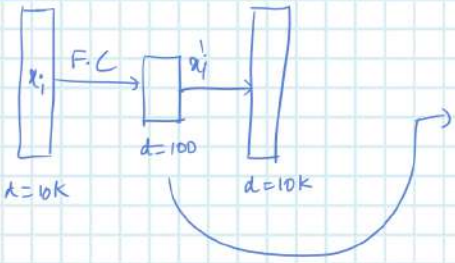
$x_i = \{\tilde{x}_i, \ldots \tilde{x}_n\}$ → noisy dataset

AutoEncoder $(x_i) = \tilde{x}_i$ → noise free, robust. Because it reduces the number of dimensions

$\mathbb{R}^d \quad \mathbb{R}^{d'}$

→ if original dataset is already noise free, we add noise to it ourselves and denoise it. Because test data points may not be noise free. Model can learn this.

Sparse Auto Encoder :-



→ if we want this to be sparse, we can use L_1 reg to the x_i'

→ If there's only one hidden layer that only uses sigmoid act. functions, the AE is strongly related to PCA.

→ AE can also be seen as unsupervised feature extraction process.

eg :- MNIST 784 → 50 dimension features.

Word2Vec : CBOW (2.19) :-

→ Not a neural network.

→ The cat sat on the wall.

'sat' is focus word

these are context words

→ word2vec uses 2 algos

(i) CBOW :- Continuous Bag of Words.

(ii) Skipgram.

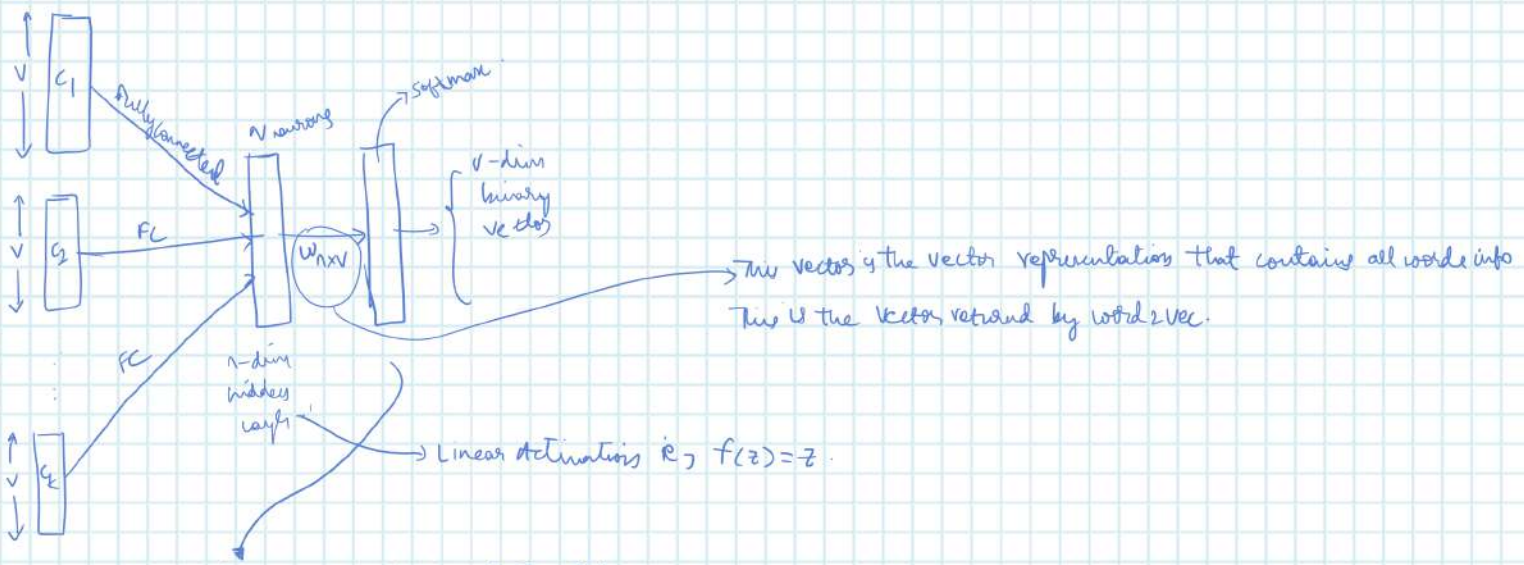
CBOW :-

① dictionary / vocab of words = V

v = length / size of vocab

② one hot encoding each word

$\omega_i \in \mathbb{R}^V$



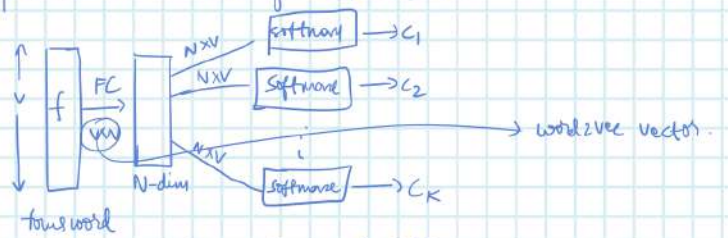
core idea: predict focus word using context words.

⇒ Multi class classification problem.

→ training is done by taking all possible combinations of focus words & context words.

Word2Vec (Skipgram) (2.20) :-

→ predict context words given focus words.



K - Multi class classifiers.

→ CBOW : 1 softmax

Skipgram : K softmax

∴ skipgram is computationally more expensive.

→ CBOW → faster than skipgram

→ better for frequently occuring words

Skipgram → can work with smaller amounts of data.

→ works well for infrequent words.

K = # of context words.

K ↑ ⇒ more context ⇒ N-dim vector for each point is better

N is typically 200/300.

→ # of weights to be trained is large in both $\approx (K+1)(N \times V)$

So we need to find algorithmic optimizations.

Word2Vec (Algorithmic Optimizations) (2.21) :-

→ lots of weights to be trained.

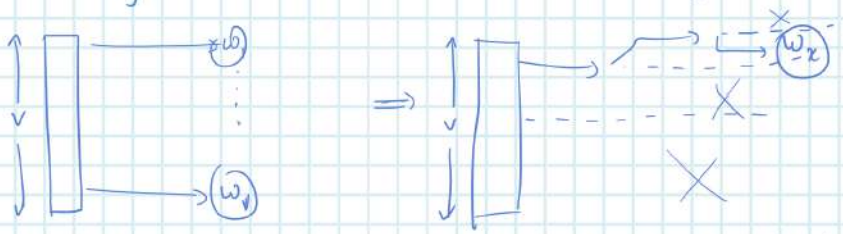
→ 2 optimizations

(i) Hierarchical softmax (algo)

(ii) Negative sampling (statistical)

Hierarchical :-

→ idea is to modify V-softmaxes to make it optimal (skipgram)

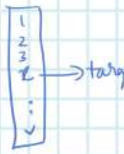


binary tree like structure. We find correct half by backpropagation as correct half results in less loss.

$= \log(V)$ softmax classifiers required.

Negative Sampling :-

→ update only a sample of words per iteration



→ target word (keep this, sample rest).

↳ always keep the target

↳ sample non target words with $P(w_i) = \sqrt{\frac{z}{freq(w_i)}}$ → 10^{-5}

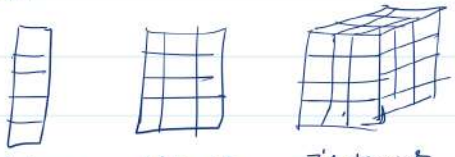
Tensorflow and Keras : Overview (3.1) :-

→ Best deep learning library as of now. Released in 2015. Before that people used Theano.

→ Can be used for research, development & deployment as well.

→ Core implementation is done in C/C++. Interfaces built in Python, Java and Javascript.

Tensor :-



1d tensor 2d tensor 3d tensor

tensor = n dim array.

Deep learning is all about tensor operations.

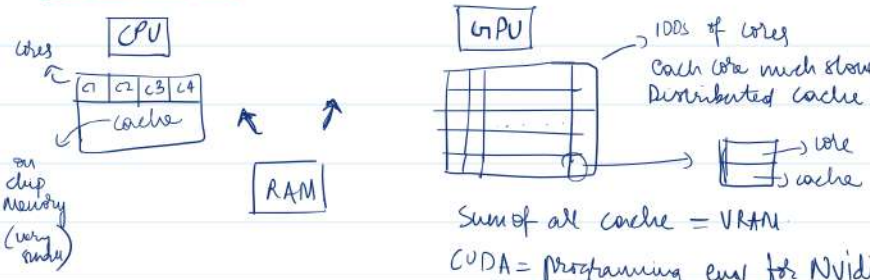
→ there is a lot of low level control in TF. Steep learning curve.

→ Keras has a lot of high level control & is easier to use. Keras internally uses tensorflow as well.

There are 2 backends for Keras : tensorflow and Theano.

→ Keras 2 is more deeply integrated with the tensorflow and has some low level control as well.

CPU v/s GPU (3.2) :-



Sum of all cache = VRAM

CUDA = programming env for Nvidia GPUs.

GPU can do lots of parallel computations.

GPUs are better if we have parallelizable tasks. Deep learning is highly parallelizable.

Softmax Classifier on MNIST (3.6) :-

→ Tensorflow has $\underbrace{\text{constants, variables}}_{\text{as seen in python}}$ and placeholders.

ex:- $\underline{W}^T X + \underline{b}$ (variables)

placeholder :- typically used to store data like mini batches etc, Memory allocated for the task.

Think of it as a modified form of variables

Rule of thumb for number of neurons, $N_h = \dfrac{N_S}{(\alpha * (N_i + N_o))}$

N_S = Number of samples in training dataset

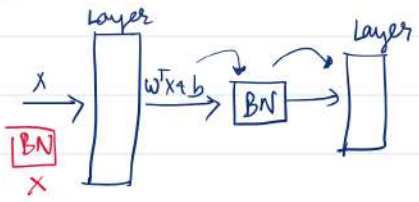
N_i = Number of input neurons

N_o = Number of output neurons

α = arbitrary scaling factor 2-10. sometimes 5-10.

Model 3 : Batch Normalization (3.10) :-

→ In batch normalization x's are not the ones that need to normalized. It's the $(W^T x + b)$ that should be normalized.



