MariaDB: …storage engines…

MyROCKS

- originally developed by Facebook (not mature in MariaDB)
- designed for read/write–heavy applications
- designed for data consistency (ACID)
  - transaction support
  - no automated foreign key support
- 2–4× better compression than InnoDB
- cannot be used with Galera Cluster

MariaDB: …storage engines…

ColumnStore

- designed for read–heavy applications
- data are distributed across multiple computers
  - 3+ nodes required (odd numbers only)
- data are highly compressed (65–95%)
- not designed for data consistency
  - no automated foreign key support

MariaDB: …storage engines

CONNECT

- allows MariaDB to read data from ‘other’ sources
  - best used as an import/export function
- can read from delimited (e.g. tab) text files
- can read from standard formats (e.g. JSON, XML)
- can read from other databases (e.g. MONGO)
- not designed for data consistency
  - no automated foreign key support

MariaDB: permissions

stored in the mysql database

- CREATE USER (ALTER USER)
  - specify authorization method
  - (usually) limit to 127.0.0.1 (localhost)
- GRANT
- basic: SELECT, INSERT, UPDATE, and DELETE
- ‘advanced’: CREATE and CREATE TEMPORARY TABLES
- FLUSH PRIVILEGES

MariaDB: CREATE TABLE

```
CREATE TABLE `Projects` (
  `ProjectID` MEDIUMINT UNSIGNED NOT NULL AUTO_INCREMENT,
  `Owner` SMALLINT UNSIGNED NOT NULL, -- non-transferable
  `Name` VARCHAR(255) NOT NULL,
  `Status` TINYINT UNSIGNED, -- 'inactive'/'active'
  `TimeStamp` TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`ProjectID`),
  FOREIGN KEY (`Owner`) REFERENCES Users (`UserID`),
  UNIQUE KEY `OwnerName` (`Owner`, `Name`)
) ENGINE=InnoDB DEFAULT CHARSET=UTF8;
```

MariaDB: SELECT

probably the most used of all keywords

extracts data and/or calculates results from data

SELECT [DISTINCT] field(s) [AS name]

FROM table(s)

WHERE field = | != | LIKE | NOT LIKE 'value'

    [can use REGEXP ‘value’ in place of a WHERE clause]

[ORDER BY field(s) [DESC | ASC]]

[LIMIT int]

MariaDB: SELECT string functions…

CONCAT

SELECT CONCAT(field1, field2, …) [AS name] …

LENGTH

SELECT LENGTH(field) [AS name] …

SELECT field WHERE LENGTH(field) = int

LOWER and UPPER

SELECT LOWER(field) …

SELECT UPPER(field) …

MariaDB: …SELECT string functions

TRIM

```
SELECT TRIM(field) [AS name] …
```

IF

```
SELECT IF(field = 'value', 'yes', 'no') [AS name] …
SELECT IF((field1 = 'value1' AND field2 = 'value2'), 'yes', 'no')
SELECT IF((field = 'value1' OR field = 'value2'), 'yes', 'no')
```

REPLACE

```
SELECT REPLACE(field, 'old', 'new') [AS name] …
```

MariaDB: SELECT date/time functions

DATE

SELECT DATE(field) [AS name] …

CURRENT_TIMESTAMP

SELECT CURRENT_TIMESTAMP() AS name

TIMESTAMPDIFF

SELECT TIMESTAMPDIFF(unit, field1,field2) [AS name] …

TO_DAYS

SELECT TO_DAYS(value) [AS name] …

MariaDB: SELECT math functions

ABS

SELECT ABS(field) [AS name] …

POWER

SELECT POWER(field, power) [AS name] …

ROUND

SELECT ROUND(field, decimals) [AS name] …

SQRT

SELECT SQRT(field) [AS name] …

MariaDB: SELECT aggregate…

COUNT

SELECT COUNT([DISTINCT] field) [AS name] …

SUM

SELECT SUM(field) [AS name] …

MAX, MIN, AVG, STD, STDDEV_SAMP, etc…

MariaDB: …SELECT aggregate

GROUP BY

for SELECT statements that have aggregate functions

SELECT … GROUP BY field …

GROUP BY … WITH ROLLUP for totals and subtotals

used to make cross tabs

SELECT genus, specificEpithet, COUNT(id) AS n FROM table GROUP BY genus, specificEpithet WITH ROLLUP;

MariaDB: nested SELECT

```
SELECT implicitTable.field, …
FROM (SELECT field …) AS implicitTable
WHERE …
```

MariaDB: INSERT

used to add rows

```
INSERT [IGNORE] INTO table
SET field = 'value'
[ON DUPLICATE KEY UPDATE field = 'value']
```

