session 3

how to ensure models aren't lying to us

ml.school

1 evaluation strategies

2 slicing out the test data

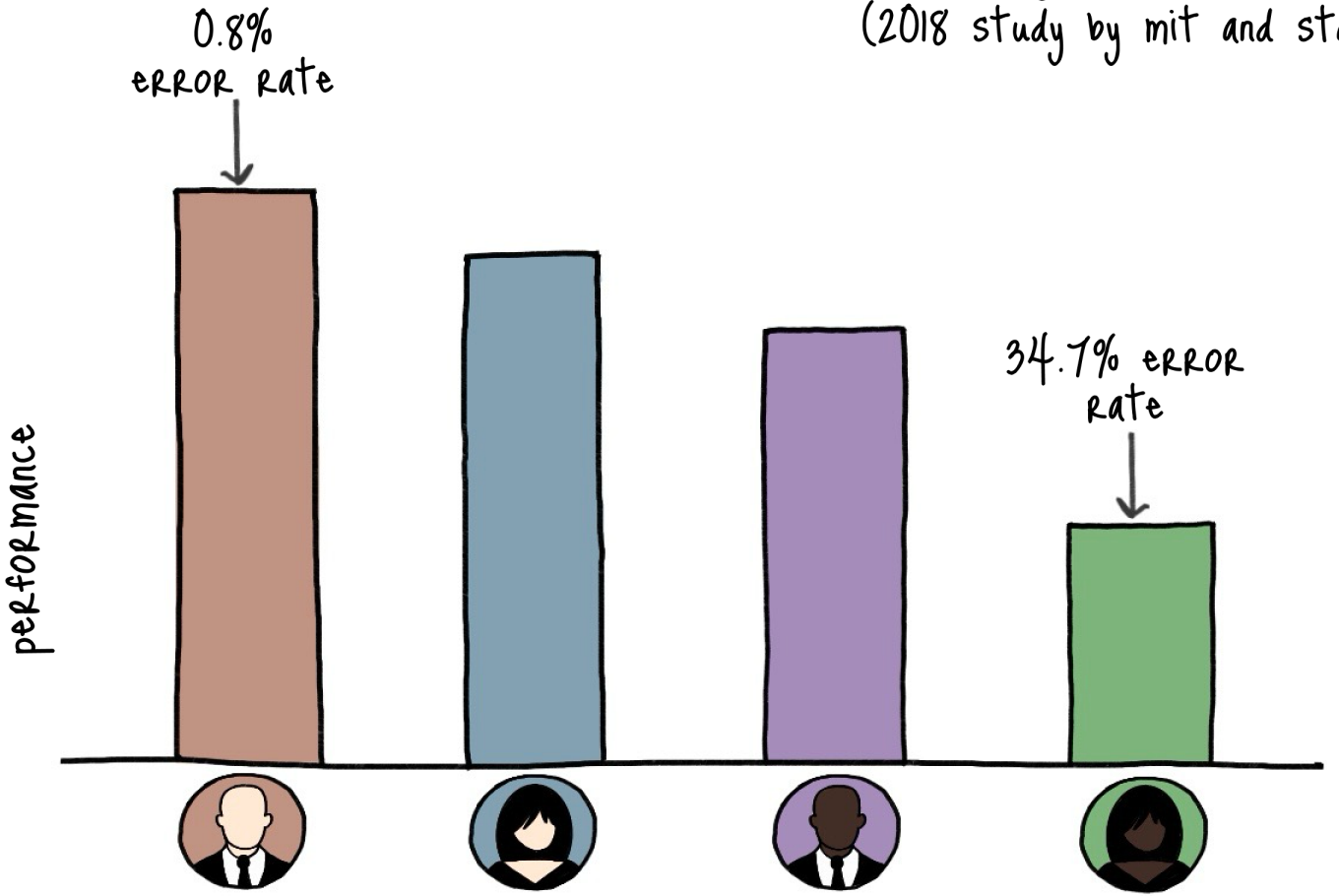
3 error analysis

4 preventing data leakages

5 working with imbalanced data

performance of commercial face recognition services

(2018 study by mit and stanford)



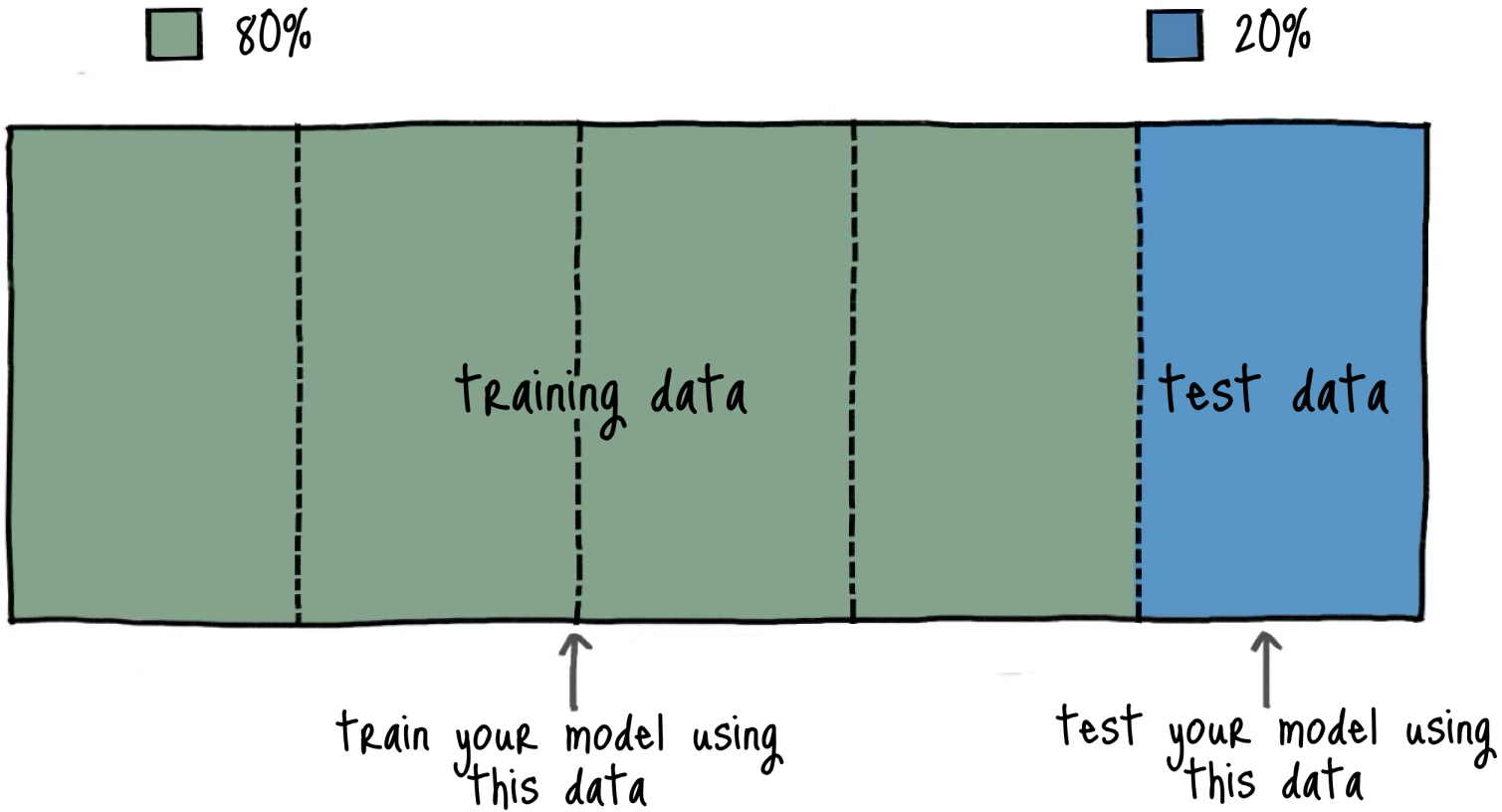
Spend **more time** evaluating your models.
If your model hasn't been thoroughly tested,
consider it **broken**.

Model evaluation starts with a **baseline.**

Use a **random** baseline, the **zero-rule** algorithm, or compare against **existing solutions** or **people.**

A **holdout set** is the simplest evaluation strategy. Sample a **small portion** of the data and use it to **infer** the model performance on **future, unseen** data.