Biological Inspiration : Visual Cortex (4.1) :-

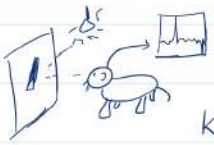
aka CNN / ConvNets.

→ Mainly used for visual tasks. such as object recognitions etc,

→ Breakthroughs in (DL + CV) happened in the last 10 years.
↳ Computer Vision

→ Hubel & Wiesel experiments in 1950-80s :-

on animals. certain neurons had more activity when light edge is of a shape.

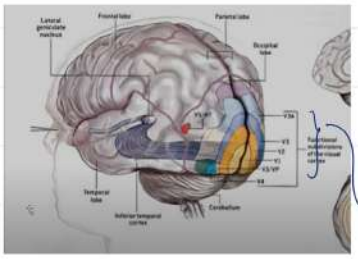


ex :- ▯ → spike in neural activity ⧸ → medium spike ▭ → no activity

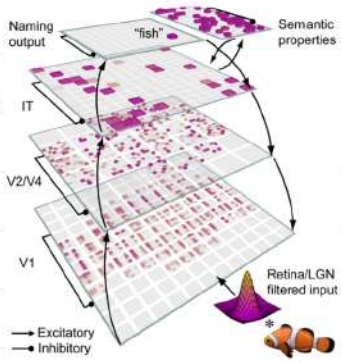
Key findings :- ① Some neurons in the visual cortex are fired when presented with light at a specific angle. ex :- V_1 : primary visual cortex → detects edges.

② There are neurons for edge detections, motions, depth, color, faces, shapes etc,

③ There is a heirarchical structure among visual cortices.



visual cortices.



→ Lots of image processing & CV is inspired by Hubel & Wiesel research.

→ ConvNets take advantage of this research & biological structure

Convolution : Edge detection on images (4-2) :-

→ performed by primary visual cortex (V_1)

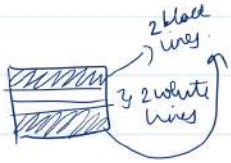
ex :- → image

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 & 255 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ -1020 & -1020 & -1020 & -1020 \\ -1020 & -1020 & -1020 & -1020 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{min max norm}} \begin{bmatrix} 255 & 255 & 255 & 255 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 255 & 255 & 255 & 255 \end{bmatrix}$$

Sobel horizontal edge detector

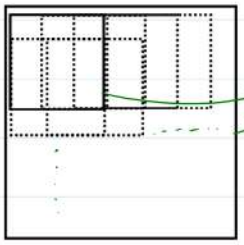
not multiplication convolution.

kernel / Filter / Mask operation



→ Sobel vertical edge detector :

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$



→ kernel
→ input
element wise product and their sum = $\mathbb{R}^1$

A color picture of a steam engine

The Sobel operator applied to that image

$$G_x = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Add matrices obtained from both for both horizontal & vertical

→ Kernel size is not fixed. Not always 3x3

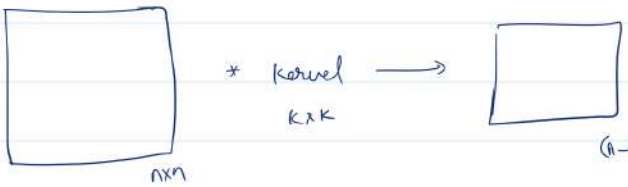
→ Kernel values can also change.

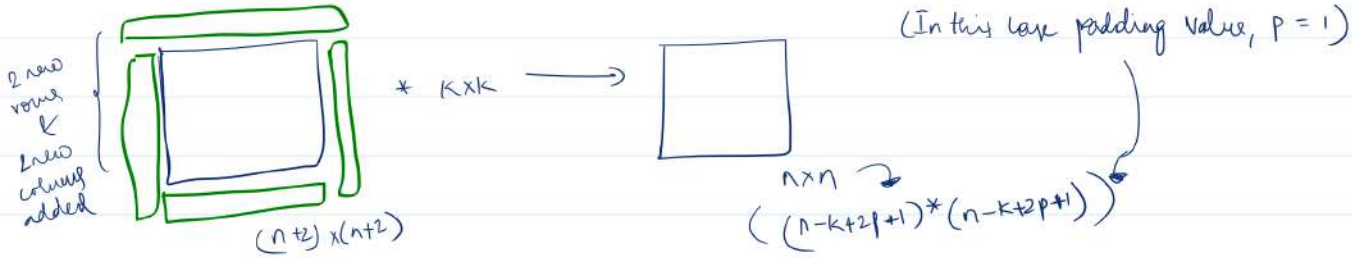
→ Box filter = $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ → blurs the image when sum is divided by number of coeff.

$$\begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix}$$ → blurs the image directly.

Convolution : Padding & Strides (4.3) :-

→

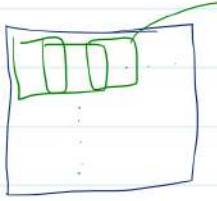




→ Padding can be filled by (i) Zeros : Most common. Adds more edges to the image through. These edges are not learned again in backpropagation.

(ii) Same Value Padding : Not common. Causes conflicts in case of multiple neighbours

Stride :-



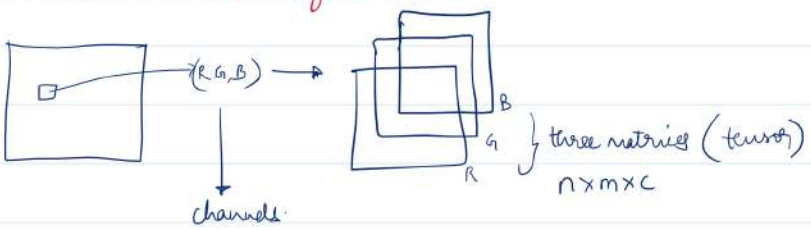
→ Moving 1 by 1 = stride of S = 1

$$n \times n \xrightarrow[K \times K]{\text{stride 's'}} \left(\left\lfloor \frac{n-k}{s} \right\rfloor + 1\right) * \left(\left\lfloor \frac{n-k}{s} \right\rfloor + 1\right)$$

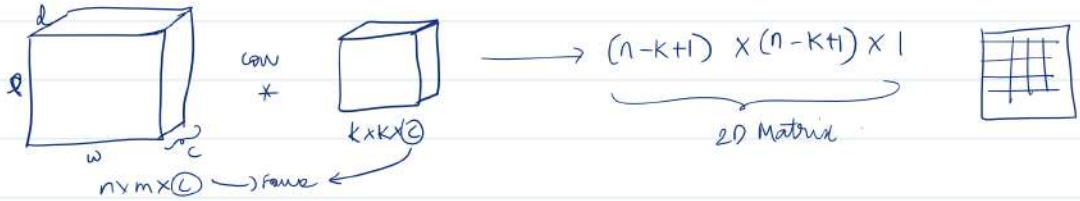
$$\rightarrow n \times n \xrightarrow[\text{padding } p,\ \text{stride } s]{K \times K \text{ kernel}} \left(\left\lfloor \frac{n-k+2p}{s} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{n-k+2p}{s} \right\rfloor + 1\right)$$

→ Padding if bigger output, stride for smaller output.

Convolution over RGB Images (4.4) :-

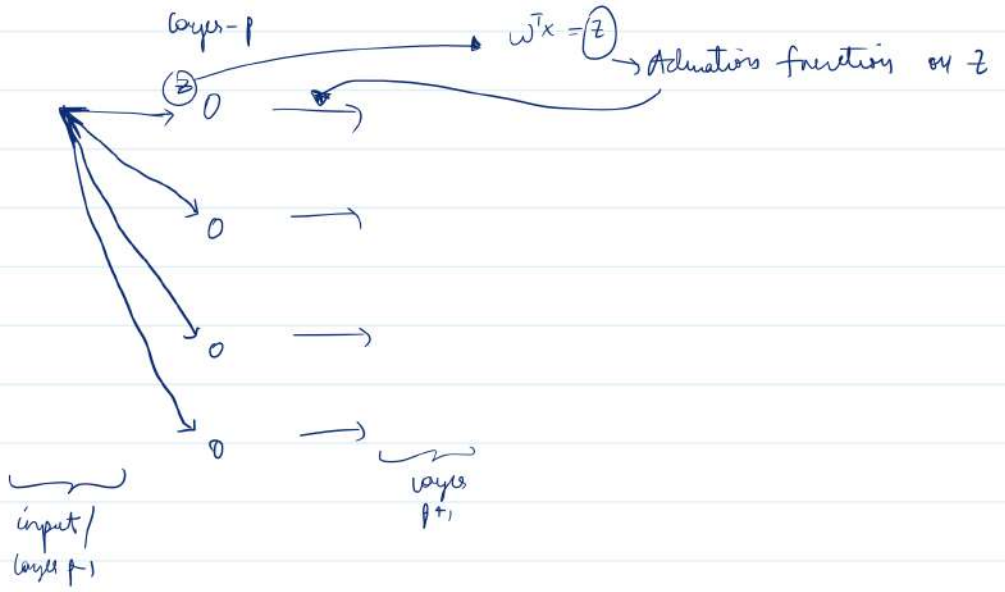


→ Convolution performed same as in 2D ie, component wise multiplication and addition



Convolution layer :- (4.5)

→ In normal MLP,



→ But in CNNs,

multiple edge detectors ⇒ multiple kernels

(1 edge detector = 1 kernel)

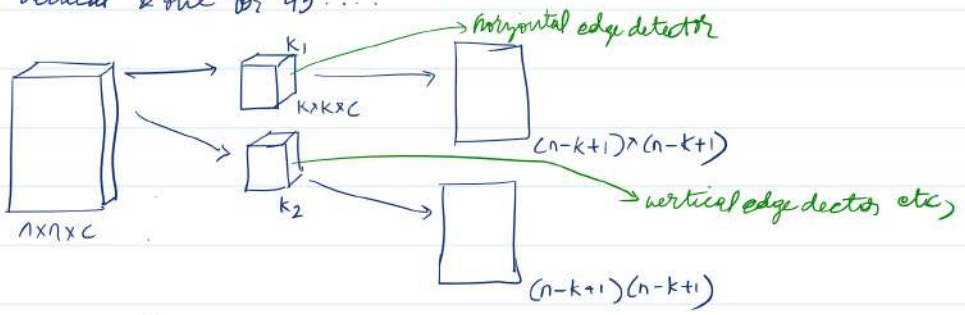
So in CNN, we learn the kernel matrices in backpropogation

MLP : learn weights

CNN : learn kernel matrices. ($I_{k \times k \times k}$)

for ex:- instead of hardcoding one kernel to identify edges, we could have one kernel for horizontal edge, one for vertical & one for 45°. . . .