–or–

```
INSERT [IGNORE] INTO table
(field1, field2, …)
VALUES (value1, value2, …)
[ON DUPLICATE KEY UPDATE field = 'value']
```

MariaDB: UPDATE

modifies data

UPDATE [IGNORE] table

SET field = 'value'

[WHERE field = | != | LIKE | NOT LIKE 'value']

MariaDB: DELETE

removes data

DELETE [IGNORE] FROM table

[WHERE field = | != | LIKE | NOT LIKE 'value']

MariaDB: temporary tables

CREATE TEMPORARY TABLE …

- same syntax as CREATE TABLE
- or CREATE TEMPORARY TABLE table (SELECT ...)

held in RAM

- be careful not to overwhelm the server

non–persistent (dropped on logout)

visible to only one user/login

useful for speeding up queries, fixing joins, and data cleaning

botanists

botanist ID	abbreviation	born	died
1	Benth.	1800	1884
2	Brainerd	1844	1924
3	Britton	1859	1934
4	Fernald	1873	1950
5	A.Gray	1810	1888
6	Hultén	1894	1981
7	L.	1707	1778

names

name ID	botanist ID	given name(s)	family name
1	1	George	Bentham
2	2	Ezra	Brainerd
3	3	Nathaniel Lord	Britton
4	4	Merritt Lyndon	Fernald
5	5	Asa	Gray
6	6	Oskar Eric Gunnar	Hultén
7	7	Carl	Linnaeus
8	7	Carl von	Linné

interests

interest ID	interest
1	Algae
2	Bryophytes
3	Mycology
4	Pteridophytes
5	Spermatophytes

botanical interests

botanical interest ID	botanist ID	interest ID
1	1	3
2	1	4
3	1	5
4	2	5
5	3	2
6	3	3
7	3	4
8	3	5
9	4	4
10	4	5
11	5	1
12	5	2
13	5	4
14	5	5
15	6	2
16	6	4
17	6	5
18	7	1
19	7	2
20	7	3
21	7	4
22	7	5

MariaDB: JOIN

JOIN tables together

must have common fields

- common field names are not required…
- but common content and datatype are required

works best (fastest) if fields are INT, SET, or ENUM

- indexing can provide additional speed
- slow on text fields (errors can occur with LIKE etc.)

can be used for SELECT, INSERT, DELETE, or UPDATE

MariaDB: ‘implicit’ joins

i.e. joins that do not use the ‘JOIN’ keyword

SELECT table1.field, table2.field, table3.field FROM table1, table2, table3;

common field names must be the same

or use FOREIGN KEY with InnoDB*

can have only one join order

SELECT table1.field, table2.field, table3.field FROM table1, table2, table3
WHERE table1.id = table2.id AND table2.id = table3.id;

common field names do not need to be the same

can have multiple join orders

MariaDB: NATURAL JOIN

same as an ‘implicit’ join

SELECT table1.field, table2.field, table3.field FROM table1 NATURAL JOIN table2 NATURAL JOIN table3;

common field names must be the same

or use FOREIGN KEY with InnoDB*

can have only one join order

MariaDB: INNER JOIN

same as a CROSS JOIN

same as an ‘implicit’ or NATURAL JOIN if all fields with common names are used in the ON statement

- each row must match (i.e. no NULL values are created)

SELECT table1.field, table2.field, table3.field FROM table1 INNER JOIN table2 INNER JOIN table3 ON (table1.id = table2.id AND table2.id = table3.id);

- common field names do not need to be the same
- can have multiple join orders

MariaDB: [LEFT | RIGHT] JOIN

LEFT contains all rows from the left (first) table plus data or NULL from the right table(s)

RIGHT contains all rows from the right (second) table plus data or NULL from the left table(s)

useful for finding (or suppressing) missing data

SELECT table1.field, table2.field, table3.field FROM table1 LEFT JOIN (table2 NATURAL JOIN table3) ON (table1.id = table2.id AND table2.id=table3.id);

MariaDB: JOIN + AS

to join a table to itself use an alias

SELECT table1.field1, table1.field2 FROM table1, table1 AS table2 WHERE table1.idx = table2.idy;

useful for hierarchical queries or multiple join orders

MariaDB: UNION

concatenate multiple SELECT results

(SELECT …) UNION (SELECT …)