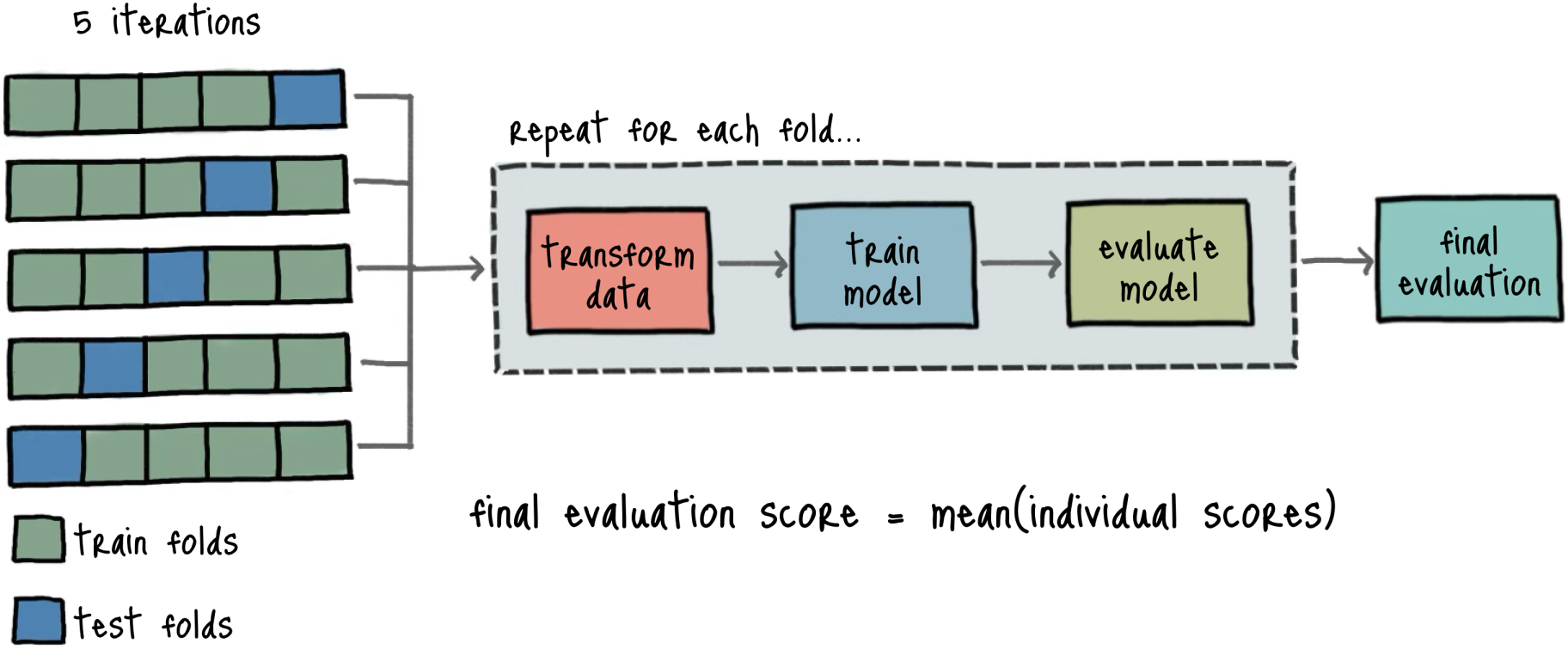
using a holdout set



A **holdout set** may lead to a **biased** model assessment. It's also an **inefficient** way to take advantage of the data, especially when working with **small** datasets.

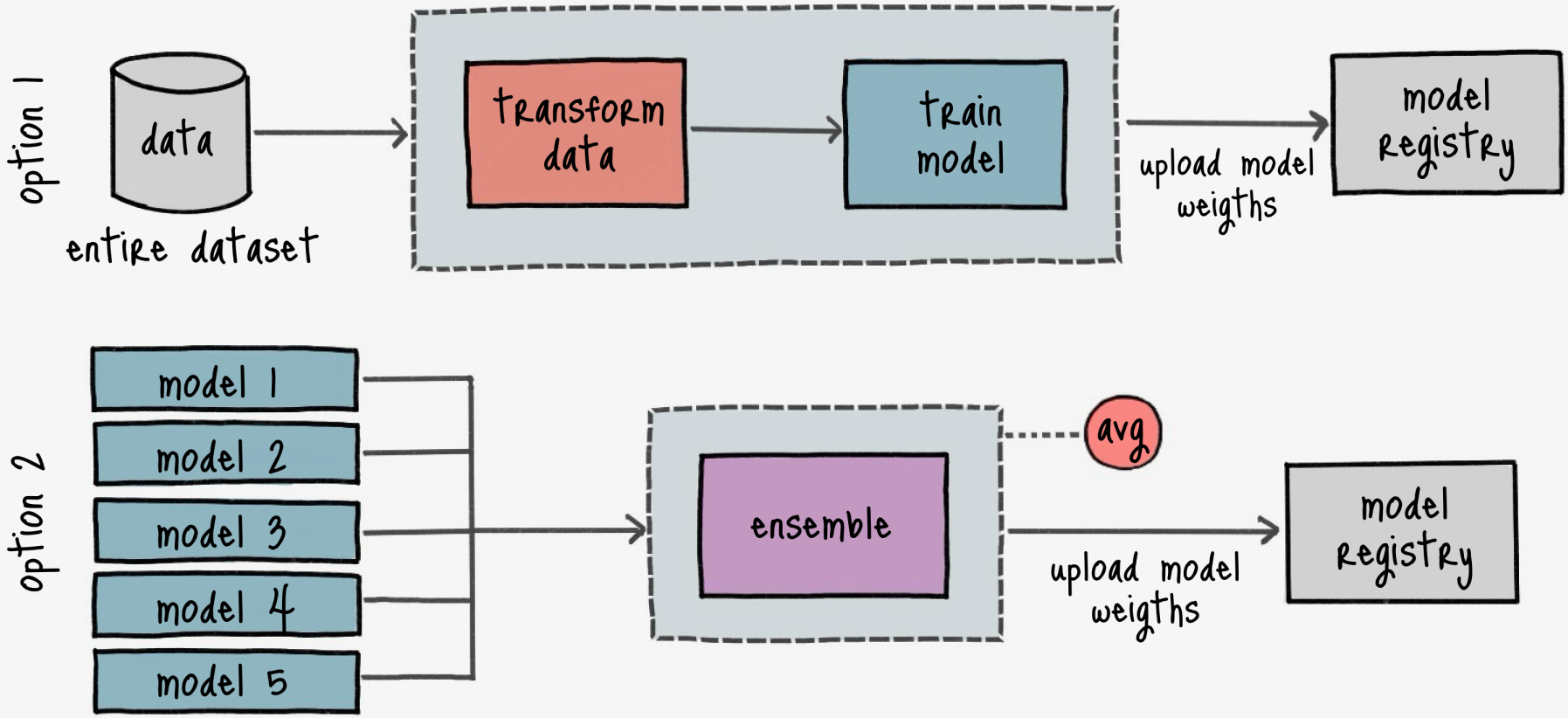
Cross-validation provides a more reliable model assessment. It produces **less biased** estimates of model performance and it’s ideal for **small** datasets.

5-fold cross-validation



final evaluation score = mean(individual scores)

final model after cross-validation



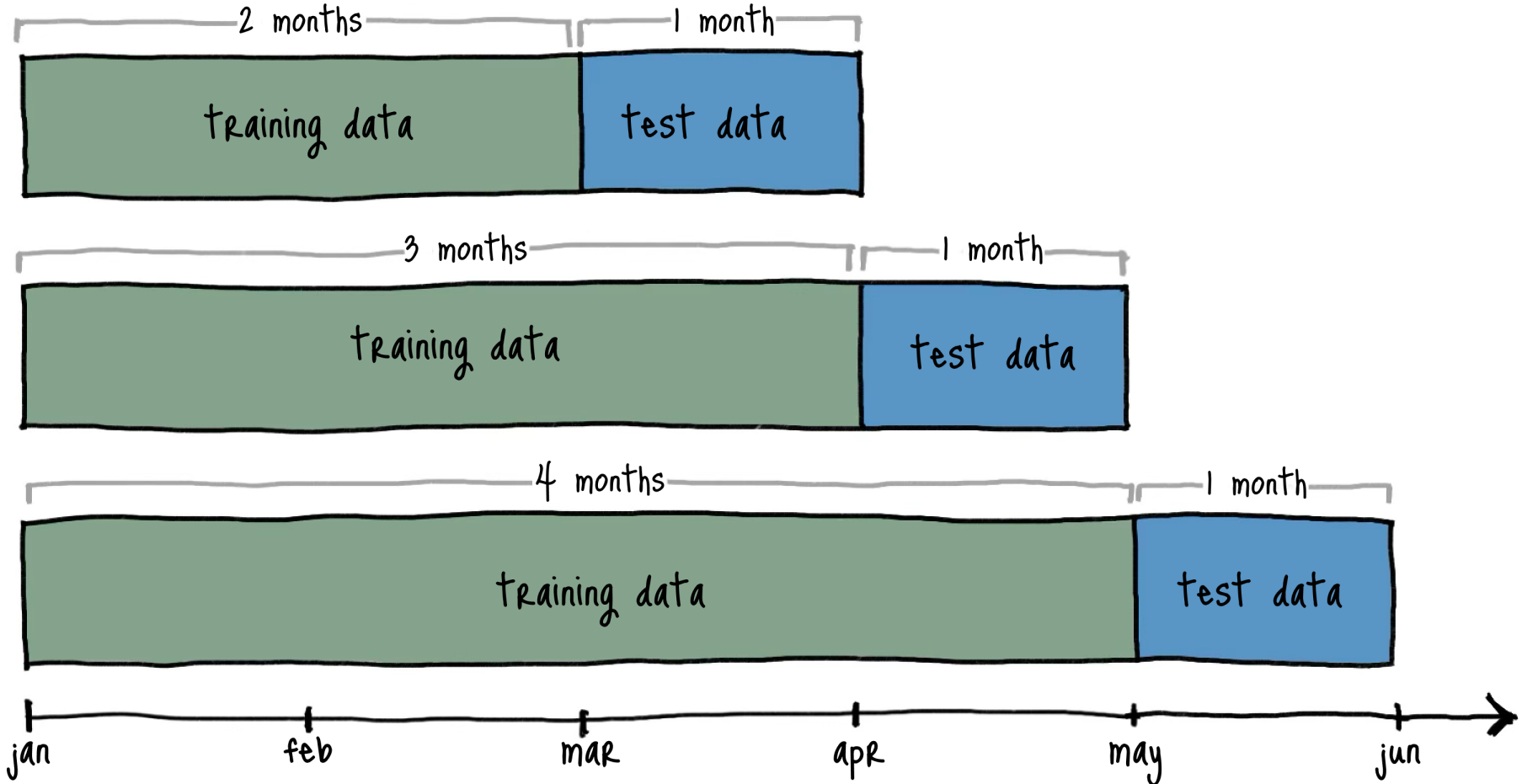
Cross-validation is prohibitive for **large** datasets.
It is too **costly** because it involves **multiple rounds** of training and validation of multiple models.