$D = \{x_i, y_i\}$



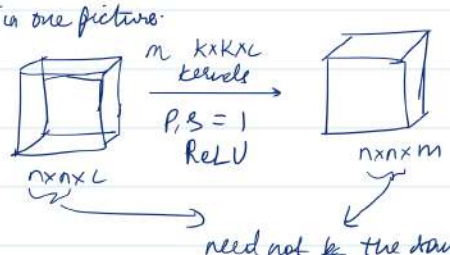
↳ kernel, padding & stride are hyperparams.

① Input $(n \times n \times c)$ * Kernel $(k \times k \times c)$ $\xrightarrow{p,s=1}$ $n \times n \times 1$

equivalent to $w^T x$

element wise ReLU →

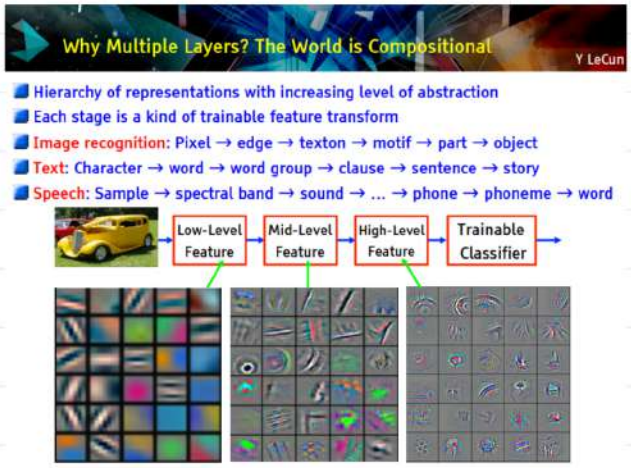
→ in one picture.



need not be the same always.

→ In a deep multi layer CNN, works like human visual cortex.

first layer identifies pixels, second edge, third texture, ..., part, object etc)



→ Using ReLU is not mandatory in CNNs.

→ kernels are initialized in the same way as in MLP.

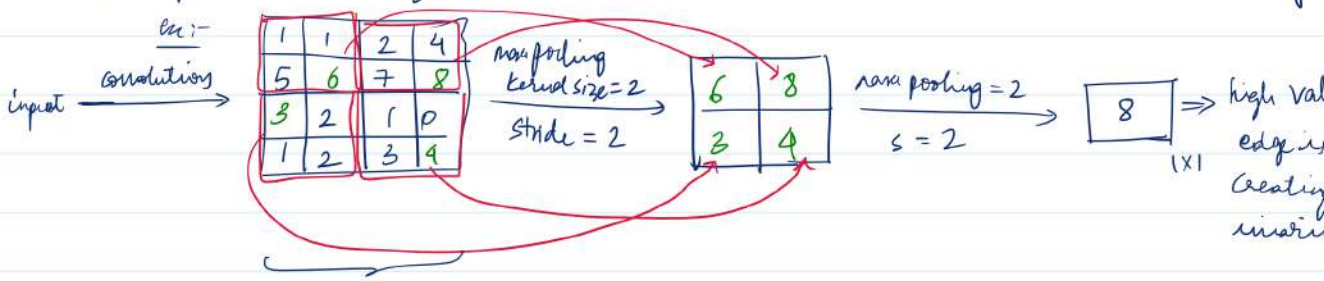
Max Pooling (4.6) :-

→ Location invariance :- We want model to detect object irrespective of it's location in image. Similarly scale invariance, rotation invariance etc)

In human body at higher levels of visual cortex, (V_7 etc) neurons are fired as soon as object is recognised irrespective of scale, size etc) Max pooling tries to replicate this

→ Max pooling can also be thought of as a kernel. Finds the max value in the kernel shape & size.

Ex :-



⇒ high value meaning edge is present somewhere. Creating location invariance.

↳ Since this is an output of convolution matrix, the value will be high if there is an edge present. Otherwise it will be a small value.

→ Very popular in ConvNets. Avg-Mean pooling is also gaining popularity.

→ Capsule Network captures both presence of an object & it's locale data.

If eye, ears, mouth & nose positions are jumbled, CNN will recognise a face, but CapsNet won't.

→ Repeated kernels automatically get rid of scale invariance. Data augmentation takes care of rotation invariance.

CNN Training : Optimization (4.7) :-

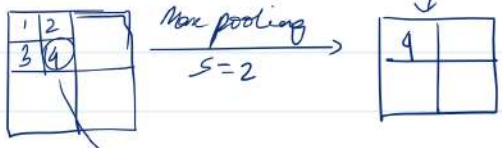
→ In order to calculate derivatives, operations need to be differentiable.

CNN operations :- Convolution Layer :- Convolution + ReLU → differentiable

Max Pooling

Convolution → dot product + addition intuitively. In MLP, it's element wise operations. In ConvNets, it's pixel wise operations.

derivative of 4 is only passed to 4 in conv matrix.



ie, $\max(x) = \frac{\partial g}{\partial x_i} = \begin{cases} 1 & \text{if } x_i = \max(x) \\ 0 & \text{otherwise} \end{cases}$

→ derivative is passed only to this value. This is done because no matter what happens to other vals, it doesn't affect maxpool thereby the end result.

→ Same thing happens to mean pool/avg pool.

→ We can use dropouts on CNN as well just like in MLPs.

Read first comment links for more content.

→ if we invert kernel matrix in correlation by 180°, it becomes convolution operation

ie,

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \Rightarrow \begin{bmatrix} d & c \\ b & a \end{bmatrix}$$

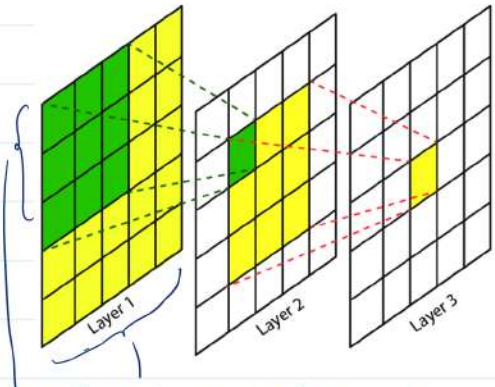
→ convolution.

→ Max pooling selects the brightest pixels from the image. Useful when image is dark & we are interested in only lightest pixels of image. Mean pooling smoothens the image & sharp features may not be extracted.

Receptive Fields and Effective Receptive Fields (4.8):-

Receptive Field :- The region on image on which kernel operates on at a given time 'T'.

Effective Receptive Field :- In deep CNN, the region on original image on which kernel at a deeper layer indirectly works on.



→ Receptive field for layer 2.

→ effective receptive field for layer 3.

ImageNet (4.10):-

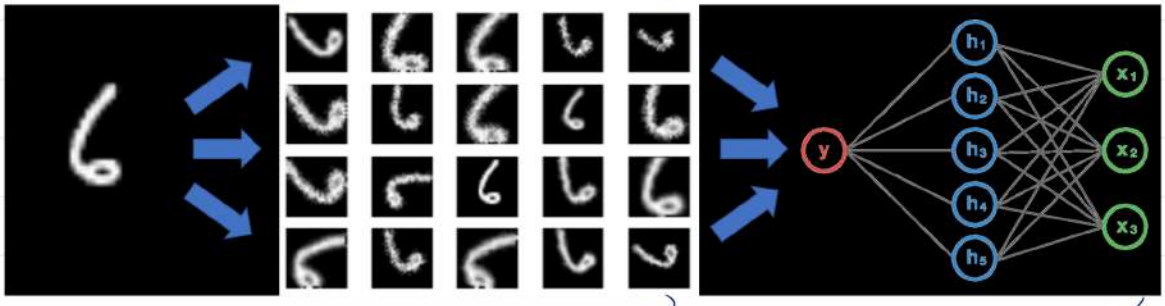
→ CNN used AlexNet in 2012 gave very good result.

→ AlexNet vs LeNet.

Data Augmentation (4.11):-

→ We want the models to be robust & work even when image is rotated, scaled etc.,

x_{ij} . y_{ij} is going to be the same.

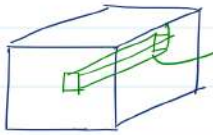


noise, scaled, rotated, sheared, cropped etc., CNN architecture

→ Why? We can get invariances like scale invariance, rotation invariance etc. Also can increase the size of the dataset.

→ Augmentation prevents model from overfitting by introducing divergence to the dataset.

AlexNet (4.13):-



input matrix

→ only this data normalized = local response normalization. Not used these days. Replaced by BatchNormalization.

→ AlexNet used dropouts, ReLU & multiple GPUs to train. Gave good score.

→ Objective of dense layer is to take results from convolution/pooling and use them to classify image into a label.

VGG Net (4.14):-

→ By Andrew Zisserman (key figure in CV) AlexNet uses all sorts of kernel sizes. VGG tries to use simple fixed kernel sizes 1C, 3X3 kernels, $S=1$, $P=$'same' and maxpool $S=2$, $2X2$.

→ VGG = Visual Geometry Group.

→ 2 types VGG-16 & VGG-19. Default networks used these days.
16 layered & 19 layered network.

→ Pretrained available in Keras.

→ In training, weight decay in regularization $= 5 \times 10^{-4}$ & dropout ratio $= 0.5$.

→ Regularization can be applied at all layers including conv layers because they contain parameters too.

→ In AlexNet MaxPooling done after almost every conv layer. In VGG, it's done after 2-3 conv layers.

ResNet (4.15):-

→ Residual Networks. By Microsoft.

← Even after adding lots of layers, error's not reducing even after dropout, lots of iters etc.)