MariaDB: complex (strange) results

always (manually) check your results

multiple join paths are often sources of problems

use AS and/or ‘implicit’ join to force particular paths

    being explicit will usually fix things

use UNION to combine partial joins

use temporary tables to build and check partial joins

    then join the temporary tables

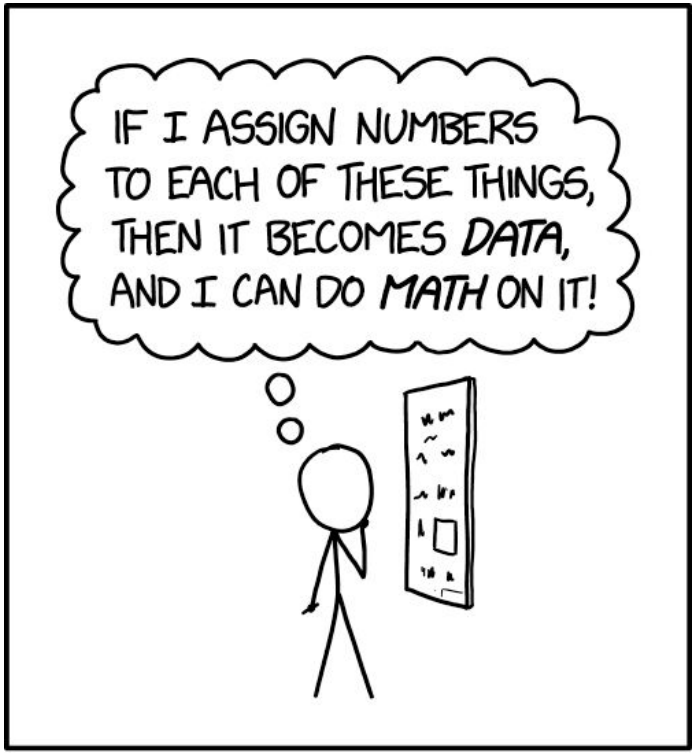
MariaDB: even more SQL

FUNCTION

- used to automate data handling (e.g. formatting)
- can be used to run ‘external’ code

TRIGGER

- used to automate actions based on changes in data
- can be used to ensure data integrity
- can be used for automatic exports



THE SAME BASIC IDEA UNDERLIES GÖDEL'S INCOMPLETENESS THEOREM AND ALL BAD DATA SCIENCE.

https://xkcd.com/2610

https://commons.w

AI versus ML

Artificial Intelligence (AI)
- broader term

algorithms are not explicitly programmed

predictions based on data
- ‘unsupervised’ learning

does not exist (yet)

an aspiration to make machines more human like
- synthetic thinking,
- cross–apply knowledge,
- creativity, emotions
- usually ends badly in science fiction

Machine Learning (ML)
- narrower term

the current ‘AI’ technology

predictions based on data
- ‘unsupervised’ learning
- ‘supervised’ learning

a wide variety of techniques

can be amazingly effective for ‘pattern recognition’ tasks

frequently a victim of the Dunning–Kruger effect (greatly overestimates its knowledge and abilities)

ML: research goals

understand the input data

- model parameters/hyperparameters ‘explain’ data
- model predictions only used to evaluate model quality

predict output from novel data

- understand the novel data and/or a make a prediction tool
- model parameters/hyperparameters are (often) ignored
  - cannot be (easily) interpreted for some methods
- quality of the model predictions are heavily optimized

ML: types of input

quantities: floating point numbers

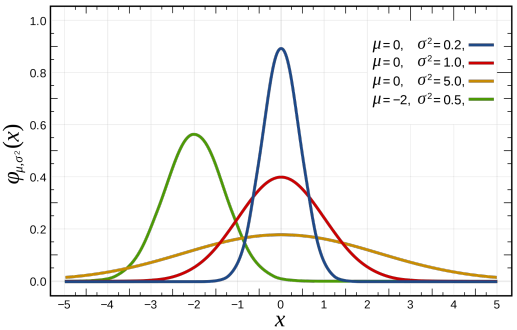
- typically real measurements

encoded representations: floating point numbers

- images, categorical data, sequence (text) data, etc.