best practice

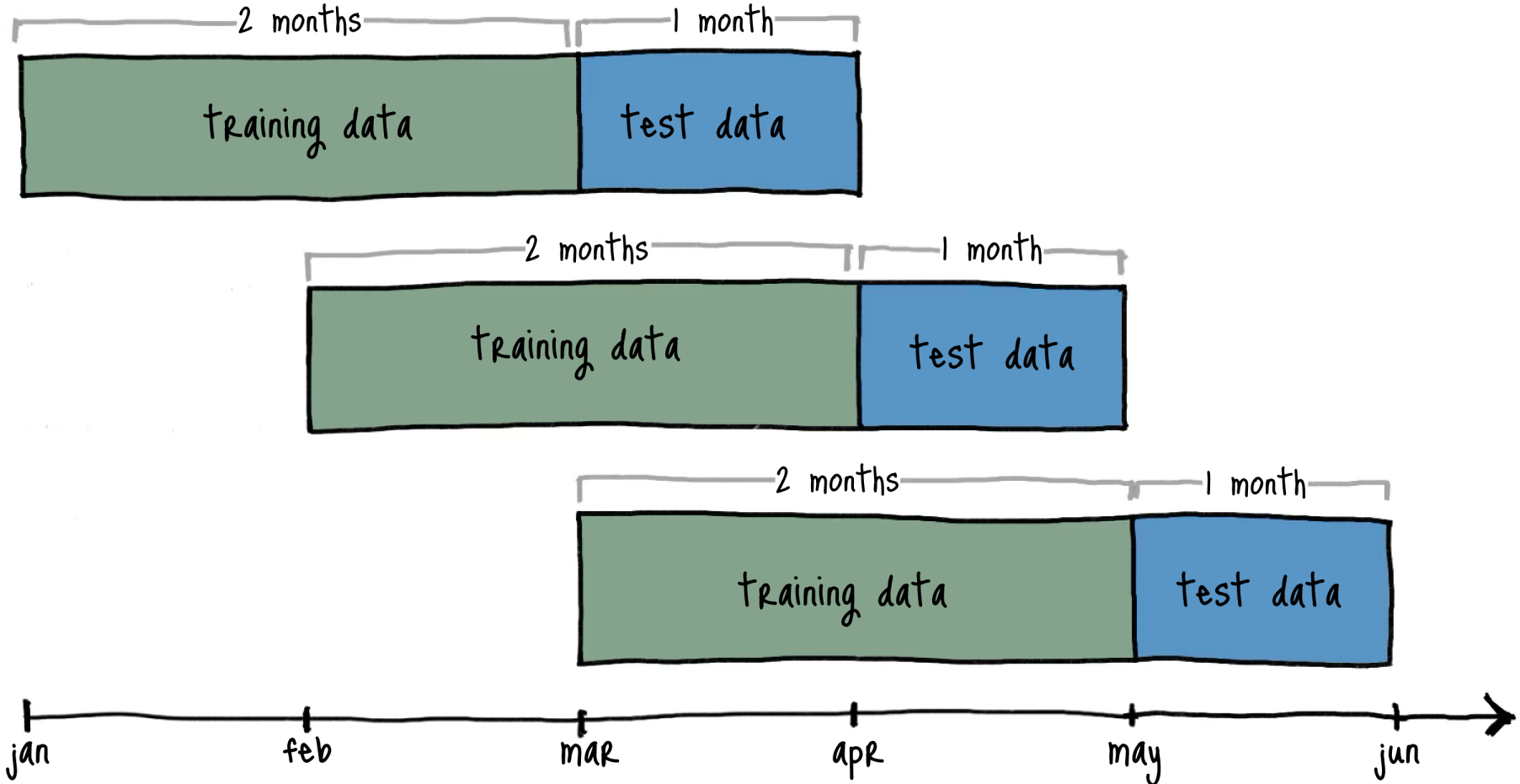
Prefer **cross-validation** over a single holdout split whenever possible to **reduce** evaluation **variance**. Fall back to a single split only when necessary.

Use **backtesting** to evaluate models on **historical data**. This technique helps ensure a model works reliably on **unseen** future scenarios.

time-based evaluation using a growing training set



time-based evaluation using a fixed lookback



best practice

Set up your data **chronologically** and use **backtesting** to evaluate how your model performs with **unseen** data under **real-world** conditions.

Invariance testing checks if predictions remain consistent after introducing **controlled changes** in the model inputs that should not alter the outcome.

Invariance testing is crucial for assessing both the **robustness** and **fairness** of models, helping prevent discrimination at scale.

invariance testing

testing focus

model consistency under controlled transformations

nature of test cases

rotations, translations, paraphrasing, feature permutations, etc.

failure cases

spurious correlations, overfitting to artifacts, lack of robustness

can we automate it

yes, using synthetic transformations

computer vision example

model should recognize objects even when rotated or resized

natural language example

a sentiment analysis model should output the same score for a paraphrased sentence

tabular data example

a credit risk model should not change its decision if the applicant's name is modified

invariance tests - examples

loan application system. model predicts whether a loan should be approved or denied

applicant	loan	income	zip	expected	prediction
john	$500k	$100k	33510	approved	approved
mary	$500k	$100k	33510	approved	denied
john	$500k	$100k	92100	approved	denied
john	$500k	100,000	33510	approved	denied

base case

model is oversensitive to irrelevant changes

best practice

Create a **small dataset** containing **irrelevant feature variations** and use it as part of the evaluation process to test the **consistency** of your model.

Behavioral testing sets up **stress tests** to simulate **critical scenarios** or **edge cases**, verifying that your model behaves correctly under these conditions.

Behavioral testing ensures models make **fair**, **logical**, and **reliable** decisions, preventing **harmful biases** and **errors** in real-world applications.

behavioral testing

testing focus

model's overall logical correctness and real-world alignment

nature of test cases

using realistic test scenarios based on business rules

failure cases

model producing incorrect, biased, or illogical responses

automation

often requires human domain knowledge to define expected behavior

computer vision example

a self-driving car should stop at a red light regardless of whether it's bright or dim

natural language example

a chatbot should respond differently to "how's the weather?" vs. "tell me a joke"

tabular data example

a loan approval model should grant loans to individuals with good credit but reject applicants with low credit scores

behavioral tests - examples

loan application system. model predicts whether a loan should be approved or denied

applicant	loan	income	zip	expected	prediction
john	$500k	$100k	33510	approved	approved
john	$1.2M	$100k	33510	denied	approved
john	$500k	$0	33510	denied	approved
john	$500k	$500k	33510	approved	denied

base case

model is not responsive enough to relevant changes

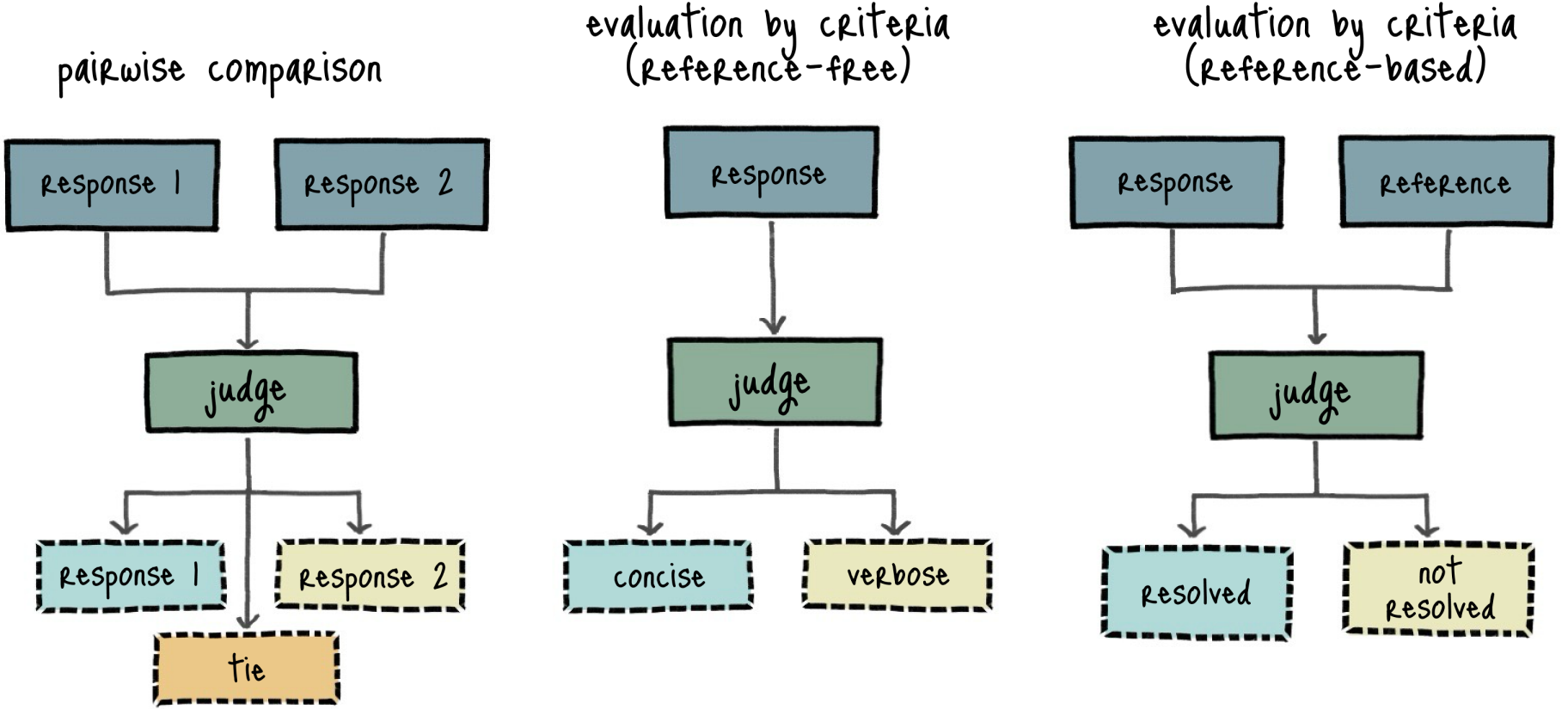
best practice

Create a **small dataset** containing **variations** that should **change model outputs** and use it as part of the evaluation process to test the **reliability** of your model.