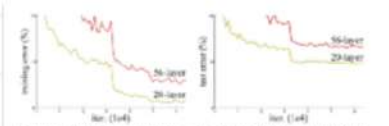
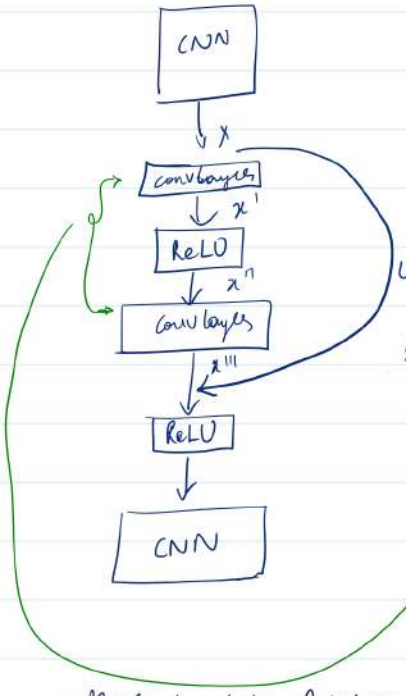
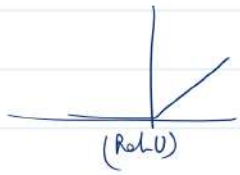


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.



→ Doesn't necessarily have to be conv layers. Can be dot prod (MLP) too.



if x is +ve, $ReLU(x) = x$

$ReLU(ReLU(x)) = x = ReLU(x)$

if x is -ve, $ReLU(x) = 0$

$ReLU(ReLU(x)) = 0 = ReLU(x)$

identity/residual block. Skip connections.

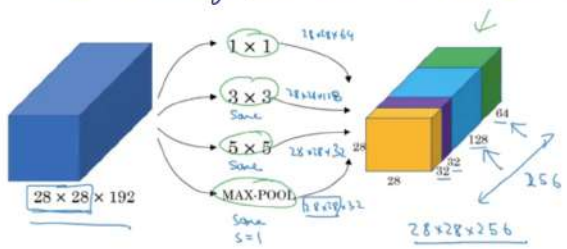
x is added to x'''

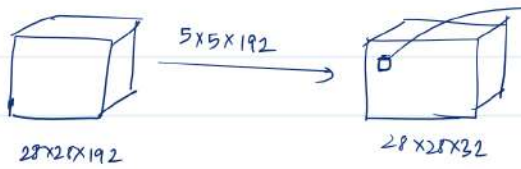
→ if because of some regularization, these 2 become 0 (weights), then $x''' = x$ instead of 0. ie, output will not be impacted & adding additional layers would not hurt performance - we'll skip over them.

→ Must be before ReLU & x & x''' must be same size ie, padding should be taken care of in conv layers.

Inception Network (4-16) :-

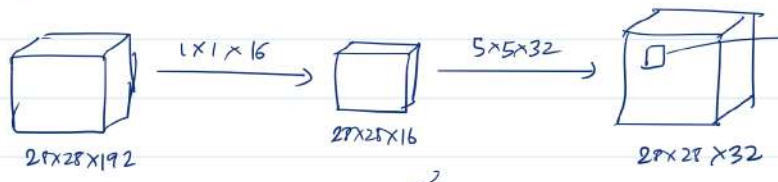
→ Instead of using just one kernel, use multiple kernels at once.



→ 

→ In this all $(28\times28\times32)\times(5\times5\times192)$ computations $= 120$ Millions

But instead

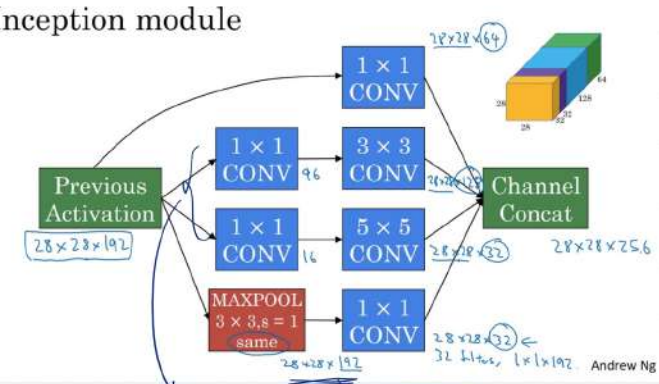


→ $\left(\left((28\times28\times16)\times(1\times1\times192)\right) + \left((28\times28\times32)\times(5\times5\times16)\right)\right)$

$=$ only 12.4 millions

→ Bottleneck layer.

Inception module



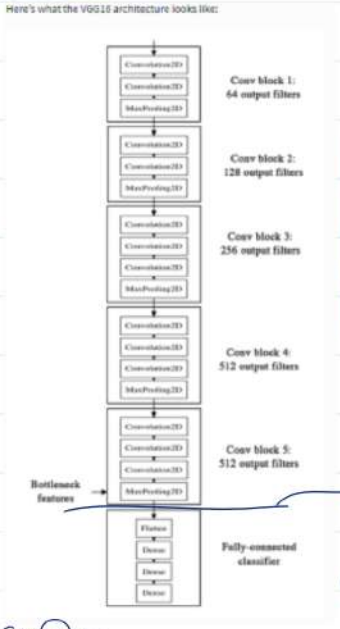
Bottleneck layers.

→ This would dominate the conv kernels above it. So we use 1×1 to reduce 192 to 32.

Transfer Learning (4.17):-

→ Instead of building NN classifier from scratch, we can reuse existing models trained on diff dataset (like VGG16). We "transfer" learnings from one model & use it for our use case.

Case ①:-

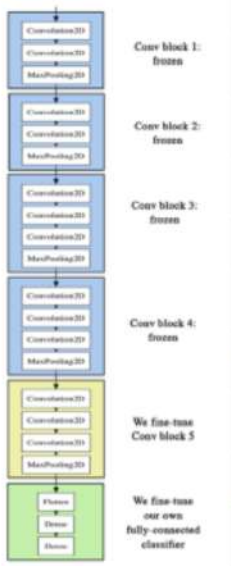


We only use these layers

→ Bottleneck layer. By this layer we have a vector representation of each image.

X ie, we use VGG as a sort of feature engineering tool. till flatten if classification problem or till bottleneck if segmentation problem (goal is to divide image into segments).

Case ②:-



We "freeze" these layers as they are anyway needed.

We modify these layers acc to our needs using our dataset D. We should ensure learning rate is small so things dont change drastically.

Case ③:- VGG16 + Imagenet to initialize new model & tune model entirely using our dataset D. Learning rate small

Case ④:- retrain model from scratch. (Bad & expensive)

→ which case to choose depends on size of our dataset

Case ①:- When D ≈ Imagenet & |D| is small

Case ②:- |D| ≈ Imagenet & |D| is large. Finetune the NN

Case ③:- |D| = reducing & D ≈ Imagenet. finetune last layers.

Case ④:- |D| = small & D ≉ Imagenet

→ $\boxed{C_1}$ — $\boxed{C_2}$ | $\boxed{C_3}$ — $\boxed{C_4}$ — $\boxed{C_5}$ — $\boxed{D}$ →

use only till here.

Case ⑤:- |D| = large & D ≉ Imagenet, init model using VGG16 & retrain.

fine tune = starting from a trained network, retraining on new dataset using very small update changes.

General trend in designing NN architecture

CNN → pool → CNN → pool → Dense → Dense (sigmoid/softmax)

Can replace with any number of dense layers as long as it makes sense.

Can also be refined as CNN – BN – Activation – Dropout – pool → . . . . → Dense → Output.

Face Recognition from scratch in CNN:-

→ 1000 images per class is good for CNN (if doing without transfer learning)

→ last couple of layers with 50-100 images per person. Data Augmentation as well.
(Transfer learning) ↳ of VGG. ↳ Realistic Setting (glasses, lighting etc.) and various angles.

→ Good loss would be cross entropy & softmax last layer.

→ If no time, use cloud APIs that are pre trained. (ex - MS Advanced Facial Recognition System)

Why RNNs (5.1):-

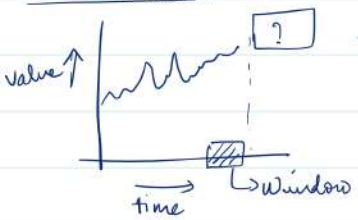
→ MLP :- $x \longrightarrow \boxed{MLP} \longrightarrow y$

CNN :- $x \longrightarrow \boxed{CNN} \longrightarrow y$
(Image)

Sequences ?

eg :- This phone is very fast } Both have the same TFIDF/BOW representation. Discarded sequence info.
① Phone this is very fast } But both are not the same.

② Time Series Data :-



→ Used windows and fourier transforms previously.

③ Machine Translations :- Language translations. The sentences hold sequential information. Using RNN we work with seq2seq models.

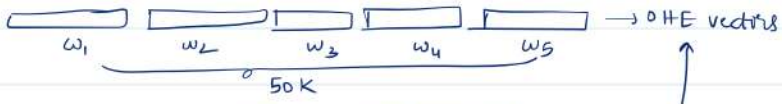
④ Speech Recognition :- Sequence of Audio → English sentence

⑤ Image captions. (Google Image Search)

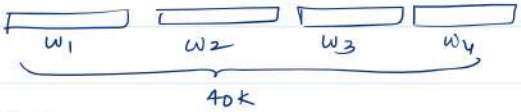
→ The idea is to build a NN that retains & leverages sequential information.

→ Problem with not using RNN & using an MLP directly is inputs are of different sizes

ex :- the review :- This phone is very fast
w_1 w_2 w_3 w_4 w_5



the review :- This phone is good
w_1 w_2 w_3 w_4



We could do zero padding to make all inputs equal but
(i) Does not work for longer test queries
(ii) Unnecessarily increasing dimensionality

Recurrent Neural Network :- (5.2)

→ Recurrent = repeats.

