machine learning zoomcamp

1.1 Introduction to machine learning

'FEATURES'

→ age (year)
→ make (BMW..)
→ mileage

'TARGET'

→ price of the car

MODEL TRAINING

Features → ML → Model
Target →

DATA → Experts → PATTERNS

DATA → ML MODEL → PATTERNS

ML → process of extracting patterns from data.

PREDICTIONS

Features + Model → Predictions

1.2 ML vs Rule-Based Systems

Email System

→ SPAM detections

RULES

→ sender = 'promotions@online.com'
→ title contains = tax review and sender domain is 'online.com'

```
def detect_spam(email):
    if email.sender == 'promotions@online.com':
        return SPAM
    if contains(email.title, ['tax', 'review']) and
            domain(email.sender, 'online.com'):
        return SPAM
    return GOOD
```

PROBLEMS

→ other kind of SPAM messages.
↳ add new rules

→ if the Email Body contains a word 'deposit'
↳ if sender domain = 'test.com' then SPAM
↳ if body >= 100 words then SPAM

add new rules → feedback → REPEAT → some other spam email

SOLUTION → use machine Learning

1) Get Data → Button to mark emails as 'SPAM' or 'NOT SPAM'
2) Define and calculate features → email title (>=10)? body (>=10)? length, particular sender? , specific domain, description contains some words like 'deposit'! ['test.com' RULES AS FEATURES]
3) Train and use the model

['COME FROM THE' RULES]

Train
features + target → train an ML model
process of training as 'fitting'

Apply
ML model + Features (data) → predictions (output): 0.8, 0.6, 0.1 → Final outcome (decision)
≥0.5 SPAM

Features (data)	Target (desired output)
[1, 1, 0, 0, 1, 1]	1
[0, 0, 0, 1, 0, 1]	0
[1, 1, 1, 0, 1, 0]	1
[1, 0, 0, 0, 0, 1]	1

based on the features

Rule Based Systems

Data →
Code → software → outcome

☆ difficult to maintain in the long term

machine learning approach

Training
Data →
outcome → (SPAM / NOT SPAM)
ML → model

predictions
Data + model ⇒ outcome

1.3 Supervised machine learning

Features (data) features ↓	Target (desired output)
→ [1, 1, 0, 0, 1, 1]	1
[0, 0, 0, 1, 0, 1]	0
[1, 1, 1, 0, 1, 0]	1
[1, 0, 0, 0, 0, 1]	1
observations	vector y

feature matrix X

$g(X) \approx y$

X: feature matrix
g: model
y: target variable

the goal of machine learning is to be able to train to come up to this function g that takes in the feature matrix and produces output that is as close as possible to the target variable.

Types of supervised machine learning

1) Regression $g(X) \rightarrow -\infty$ to $+\infty$
   - ↳ car price, house price
   - [output is a number]
2) classification
   - [output is a category]
   - ↳ email spam, identify objects in picture

   sub categories
   - → multiclass [car / cat / dog / ...]
   - → binary [spam / not spam] $g(X) \rightarrow$ probability 0 to 1
3) Ranking
   - ↳ recommending systems

Supervised machine learning is about teaching computer / machine an algorithm by showing different examples (X feature matrix) and the output predictions (y target variable).

Goal is to come up with a function g such that $g(X) \approx y$.

1.4 CRISP-DM ML Process

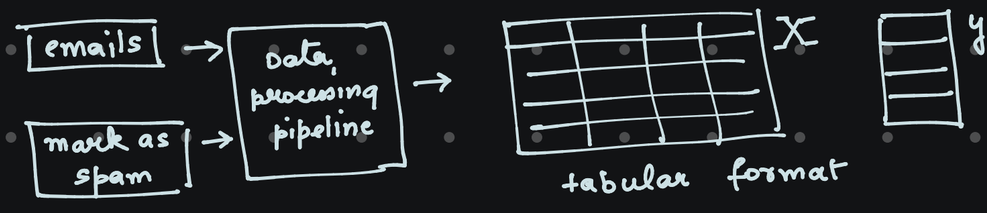
CRISP-DM : cross industry standard processing data mining
↳ methodology for organizing ML projects.

ML Projects
- → understand the problem
- → collect the data
- → Train the model
- → use it

1) Business understanding
   - → identify the business problem
     - → understand how we can solve it
   - → how would we measure the success of our project? reduce the amount of spam by 50%.
   - ⇒ Do we actually need ML?