LLM-as-a-Judge is a technique that uses an LLM to evaluate the **quality** of text outputs based on **custom criteria** defined in an evaluation prompt.

You can use LLM-as-a-Judge to choose the **best of two** answers, assess **specific qualities** of the answer, or evaluate the answer based on **additional context**.

llm-as-a-judge scenarios



llm-as-a-judge example prompts

pairwise comparison

given the question and the two different responses below, decide which response is better based on its relevance, helpfulness, and level of detail. if both responses are equally good, return "tie", otherwise return "response 1" or "response 2".

evaluation by criteria (reference-free)

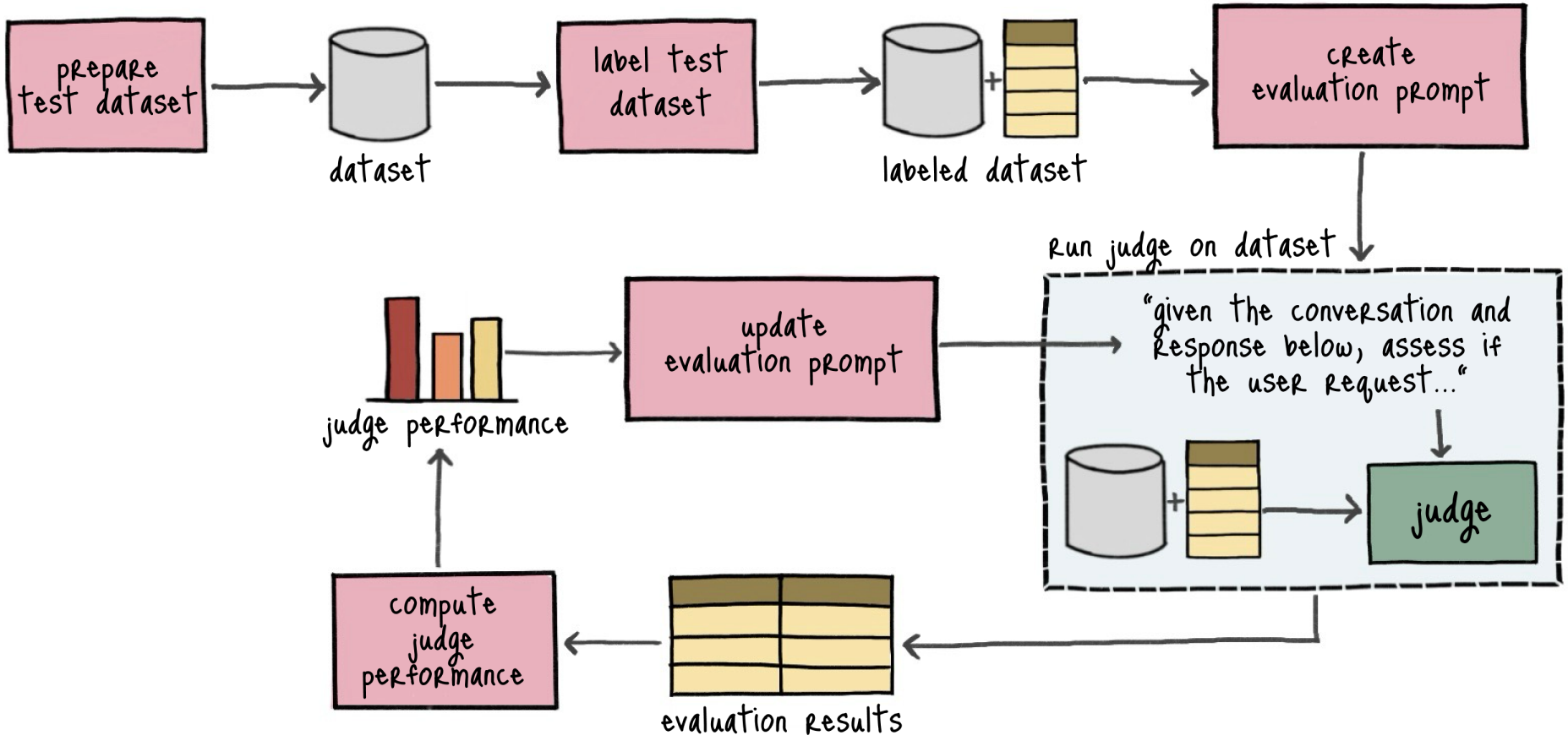
evaluate the following response for conciseness. a concise response is clear and direct, without unnecessary words. return either "concise" or "verbose".

evaluation by criteria (reference-based)

given the conversation and response below, assess if the user request was resolved. if the issue was addressed and the user confirmed it or showed satisfaction, return the label "resolved", otherwise return "not resolved".

To create a **judge**, start with a **labeled dataset**, design a clear **evaluation prompt**, and **iteratively** refine it to align the model's outputs with your expectations.

building an llm judge application

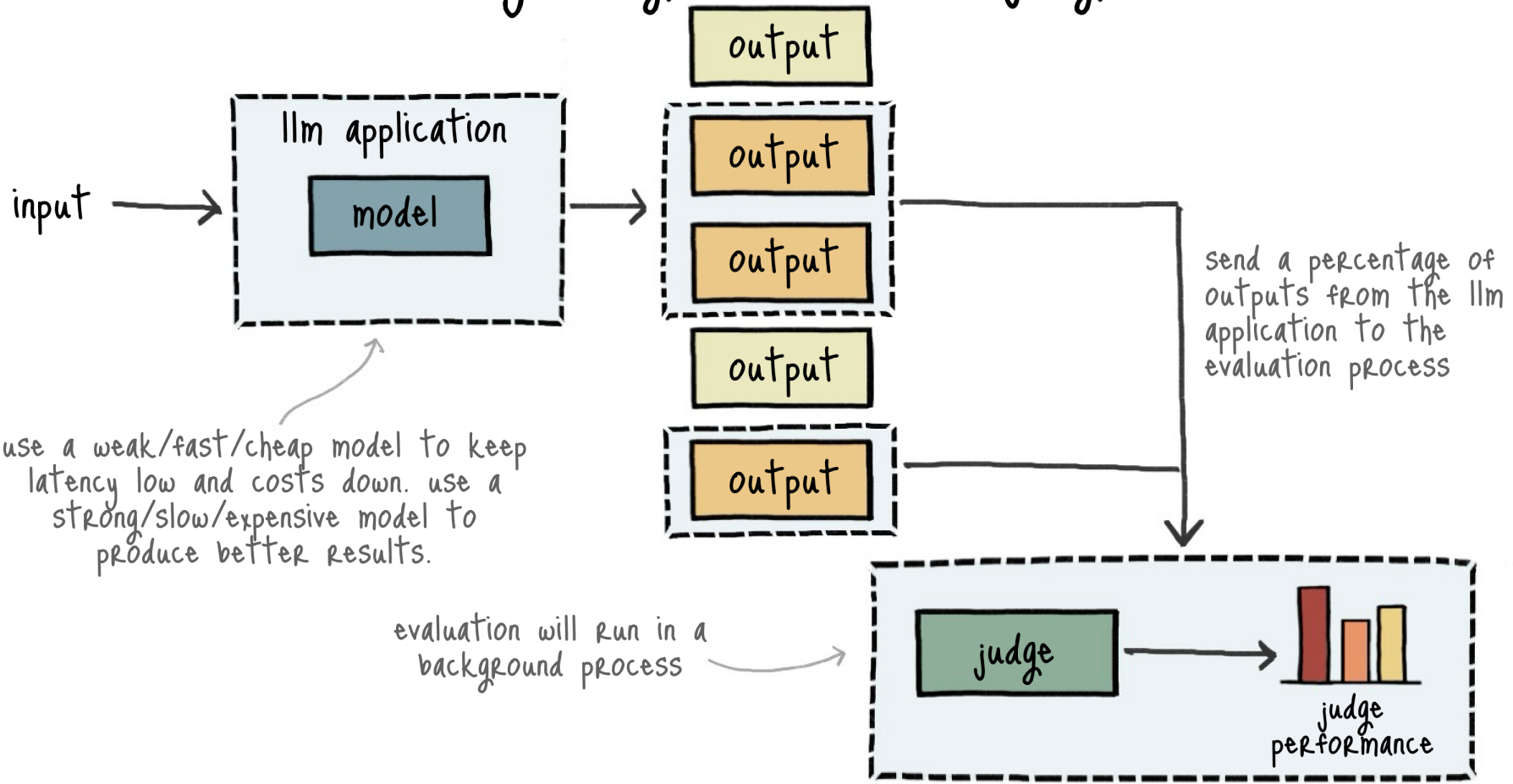


best practice

Build the judge using the **same** model you are using in your application. After you have the judge working, **replace it** with a smaller/cheaper model and **iterate**.