→ Taking example of Amazon Fine Food Reviews

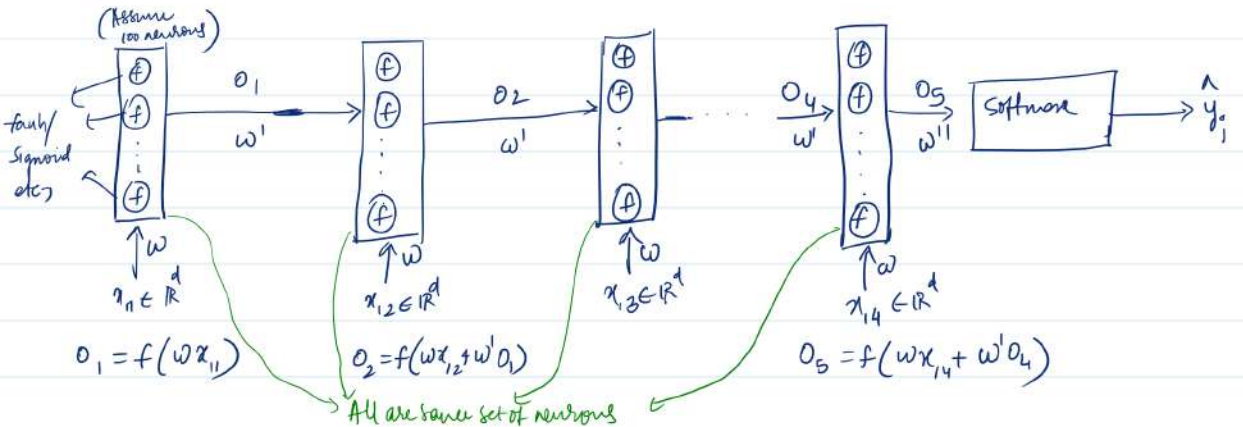
$D = \{x_i, y_i\}$

$x_i \rightarrow x_{i1}, x_{i2}, \ldots x_{im_i}$ where m_i = len of x_i

Assume size of Vocabulary = d

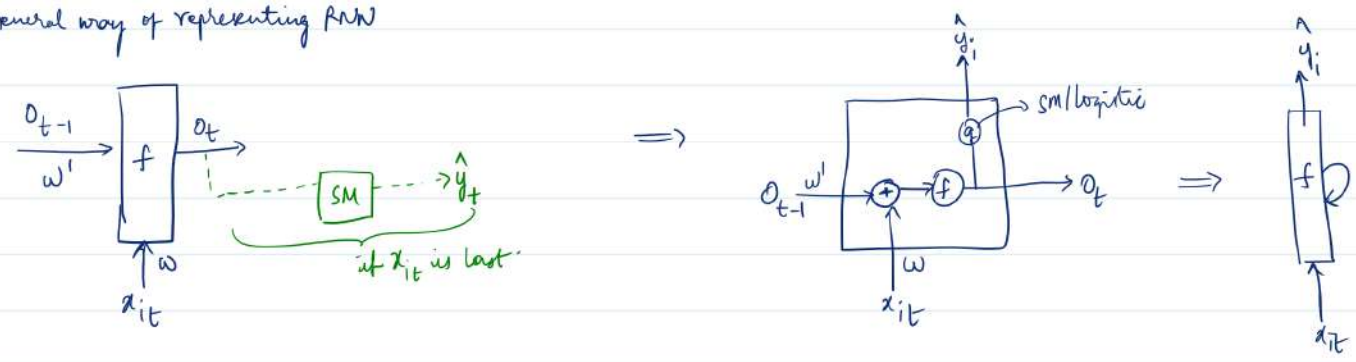
$x_{ij} \in \mathbb{R}^d \rightarrow$ Binary vector (one hot encoding) ; $x_1 = \langle x_{11}, x_{12}, x_{13}, x_{14}, x_{15} \rangle$, y_1

At t=1 (iteration 1) → At t=2 → At t=3 ..... At t=5 $\qquad$ w' is same at all stages.



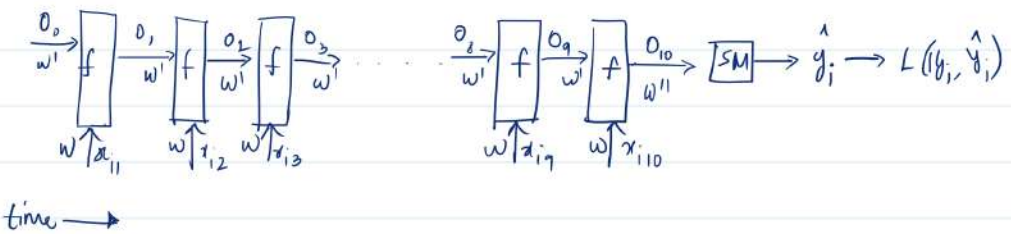
→ For the next sentence the same W are reused.

→ General way of representing RNN



Training RNNs : Backpropogation (5.3) :-

→ Backprop for RNN = Backprop over time



$O_1 = f(wx_{i1} + w'O_0)$

$O_2 = f(wx_{i2} + w'O_1)$

$\vdots$

$O_{10} = f(wx_{i10} + w'O_9)$

$\hat{y}_i = g(w'' \cdot O_{10})$

→ Forward propogation = direction of arrows (time)

Back prop : opposite direction.

$$\frac{\partial L}{\partial \hat{y}_i} \quad \Big| \quad \frac{\partial L}{\partial O_{10}} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial O_{10}} \quad \Big| \quad \frac{\partial L}{\partial w''} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w''} \quad \Big| \quad \frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial O_{10}} \cdot \frac{\partial O_{10}}{\partial w}$$

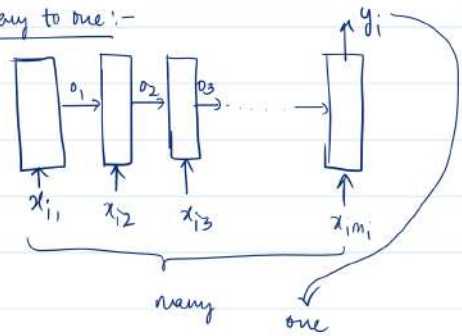
$$\frac{\partial L}{\partial w} = \left(\frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial O_{10}} \cdot \frac{\partial O_{10}}{\partial w}\right) + \left(\frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial O_{10}} \cdot \frac{\partial O_{10}}{\partial O_9} \cdot \frac{\partial O_9}{\partial w}\right) + \left(\frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial O_{10}} \cdot \frac{\partial O_{10}}{\partial O_9} \cdot \frac{\partial O_9}{\partial O_8} \cdot \frac{\partial O_8}{\partial w}\right) + \cdots \quad (\quad)$$

lots of multiplications involved

→ Since we are performing backpropogation over time & $\frac{\partial L}{\partial w}$ will have lots of multiplications based on number of words, we run into vanishing gradient & exploding gradient problem. This is NOT because of many layers but because of sequence (sentence) length. LSTMs & GRUs solve this problem.

Types of RNN Architectures (5.4) :-

① Many to one :-

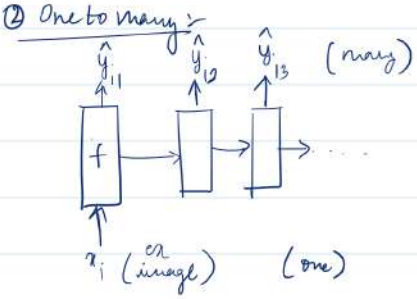


→ many to one.

example :- ① sentiment classification (food reviews)

② Movie review → star rating (multiclass classfn)

② One to many :-



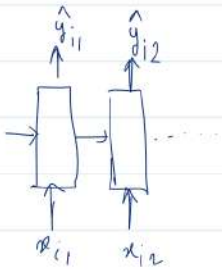
ex :- image captioning

→ "contains two human faces"

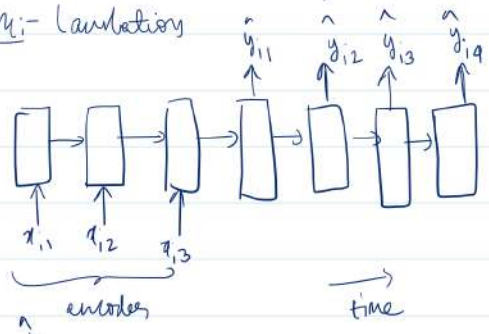
sequence

③ Many to Many :- input → seq
output → seq

type① :- input & output same length
ex:- parts of speech of sentence



type② :- diff - i/p & o/p
ex:- translation



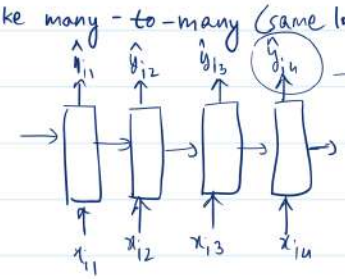
seq2seq models

④ one to one :-

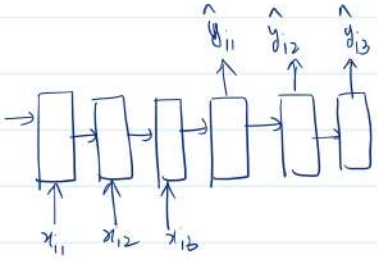
→ simple MLP. No sequence

Need for LSTM/GRUs (5.5) :-

→ Take many-to-many (same length for example)



→ $\hat{y}_{i4}$ in this model depends mostly on x_{i4}. It's value does not depend on x_{i1} because the gradient vanishes by the time it reaches the calculation of $\hat{y}_{i4}$.
But in real world there may be cases where $\hat{y}_{i4}$ is more dependent on x_{i1} than x_{i4}.
There's no long term dependancy in this case. (in simple RNN)
later output depends lot more on earlier inputs.

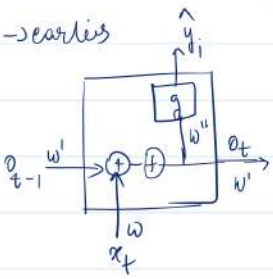


→ $\hat{y}_{i3}$ could depend on x_{i1} in case of translation. Simple RNN can't handle this.

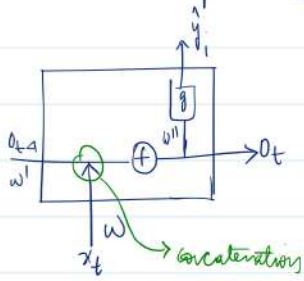
LSTM (5.6) :-

→ It can take care of both long term & short term dependancies.