(usually) data should be normally (Gaussian) distributed

- may need to transform data into normal distribution
- may need to zero–center data
- may need to scale data (*z*-score, log, ln, 0–1, etc.)



https://commons.wikimedia.org/wiki/File:Normal_Distribution_PDF.svg

ML: encoded input representations…

images are just arrays of integers

- (usually) three integers per pixel
- convert to arrays of zero–centered, *z*-score standardized floating point numbers

categorical data are coded as an array of floating point numbers

- 0.0/1.0
- one–hot: memory inefficient, not easily updatable
- binary: memory efficient, not easily updatable
- *n*–bit hash: memory efficient, (may be) easily updatable

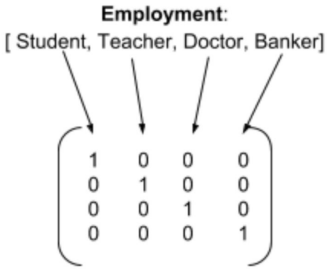


Figure 2.2: Graphical representation of one-hot encoding

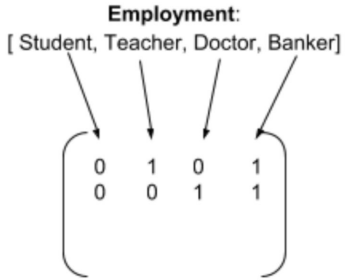


Figure 2.3: Graphical representation of binary encoding

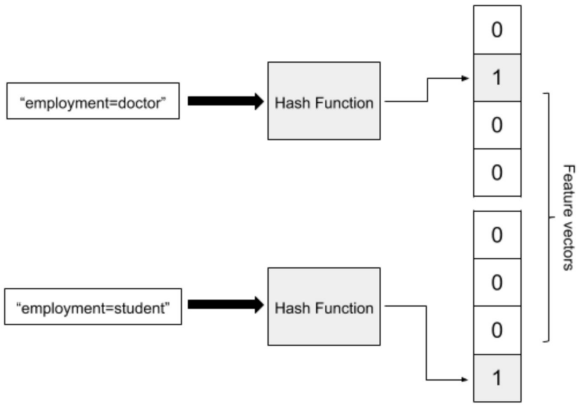


Figure 2.4: Graphical representation of feature hashing

(Seger 2018; http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-237426)

Table 4.1: Performance on Census Data

(a) Linear Model Performance

Input Type	*PR-AUC*	*Precision*	*Recall*	*Dimension*
One hot	0.730	0.72	0.58	118
Binary	0.664	0.70	0.51	35
Feature Hashing	0.600	0.66	0.33	35

(b) Non-Linear Model Performance

Input Type	*PR-AUC*	*(+/-) Std*[a]	*Precision*	*Recall*
One hot	0.728	0.002	0.72	0.57
Binary	0.714	0.006	0.71	0.57
Feature Hashing	0.691	0.006	0.70	0.53

[a]The standard deviation is based on 10 re-runs of training and evaluating each model. The PR-AUC reported is the mean of these 10 runs.

(Seger 2018; http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-237426)

ML: …encoded input representations…

sequences are coded as an array of floating point numbers

- (too) many proposed encodings (ca. 20+)
  - optimal encoding(s) is unknown
- reduced amino acid alphabets are often used
- one–hot encoding (amino acids or nucleotides)
- kmer frequency (amino acids or nucleotides)
  - discards sequence order information
  - e.g. amino acids: k = 2, 3; nucleotides k = 6, 8
  - reduced kmers

ML: …encoded input representations

sequences are coded as an array of floating point numbers

- embeddings (amino acids or nucleotides)
  - learned (or precomputed) clustering of tokens
    - variation partitioned into reduced–conflict hyperplanes
  - tokens can be single letters, byte pairs, kmers, etc.
- multivariate (amino acids)
  - physicochemical properties
  - substitution matrix (e.g. BLOSUMx, PSSM)
  - alignment to a reference database (MSA or pairwise)

Table 2. Five factor solution scores for the 54 selected amino acid attributes

Amino acid	Factor I	Factor II	Factor III	Factor IV	Factor V
A	−0.591	−1.302	−0.733	1.570	−0.146
C	−1.343	0.465	−0.862	−1.020	−0.255
D	1.050	0.302	−3.656	−0.259	−3.242
E	1.357	−1.453	1.477	0.113	−0.837
F	−1.006	−0.590	1.891	−0.397	0.412
G	−0.384	1.652	1.330	1.045	2.064
H	0.336	−0.417	−1.673	−1.474	−0.078
I	−1.239	−0.547	2.131	0.393	0.816
K	1.831	−0.561	0.533	−0.277	1.648
L	−1.019	−0.987	−1.505	1.266	−0.912
M	−0.663	−1.524	2.219	−1.005	1.212
N	0.945	0.828	1.299	−0.169	0.933
P	0.189	2.081	−1.628	0.421	−1.392
Q	0.931	−0.179	−3.005	−0.503	−1.853
R	1.538	−0.055	1.502	0.440	2.897
S	−0.228	1.399	−4.760	0.670	−2.647
T	−0.032	0.326	2.213	0.908	1.313
V	−1.337	−0.279	−0.544	1.242	−1.262
W	−0.595	0.009	0.672	−2.128	−0.184
Y	0.260	0.830	3.097	−0.838	1.512

```
F1  1.357 -1.019 -1.337 -1.239 -0.228
F2 -1.453 -0.987 -0.279 -0.547  1.399
F3  1.477 -1.505 -0.544  2.131 -4.760
F4  0.113  1.266  1.242  0.393  0.670
F5 -0.837 -0.912 -1.262  0.816 -2.647
    E      L      V      I      S
```