After building the judge, integrate it into your application and use it to **evaluate** a **percentage** of the **outputs** to detect drift and track any trends over time.

llm observability using an llm-as-a-judge evaluation



pros and cons of llm-as-a-judge evaluations

the good things

- produces high-quality evaluations closely matching human judgment
- simple to set up because they don't need reference answers
- flexible - you can evaluate anything
- scalable - they can handle multiple evaluations very fast
- easy to adjust as criteria change
- domain experts can participate in the prompt creation

the bad things

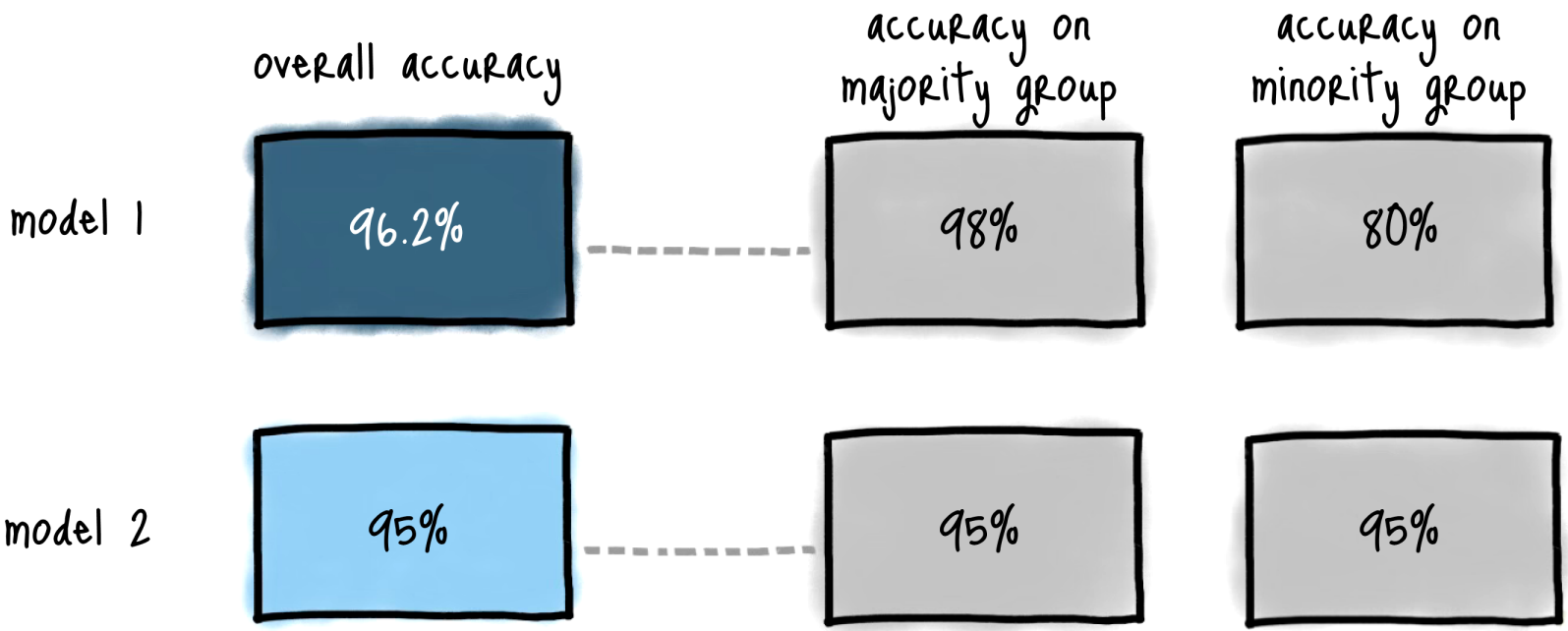
- llms are probabilistic, so slightly different prompts can lead to different outputs
- llms may suffer from self-bias, first-position bias, or verbosity bias
- using third-party llms for evaluations has privacy risks
- faster/cheaper than humans, but slower/more expensive than rule-based evaluations
- requires additional effort to prepare and run

best practice

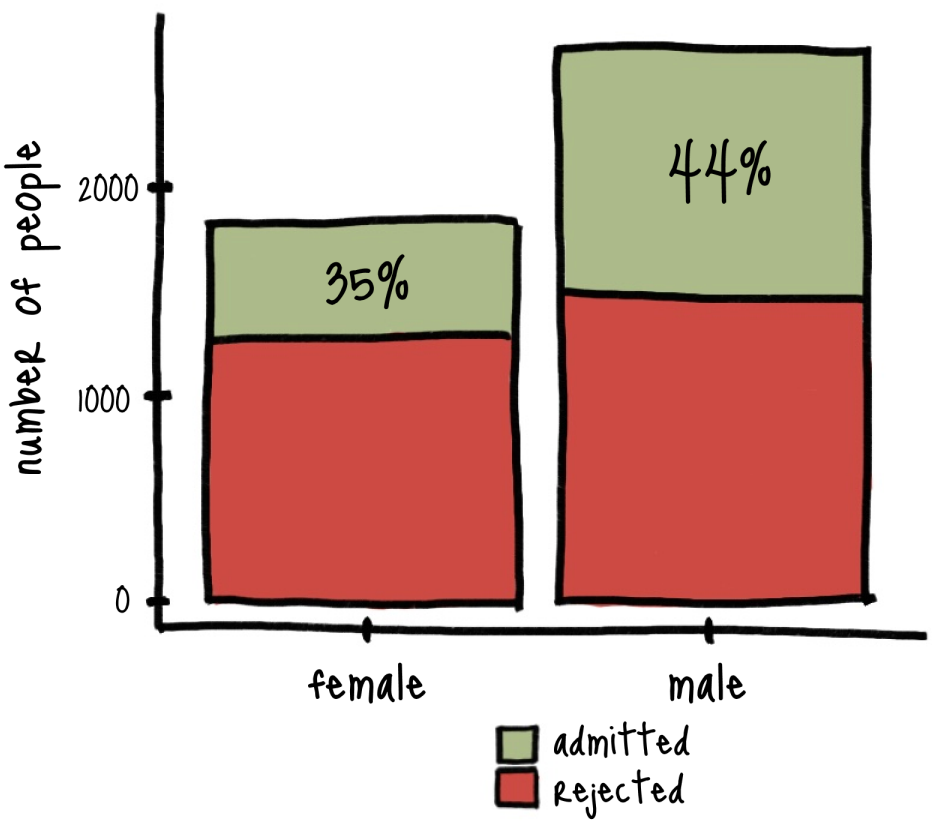
Do not use **opaque judges**. Any changes in the judge's model and prompt will change its results. If you can't see how the judge works, you can't interpret its results.

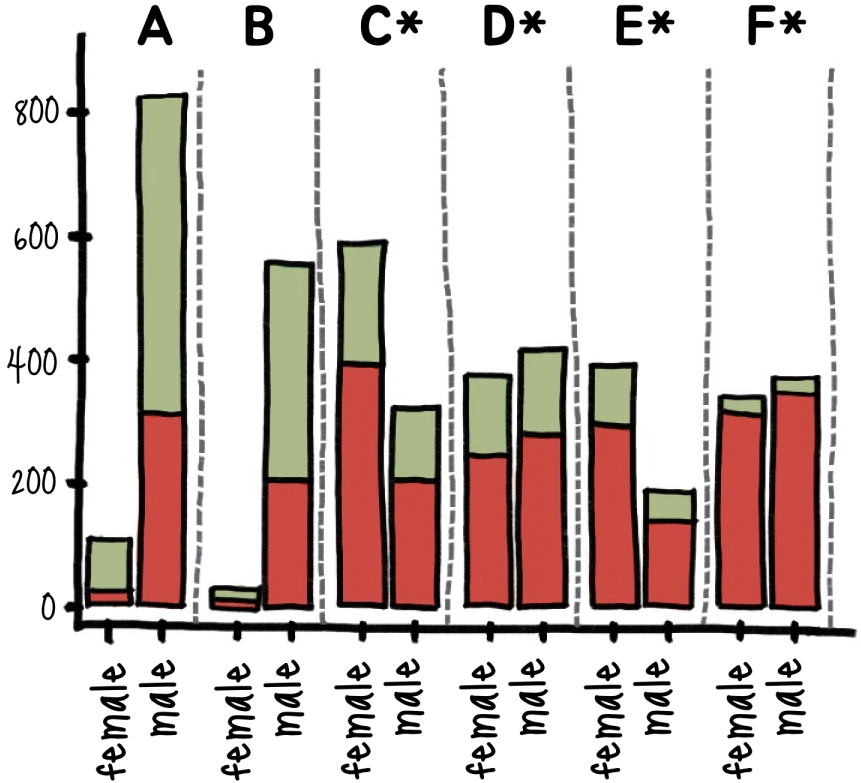
Summarization metrics **conceal how models work.** Reducing evaluation to a single metric may **hide** severe failure cases that happen on specific **slices** of the data.