→ earlier



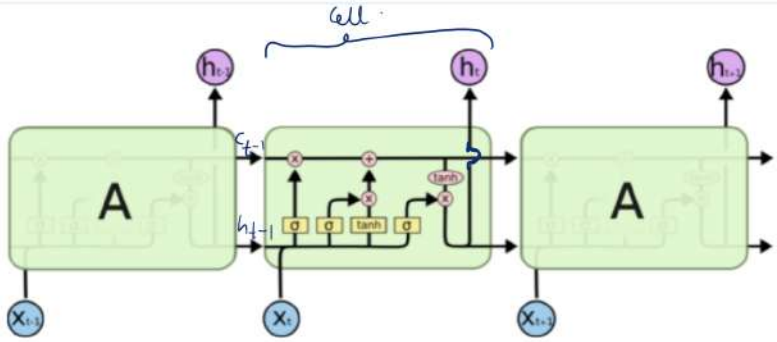
Now →



concatenation operation. Not addition like before.

$o_t = f(w x_t + w' o_{t-1})$
$\hat{y}_{it} = g(w'' o_t)$

$o_t = f([w', w][x_{it}\ o_t])$
$\hat{y}_{it} = g(w'' o_t)$



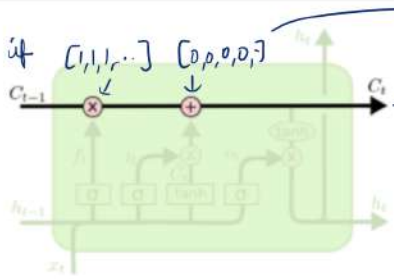
The repeating module in an LSTM contains four interacting layers.

3 outputs.

$h_t = y_t$

$x_t = x_t$

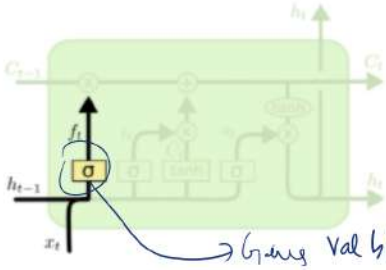
$c_t = o_t$



$c_t = c_{t-1}$ (Works like identity function)

→ This handles how much info from prev layer needs to be remembered (×) and how much from current data point is to be added (+)

Forget Gate Layer :-

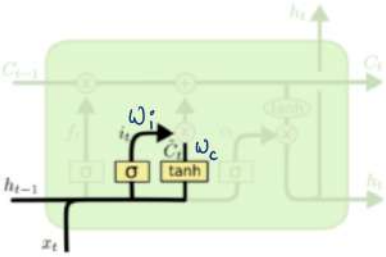


→ Gives val b/w 0 & 1.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

→ handles how much information we are going to throw away from the current layer.

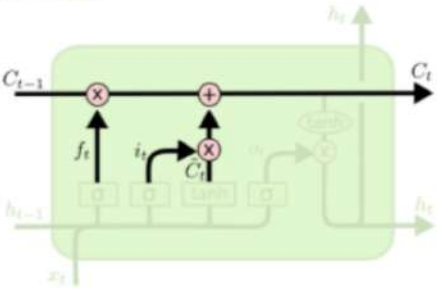
Input Gate Layer :-



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

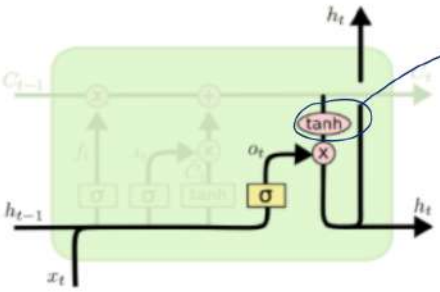
→ handles what new information we are going to store in cell state.

Cell state :-



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output layer :-



→ to get val b/w −1 & +1 so we can multiply with o/p of sigmoid gate & only o/p the parts we want to.

$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

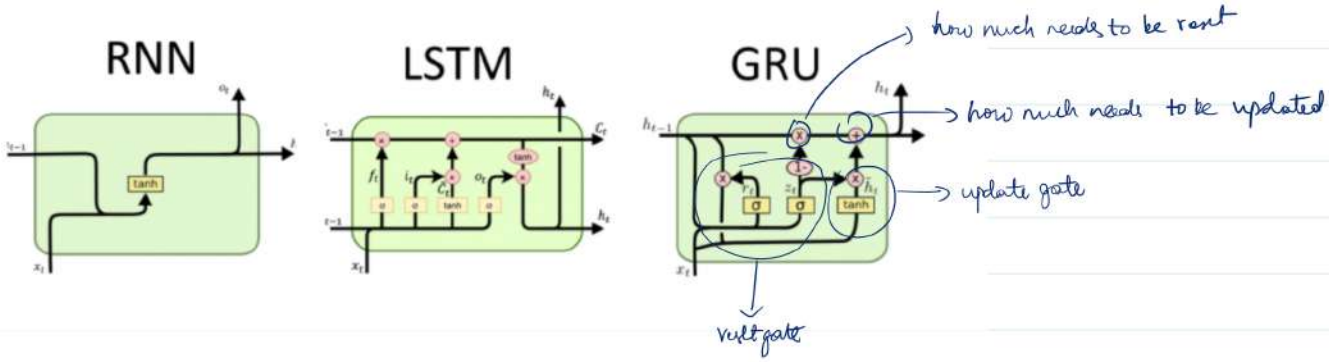
→ 4 weights. All are concatenations of x_t & h_{t-1}. All learn diff weights through.

GRU (Gated Recurrent Unit) (5.7):-

LSTM - 1997 , GRU - 2014.

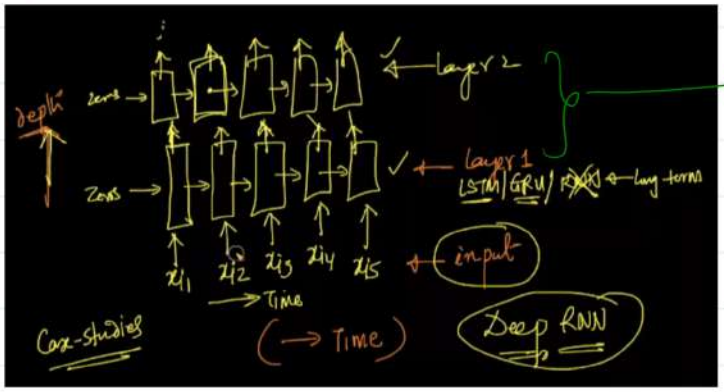
→ simplified & faster to train version of LSTMs. As powerful as GRU. Merges the cell state & hidden state.

2 gates → reset gate, update gate.



fewer equations ⇒ fewer updates ⇒ fewer backpropagation ⇒ faster.

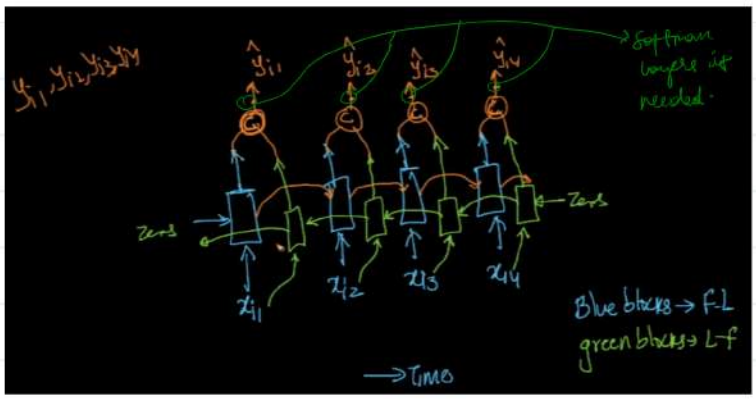
Deep RNN (5.8):-



→ Not always equal. Number of cells is a hyperparameter like neurons in MLP.

Bidirectional RNN (5.9):-

→ What if y_{it} is dependent on $x_{i\,t+k}$. Simple RNN can't work with such data.



Use cases:-

→ NLP tasks

→ Speech to text.

IMDB Text Classification (5.10):-

→ IMDB keras

$x_1 = \langle x_{11}, x_{12}, \ldots \rangle, y_1$

$x_2 = \langle x_{21}, x_{22}, \ldots \rangle, y_2$

V = set of all the words in all the reviews.

Step ①:-

word	freq
The	10000
a	20000
⋮	⋮

Step②:- sort by frequency

Step③:-

	word	freq
1	a	20000
2	an	18000
3	the	1000
⋮	⋮	⋮

give words numbers

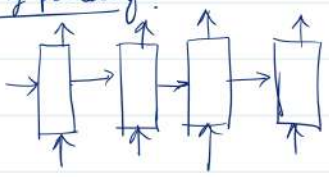
Step④:- x_1 = This phone is a good phone

$\langle 81, \ldots, 1, \ldots \rangle$

$\langle x_{11}, \ldots x_{14}, \ldots \rangle$

→ padding :- convert all reviews to the same length by inserting zeros at the beginning.

Why padding?



$x_1 = \langle x_{1,1}, x_{1,2}, \ldots x_{1,150} \rangle$

$x_2 = \langle x_{2,1}, x_{2,2}, \ldots, x_{2,100} \rangle$

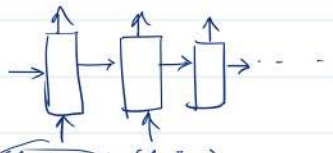
→ Normally, sent one i/p after another.

↓

Similar to SGD batch size = 1.

With padding, all elements have the same size.

⇒



as possible. Speeds up training.

$\{x_{11}, x_{21}, x_{31}, x_{41}, \ldots\}$ $\{x_{12}, x_{22}, \ldots\}$ ....

size depends on SGD batch size

→ embedding takes a +ve integer & returns a vector.