(Atchley et al. 2005; https://doi.org/10.1073/pnas.0408677102)

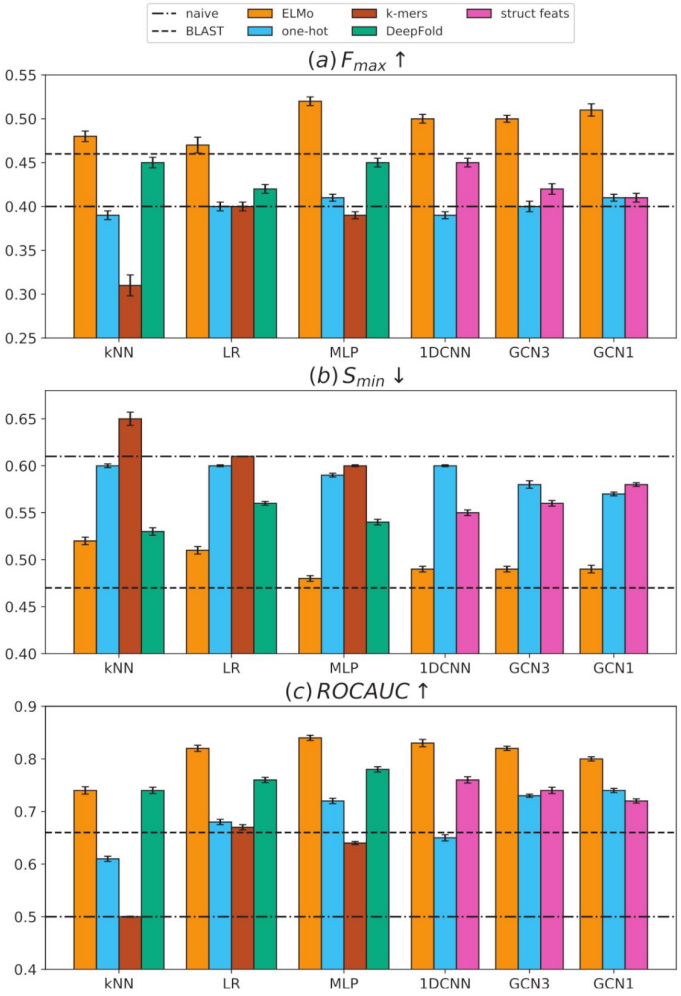


Fig. 2. $F_{\max}$ (**a**), $S_{\min}$ (**b**) and ROCAUC (**c**) of models trained using either ELMo embeddings (orange), one-hot encodings (blue), *k*-mer counts (brown), DeepFold (green) or structural features (pink), averaged over the cross-validated 30% sequence identity *PDB* test subsets. The arrows denote that lower values (in $S_{\min}$) and higher values (in $F_{\max}$ and ROCAUC) correspond to better performance. The error bars denote the standard deviation of the cross-validated results. The dashed line corresponds to the performance of BLAST and the dashed dotted line to the naive baseline

(Villegas-Morcillo et al. 2021; https://doi.org/10.1093/bioinformatics/btaa701)

ML: types of output

quantities: ‘regression’

predicts one or more meaningful numbers from input

symbols (numeric): ‘classification’

categorizes the input into one or more classes

easier for mutually exclusive classes

(sometimes) easier for hierarchical classes

representations (numeric): ‘translation’

transforms input into something different with the same encoding

ML: types of data

labeled: input + output data

unlabeled: input + algorithm to calculate output

train: (un)labeled data used to build the ML model

may be memorized (overfitting)

validation: labeled data used to evaluate the ML model

cannot be memorized, but may bias hyperparameters

test: labeled data used for evaluation the ML model

an unbiased estimate of model performance

cannot be memorized or bias hyperparameters

ML: evaluating model predictions

regression: difference between predicted and ‘true’

e.g. r^2, Root Mean–Square Error (RMSE), etc.

classification: correct or incorrect

measured by counting true|false negatives|positives

e.g. precision, recall, accuracy, F_1, AUPRC, etc.

translation: similarly between predicted and model of ‘true’

scores are very task specific