predicting profitable loans



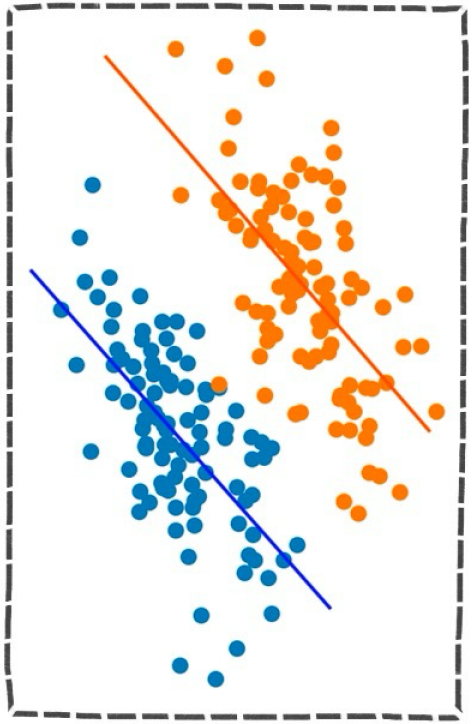
admissions at uc berkeley

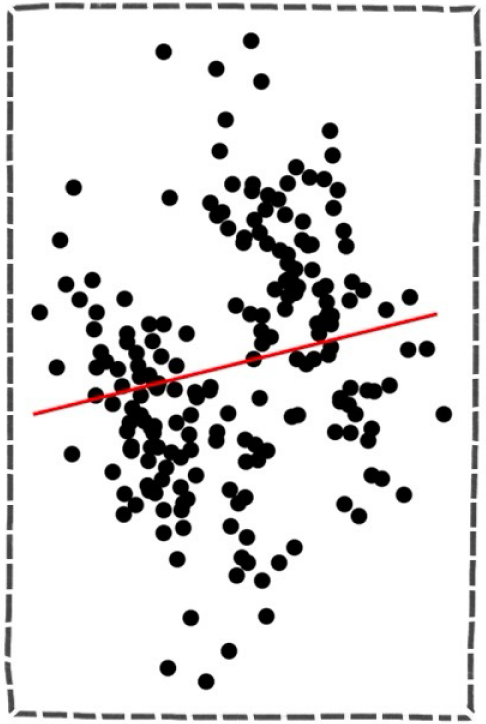




The **Simpson's Paradox** is a phenomenon in which a trend **appears** in separate subsets but **disappears** or **reverses** when we **combine** all the data together.

simpson's paradox



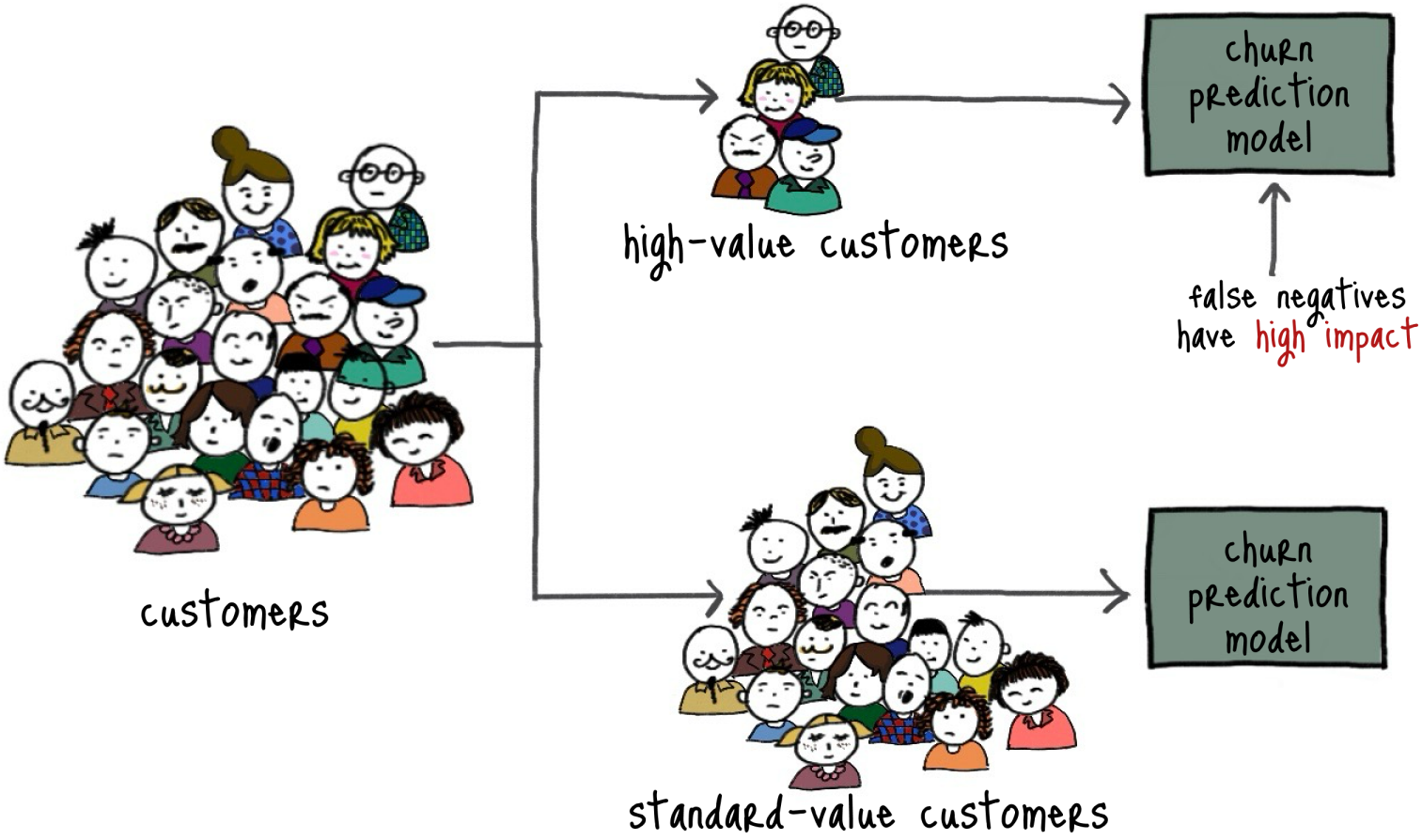


the trend from separate subsets reverses when we combine all the data together

best practice

Identify and **split** your data into meaningful **subsets.** Evaluate your model's performance on each of these subsets **separately** to uncover any weaknesses.

disproportionately important samples



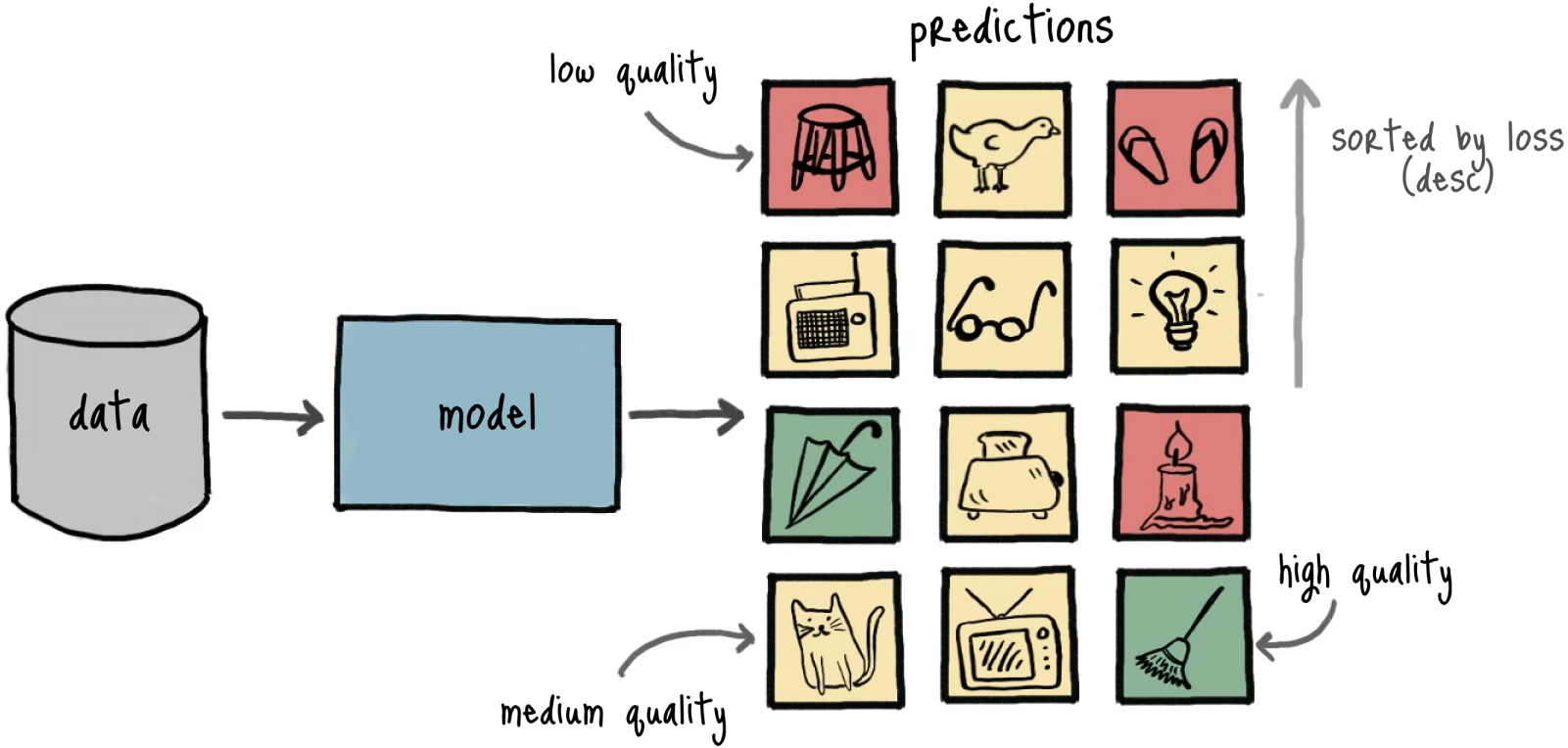
best practice

Create a dataset containing any **disproportionally important samples**, and evaluate and monitor your model's performance on them separately.