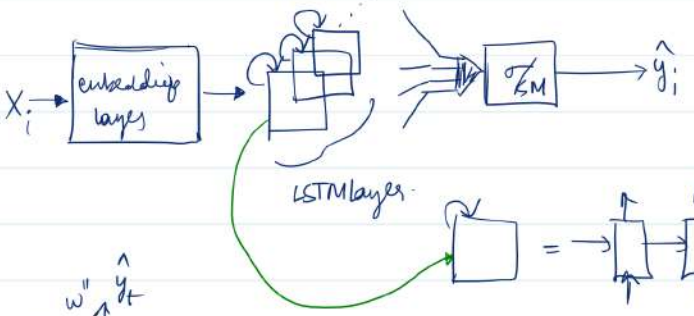
ex:- 1 → embedding → $[\underbrace{\ldots\ldots}_{\text{embedding vector length}}]$

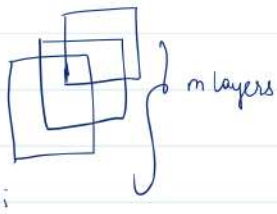
→ if there are m inputs & n outputs to LSTM,

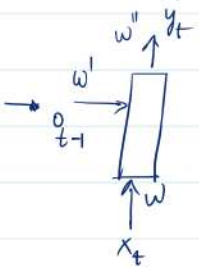
$\#\text{params} = 4(nm + n^2 + n)$

↳ bias (Keras by default adds bias)

→







$\hat{y}_t$ = Scalar

$w'' = m \times 1$

$w' = m \times m$

$w = n \times m$

$x_t = 1 \times n$

$o_{t-1} = 1 \times m$

4 gates exist (3 σ, 1 tanh)

each = $m \times m + n \times m = m^2 + nm = 4(m^2 + nm + m)$

↳ Bias

m = # of neurons in layer = # of "outputs"

n = # of dimensions of x_t

2)Also I'm finding it difficult to understand top level on how the process happens, so here the input we'll pass in x_train[0] right? It will be a vector of 600 dimension because of pad sequence . Then first embedding layer converts each feature of 600 features to 32 dimensions right? So it will be like a list of lists. Then LSTM take in each word(of 32 dim) i.e (first list in list of lists) and then takes the second word and so on and performs forward and backward prop and outputs the final binary decision for one sentence right? Is this how it works internally from a top level point of view. Please correct me if I'm mistaken anything.

Reply Jul 02, 2019 02:46 AM

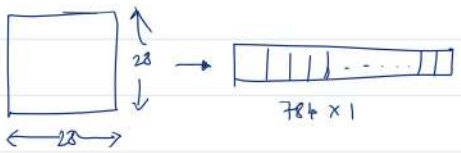
AppliedAI

yes, you are correct.

Generative Adversarial Networks (GAN) :-

→ Popular since 2018.

→ task:- Given MNIST images $[D = \{0,1,2,\dots,9\}]$, create/generate new MNIST images $D' = \{0,1,2,\dots,9\}$ that are similar to images in D but not the same.



simpler task:- Given $D = \{$set of heights of students in a class$\}$, generate $D' = \{h_1', h_2', h_3' \dots\}$

very similar

$D = \{h_1, h_2, \dots, h_n\}$

→ PDF $= P_\theta(H)$. Assume it's $N(150, 30)$

↳ could be normal / uniform / lognormal etc, can be determined using QQplot / KS-testing.

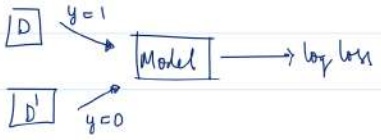
$\underbrace{\underbrace{\mu, \sigma}_{\text{normal}} \quad \underbrace{a, b}_{\text{uniform}} \quad \underbrace{\mu, \sigma}_{\text{lognormal}} \dots}_{\theta}$

② Once the distribution has been found, we can generate random samples from the distribution $= D'$

D' & D have the same distribution

∴ D & D' are "similar" (probabilistically)

③ Measure how similar D & D' are. We could use something like KL-Divergence, JS-dist etc, or we could do something like this (Model Based Similarity test)



if model is able to seperate well ⇒ D & D' are different

ie, model is able to distinguish b/w them. Vice-versa.

G → Generating data set

A → D & D' are "adversaries". They are compared with one another.

N → We use Deep Neural Networks instead of prob models generally.

→ Heights : Univariate, distribution is easy.

MNIST : Multivariate with 784 variables. Complex distributions. KL div and JS-dist won't scale as well. This is why models are preferred.

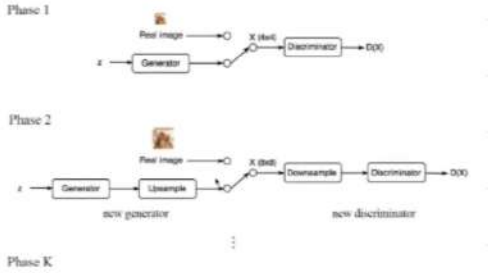
→ GANs are extremely hard to train and involve lots of hyperparameter tuning. There are hacks to make them work.

→ Pixel CNN, Pixel RNN :- Generating data based on each pixel. Hard to generate new samples from.

→ Variation Auto Encoders :- Do not work as well as GANs today.

→ There are task specific GANs.

Progressive GANs :- to create fake images

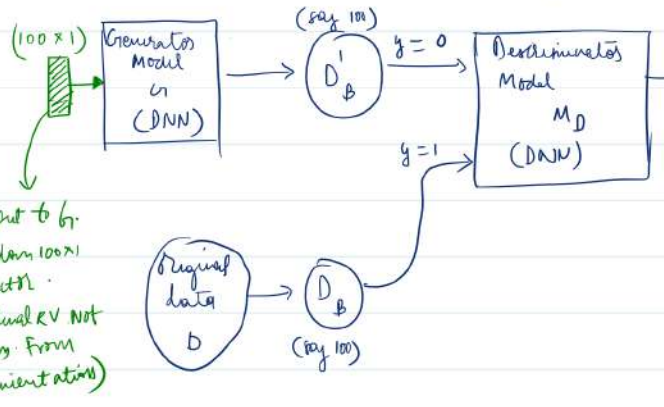


↓ slowly increase quality

→ SRGAN to deblur an image.

→ Most of the intuition behind GANs come from game theory.

→



→ maximize log loss (then $D' \simeq D$)

2 stages :-

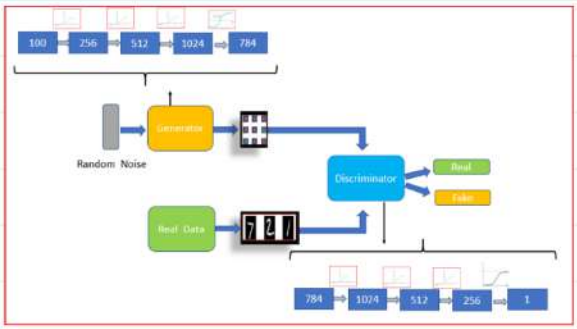
① Fix M_D & train G.

Initially i/p to G is random & the weights are also random. ⇒ image generated is also very random.

⇒ Model will discriminate easily.

② Fix G & train M_D. We do Gradient Ascent to maximize log loss. Error is sent back to G & weights are updated.

→ This is similar to SVM SMO - when 2 vars are to be minimized, we fix one, optimize the other & then fix that & optimize prev one. Can be done in Keras by using model_name.trainable = False. This freezes the model.



initially everything is random.

By the end, looks indistinguishable.

→ After lots of experimentation, [-1, 1] normalization aka tanh showed better results than [0, 1] norm aka sigmoid.

∴ tanh activation is preferred.

→ DCGAN (Deep Convolutional GAN) uses CNN's for generator & discriminator.

→ SGD for discriminator & Adam for generator best optimizers.

→ Use dropouts in G in both train & test phase.

Encoder Decoder Models :-

→ Sequence 2 Sequence was used as the algo of google translate for a while.

Gmail AutoReply, AutoSuggestions also use(d) Seq2Seq.

→ Original seq2seq model paper written by Ilya Sutskever.

* Machine Translation :-

$\langle x_1 \; x_2 \; x_3 \; \cdots \; x_t \rangle \xrightarrow{MT} \langle y_1 \; y_2 \; y_3 \; \cdots \; y_k \rangle$ t need not be equal to k.

sequences.

$\langle x_{11}, x_{12} \cdots \rangle$ $\langle y_{k1}, y_{k2} \cdots \rangle$

→ Image captioning

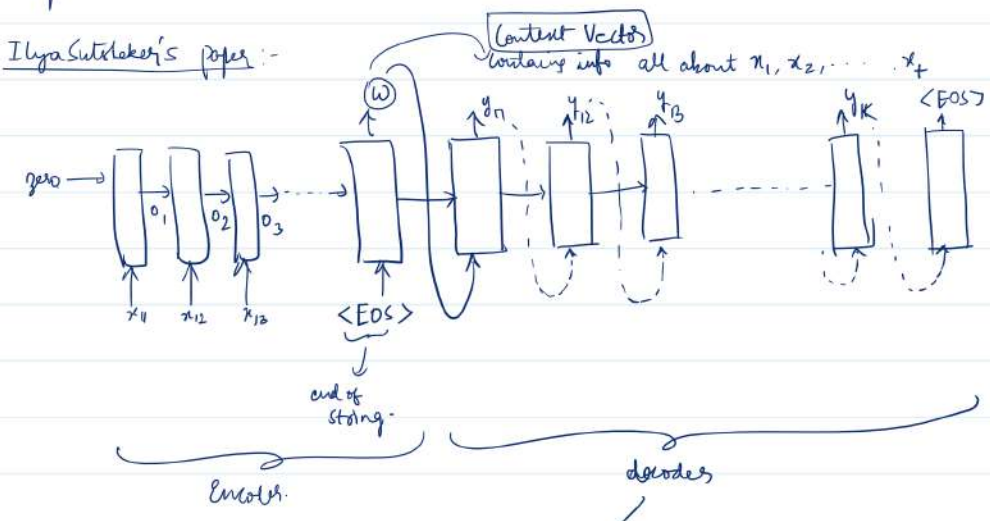
→ Mathematically, $P(\langle y^1, y^2, y^3 \ldots y^k \rangle | \langle x^1, x^2, x^3, \ldots x^t \rangle)$

lots of data $\xrightarrow{\text{Train}}$ Model $\xrightarrow{\text{Test}}$ $\langle \hat{y}_1, \hat{y}_2, \cdots \hat{y}_k \rangle$ $\hat{y} = \underset{y}{\operatorname{argmax}} \; P(y|x)$

(x_i, y_i) $\langle x_1, x_2, x_3 \cdots x_t \rangle$

↑↑ sequence

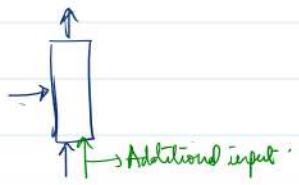
Ilya Sutskever's paper :-



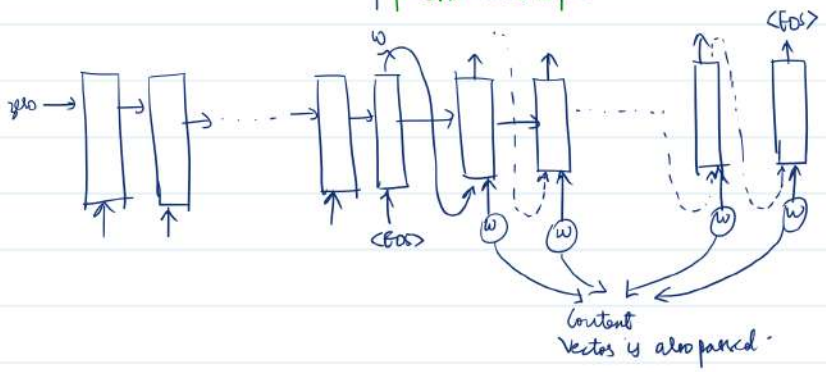
Generally LSTMs are used. Sometimes GRU's too.