2) Data understanding
   - → analyze available data sources
     - → decide if we need to get more data?
   - → Identify the data sources
     - ↳ it may influence the goal
     - ↳ we may go back to the previous step and adjust it.

3) Data Preparation
   - → Transform the data so it can be put into a ML algorithm.
     - ↳ extracting different features
   - → clean the data → Build the pipeline → convert into tabular format.



4) Modeling
   - → Train the models:
     the actual machine learning happens here
     - ↳ Try different models [logestic regression, Decission tree, neural network ...]
     - ↳ select the best one
   - → Sometimes, we may go back to data preparation:
     - → add new features
     - → fix data issues

5) Evaluation
   - → measure how well the model solves the business problem.

   Is the model good enough?
   - have we reached the goal?
   - Do our metrics improve?

   Do a retrospective:
   - was the goal achievable?
   - Did we solve / measure the right thing?

   After that we may decide to:
   - Go back and adjust the goal
   - Roll the model to more users / all users
   - stop working on the project

6) Evaluation + Deployment

   often happens together:
   - online evaluation: 5% users evaluation of live users.
   - it means: deploy the model, evaluate it.

Deployment
- → deploy the model to production.
  - ↳ roll the model to all users.
  - ↳ Proper monitoring
  - ↳ Ensuring the quality and maintainability.

* ML Projects may require many iterations!

1.5 Model Selection Process

selecting the best model

X y g	August
July	g (✉) = 0.7

data

80%	20% [validation data set]
Training data	Test data

① split the data
② Train the model (60% data)
③ validate the model (20% data)
④ select the best model
⑤ Test the model (20% data)

TRAIN → $X\ y \rightarrow g$, $g(X) \approx y$

VAL → $X_v\ y_v$, $g(X_v) = \hat{y}_v$, $\hat{y}_v \approx y_v$

$\hat{y}_v$	y_v	
0.8 1	1 ✓	total = 4
0.7 1	0 ×	correct = 2
0.1 0	0 ×	$\frac{2}{4} = 0.5$
0.6 1	1 ✓	50% correct model

multiple comparison problem

In statistics when we perform the same comparison many times so when we use many different models and evaluate them against the same validation data set one of the models can get particularly lucky (just by chance)

Train +	Valid +	Test
60%	20%	20%
$X\ y \rightarrow g$	$X_v\ y_v \rightarrow$	$X_T\ y_T$

1.7 Introduction to NumPy

```
import numpy as np
```

creating arrays → `np.zeros(5)` → array([0,0,0,0,0])

```
np.ones(5)
array([1,1,1,1,1])
np.full(5, 2.5)
array([2.5, 2.5, 2.5, 2.5, 2.5])
a = np.array([1,2,3,4])
a   array([1,2,3,4])
a[3]   4
a[2] = 10
a   [1, 10, 3, 4]
np.arange(10)   [0,1,2,3,4,5,6,7,8,9]
np.arange(3,10)
np.linspace(0,1,11)
```

multi-dementional arrays

(row, column)

```
np.zeros((5,2))
n = np.array([[1,2,3],
              [4,5,6],
              [7,8,9]])
n[0,1]   2
n[2]
array([7,8,9])
n[:,1]
array([2,5,8])
```

Randomly_generated arrays

```
np.random.rand(5,2)
np.random.seed(2) → get same random numbers.
np.random.randn(5,2) → normal distribution
np.random.randint(low=0, high=100, size=(5,2))
```

Element-wise operations

```
a = np.arange[5]
array([0,1,2,3,4])
a * 2
array([0,2,4,6,8])
(10 + (a*2)) ** 2 / 100
```

comparison operations

```
a   array([0,1,2,3,4])
a >= 2
array([False, False, True, True, True])
b   array([1,0,3,4,5])
a > b
array([False, True, False, False, False])
a[a>b] → returns all the elements of array a for which the condition [a>b] is True.
array([1])
```

a.min() a.max() a.sum() a.mean() a.std()

1.8 Linear Algebra Refresher

Vector operations

$$2 \times \begin{bmatrix}2\\4\\5\\6\end{bmatrix} = \begin{bmatrix}4\\8\\10\\12\end{bmatrix} \qquad \begin{bmatrix}2\\4\\5\\6\end{bmatrix} + \begin{bmatrix}1\\0\\0\\2\end{bmatrix} = \begin{bmatrix}3\\4\\5\\8\end{bmatrix}$$