Error analysis is the process analyzing the mistakes a model makes on a validation or test dataset.

It helps identify **underperforming subpopulations**.

divide samples in subgroups | compute model's performance | establish a baseline | determine the size of each subgroup | compute potential impact



divide samples in subgroups

compute model's performance

establish a baseline

determine the size of each subgroup

compute potential impact

70%

on low quality images

90%

on high quality images

82%

on medium quality images

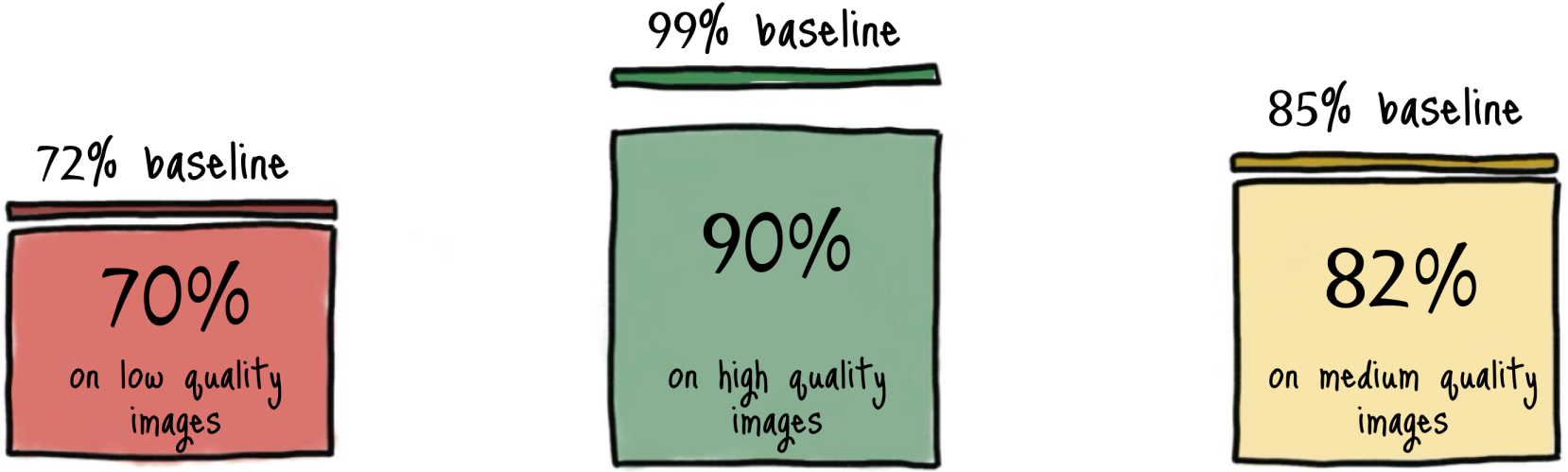
divide samples in subgroups

compute model's performance

establish a baseline

determine the size of each subgroup

compute potential impact



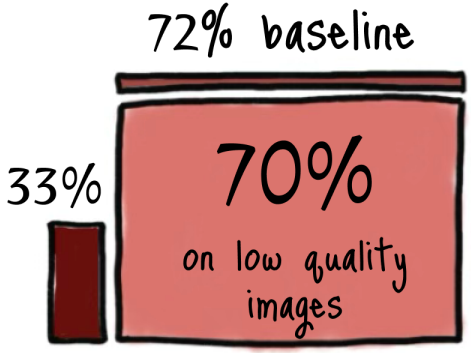
divide samples in subgroups

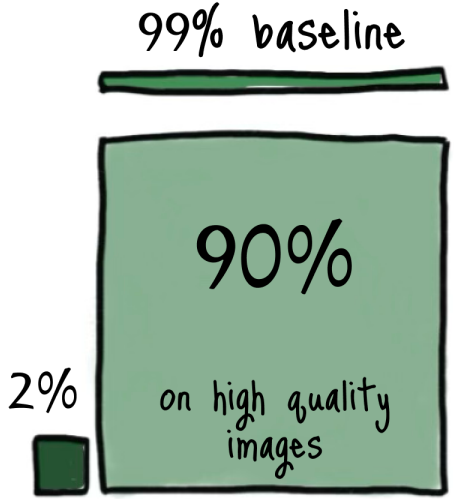
compute model's performance

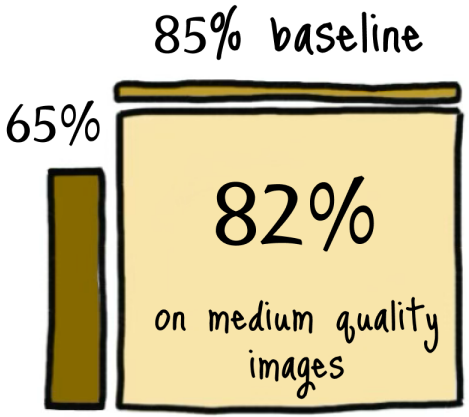
establish a baseline

determine the size of each subgroup

compute potential impact







divide samples in subgroups

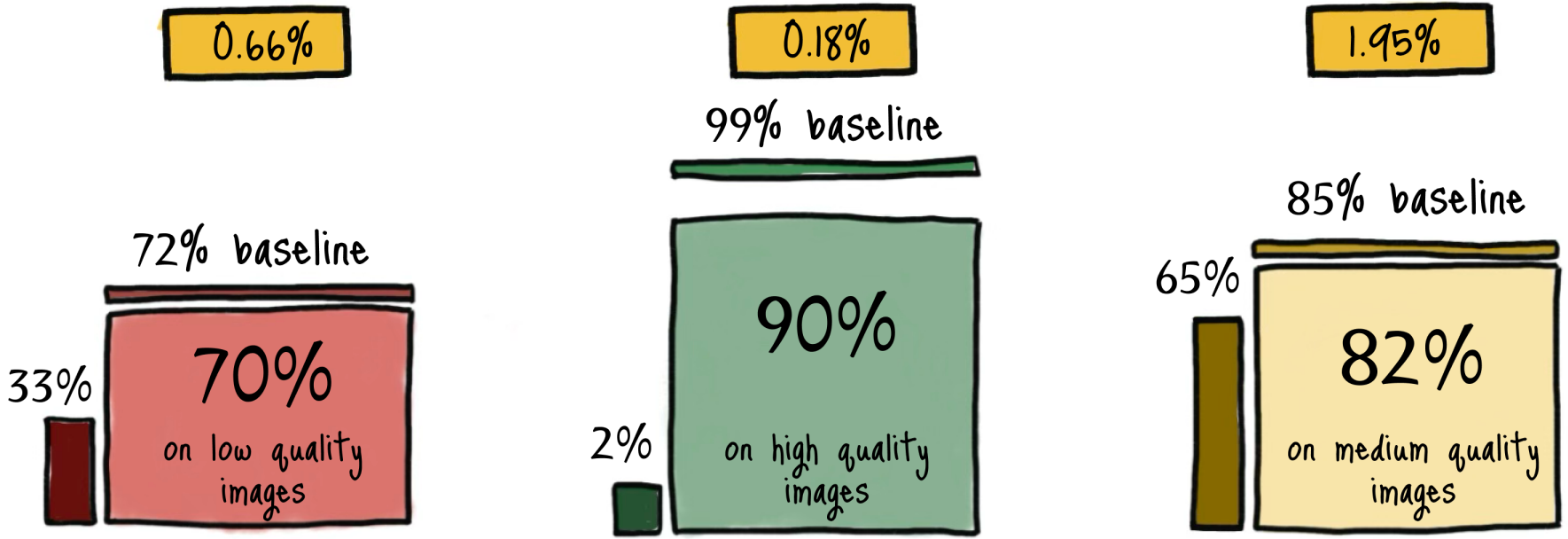
compute model's performance

establish a baseline

determine the size of each subgroup

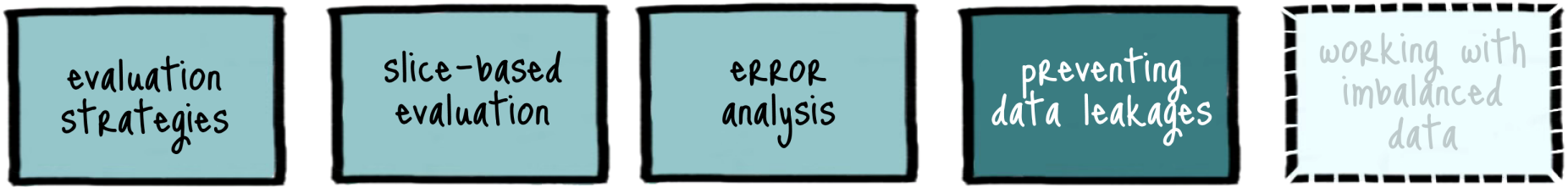
compute potential impact

impact = (baseline - performance) * frequency



best practice

Don’t waste time trying to fix **everything at once**.
Use **error analysis** to find **high-impact** areas in your data and focus your improvements there.



Data leakages lead to **useless** models.