Cho's paper :-

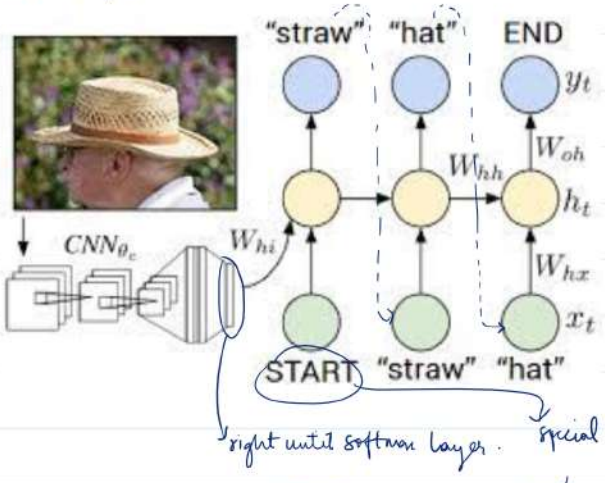
uses a modified LSTM cell



→ Since default LSTM is so well optimized & since this modified LSTM does not show massive improvement, not often used.



Karpathy's paper :-



right until softmax layer.

special START character/all zeros

encoder

decoder

→ I want food <EOS>

↑ ↑↑↑↑ ↑↑↑↑

characters level model, if each char is passed one by one after OHE

↑ ↑ ↑

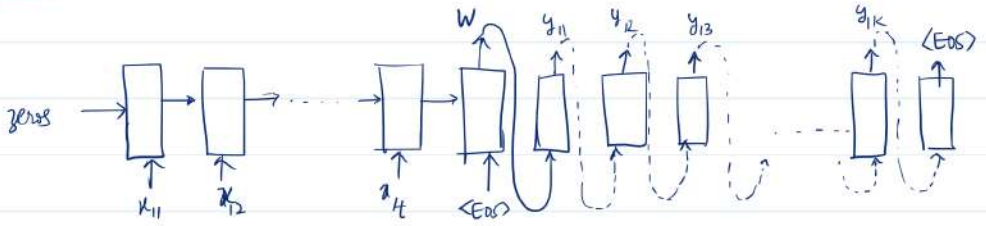
Word level model, if each word is passed one by one after OHE/W2V/. . .

→ Above models do not work well with long sentences and is not human like.

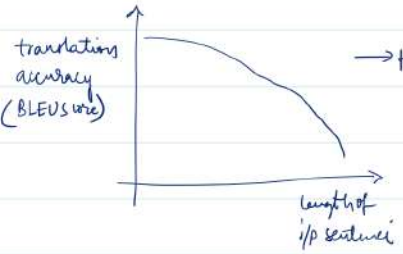
Humans do it by looking at whole input, generate some o/p, go back look at i/p again, then produce o/p.

This is the intuition behind attention models.

Attention Based Models :-

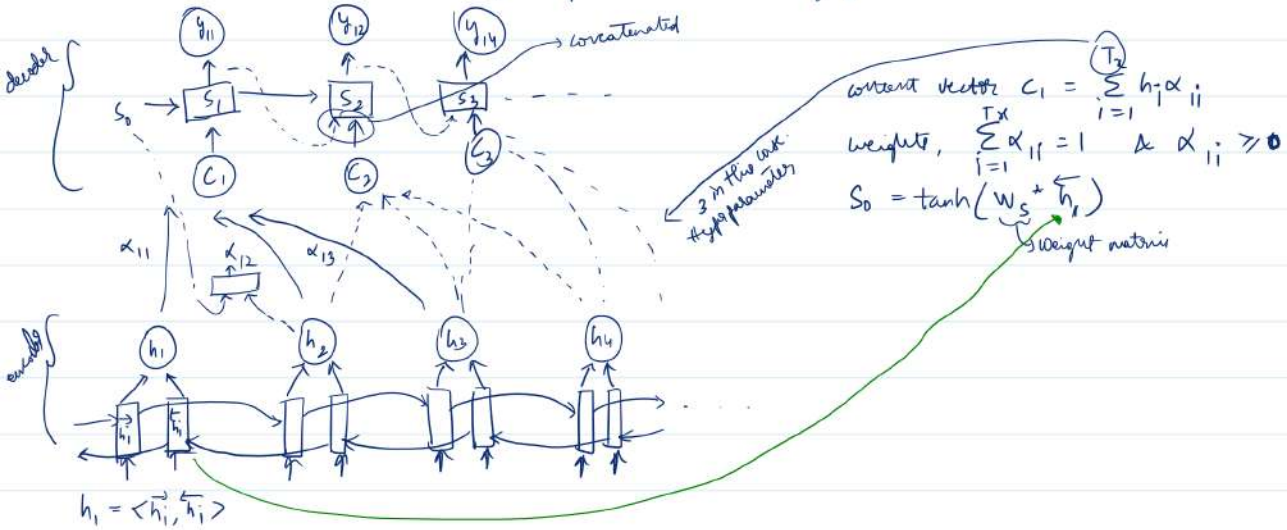


→ W on it's own is not able to capture the essence of the entire sentence if it's long



→ for simple encoder-decoder models.

→ We use bidirectional RNN because it might depend on words before/after a word in encoder. Decoder is unidirectional RNN



$h_1 = \langle \overrightarrow{h_1}, \overleftarrow{h_1} \rangle$

context vector $C_1 = \sum_{i=1}^{T_x} h_i \alpha_{1i}$

weights, $\sum_{i=1}^{T_x} \alpha_{1i} = 1$ & $\alpha_{1i} \geq 0$

$S_0 = \tanh(W_s * \overleftarrow{h_1})$

How to compute α_{ij}'s :-

$$\alpha_{ij} = \frac{\overbrace{\exp(e_{ij})}^{\text{always +ve/0.}}}{\underbrace{\sum_{k=1}^{T_x} \exp(e_{ik})}_{\text{denom will be same so sum = 1}}}$$

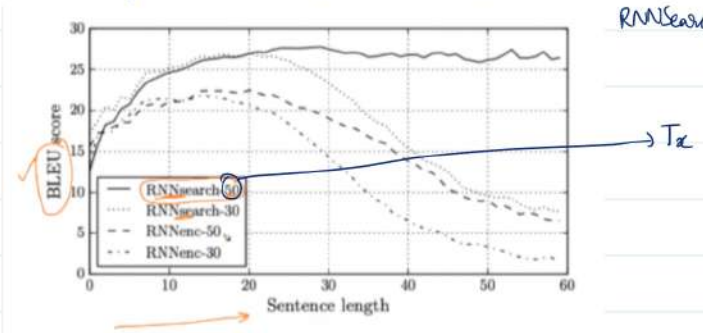
$$e_{ij} = a(S_{i-1}, h_j)$$

Attention model (how much attention needs to be given to encoder). We can use a simple feed forward NN for this.

ex:- $e_{12} = a(S_0, h_2)$ prev s & current h

→ Train using Backprop through time using adam & let it converge.

RNNSearch = Attention Model.

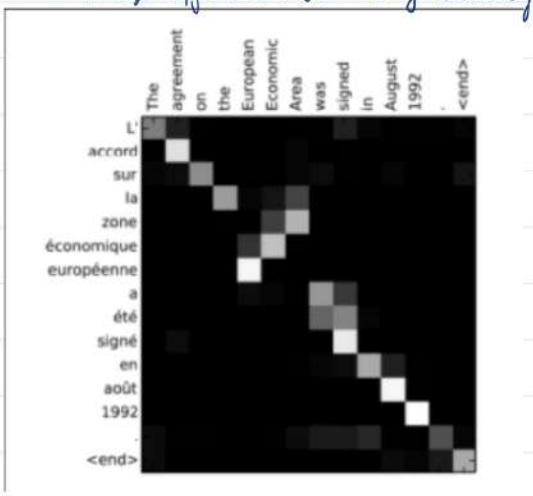


Drawback :- Time complexity $= O(k_1 \cdot k_2)$

k_1: lengths of i/p

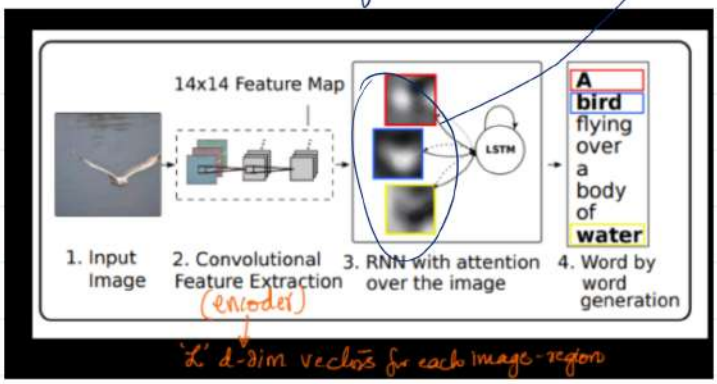
k_2: length of o/p.

→ We can visualize α_{ij}'s like this to debug. Accuracy Measurement can be done by using metrics like BLEU score.



→ Image-caption generation using Visual Attention :-

every region is basically like a word. if region is white ⇒ α of that region is large.



→

(Can also be used as localization)