(u + v)

column vector

multiplication

- v × v
- m × v
- m × m

vector × vector [dot product]

$$\sum_{i=1}^{n} u_i v_i$$

$$\begin{bmatrix}2\\4\\5\\6\end{bmatrix} \cdot \begin{bmatrix}1\\0\\0\\2\end{bmatrix} = 2\cdot1 + 4\cdot0 + 5\cdot0 + 6\cdot2 = 2+0+0+12 = 14$$

turn a row vetor to column vetor we use transpose.

v^T [2 4 5 6] ← row vetor

Column vector ← u $\begin{bmatrix}1\\0\\0\\2\end{bmatrix}$

`u.dot(v)`

```
def vector_vector_mul(u,v):
    assert u.shape[0] == v.shape[0]
    n = v.shape[0]
    result = 0.0
    for i in range(n):
        result = result + u[i] * v[i]
    return result
```

matrix × matrix

$$U \begin{bmatrix}2&4&5&6\\1&2&1&2\\3&1&2&1\end{bmatrix} \quad V \begin{bmatrix}1&1&2\\0&0.5&1\\0&2&1\\2&1&0\end{bmatrix} = \begin{bmatrix}|&|&|\\UV_1&UV_2&UV_3\\|&|&|\end{bmatrix}$$

matrix × vector

$$\begin{bmatrix}2&4&5&6\\1&2&1&2\\3&1&2&1\end{bmatrix} \begin{bmatrix}1\\0.5\\2\\1\end{bmatrix} = \begin{bmatrix}u_0^T v\\u_1^T v\\u_2^T v\end{bmatrix}$$

row of matrix × vector

```
def matrix_vector_mul(U,v):
    assert U.shape[1] == v.shape[0]   (column, rows)
    num_rows = U.shape[0]
    result = np.zeros(num_rows)
    for i in range(num_rows):
        result[i] = vector_vector_mul(U[i], v)
    return result
```

`U.dot(v)`

```
def matrix_matrix_mul(U,V):
    assert U.shape[1] == V.shape[0]
    num_rows = U.shape[0]
    num_column = V.shape[1]
    result = np.zeros((num_rows, num_column))
    for i in range(num_column):
        vi = U[:,i]
        Uvi = matrix_vector_mul(U, vi)
        result[:,i] = Uvi
    return result
```

`U.dot(V)`

IDENTITY MATRIX

$$I = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$$ `np.eye(3)`

- → square matrix (#row = #column)
- → all diagonals have 1

if any matrix $U \cdot I = U$

MATRIX INVERSE

A matrix A^{-1} inverse of matrix A

A^{-1} is a matrix such that when we multiply it by A we get an Identity matrix I.

$$A A^{-1} = I$$

```
Vs = np.linalg.inv(V)   (matrix, linear algebra, inverse)
Vs.dot(V)
```

we will get Identity matrix

1.9 Introduction to Pandas

```
import numpy as np
import pandas as pd

df = pd.DataFrame(data, columns=columns)
```
row data, list of columns

data can also be a list of dictionaries.
(Keys → column name)

```
df.head()  → First 5 Rows
df.head(n=2)

df.Make   ← column name
df['Make']

df[['Make', 'model', 'MSRP']]
df['id'] = [1, 2, 3, 4, 5]
```

to delete a column
```
del df['id']
```

range index
```
df.index
df.Make.index
df.loc[1]
df.loc[[1, 2]]
```
index

returns a new df
```
df.reset_index(drop=True)
df.reset_index()
```
delete the column with previous indices.
create an index column to store the previous indices

```
df.index = ['a', 'b', 'c', 'd', 'e']
```

positional index
```
df.iloc[1]
df.iloc[[1,2,4]]
```

```
df['Engine HP'] / 100
df['year'] >= 2015
df[
    df['year'] >= 2015
]
df[
  (df['year'] >= 2015) &
  (df['Make'] == 'Nissan')
]
```

```
df.isnull()
df.isnull().sum()
```

```
df['vechicle_Style'].str.lower()
df['vechicle_style'].str.replace(' ', '_')
```

```
df.MSRP.max()
df.MSRP.min()
df.MSRP.mean()
df.MSRP.describe()
df.describe().round(2)
```

```
df.Make.nunique()
df.nunique()
```

```
df.groupby('Transmission_type')
  .MSRP.mean()
```

```
SELECT
    transmission_type,
    AVG(MSRP)
FROM
    cars
GROUP BY
    transmission_type
```

```
df.MSRP.values
df.to_dict(orient='records')
```