A data leakage happens when information related to the ground truth is introduced within the training data.

data leakage in x-ray images

20,000 x-ray images
from 10,000 patients

random split

train data

16,000 x-ray images
80% of the dataset

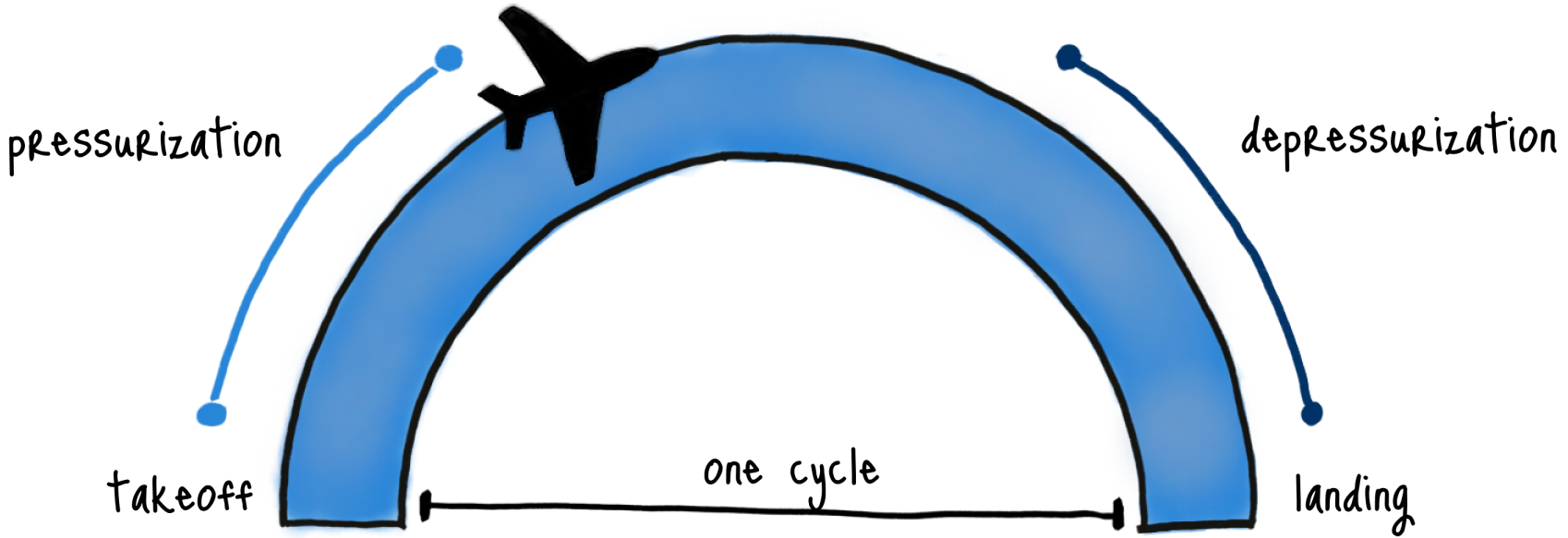
+

test data

4,000 x-ray images
20% of the dataset

All **feedback** channels are model **training** mechanisms. Every time you **touch** your **test data**, you are running the risk of **introducing leaks**.

the quality of test data measured in testing cycles



every time you use your testing data you are putting pressure on it and overfiting the model

best practice

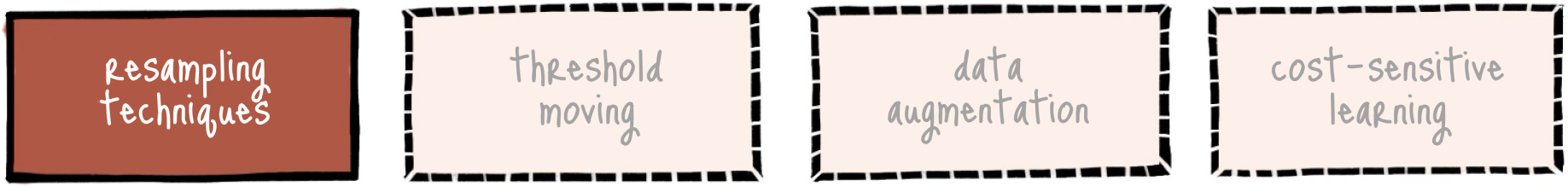
Set aside your **test data** and keep it fully **isolated**.

Use it only for the **final** model **evaluation**.

Consider **discarding** this data after using it.

Understanding how to evaluate models trained on **imbalanced data** is critical. In real-world applications, imbalanced classes often represent **high-stakes** cases.

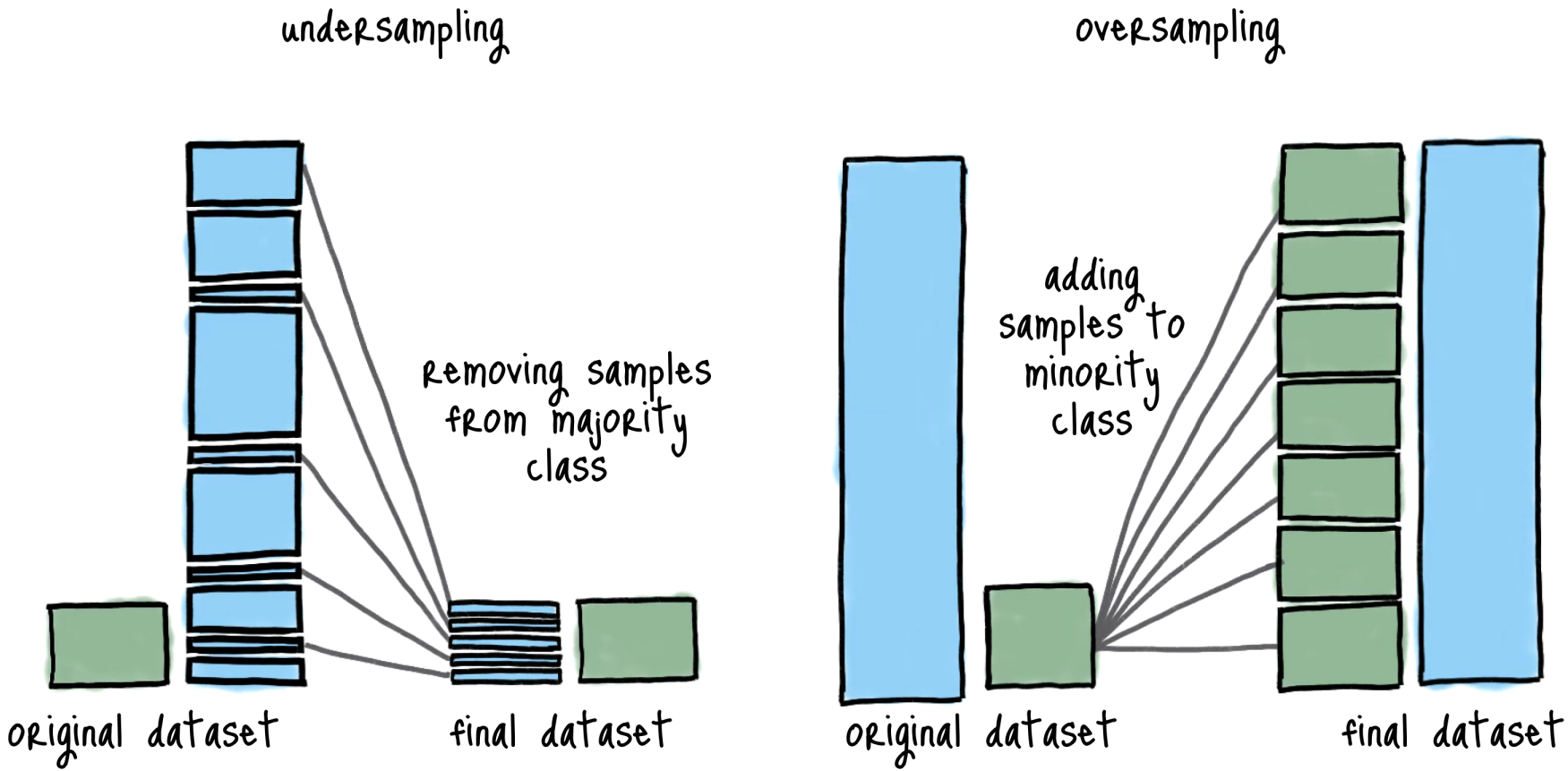
You can handle imbalances by **resampling**, using the **correct metrics**, **robust algorithms**, **class weights**, **cost-sensitive learning**, and **threshold moving**.

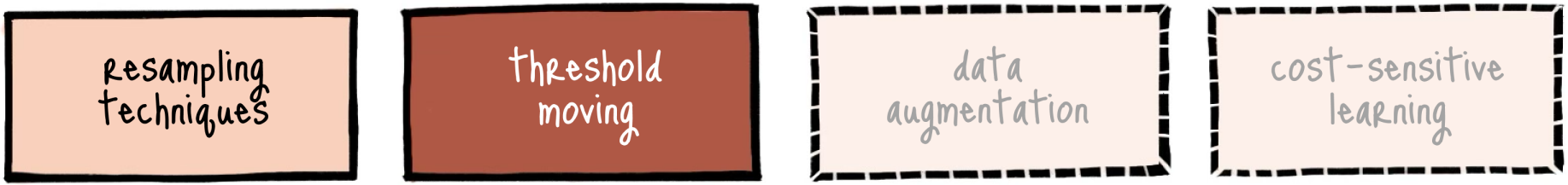


Resampling techniques **adjust** the number of samples in each class to create a more balanced dataset.

Be **cautious** when changing your data **distribution**.

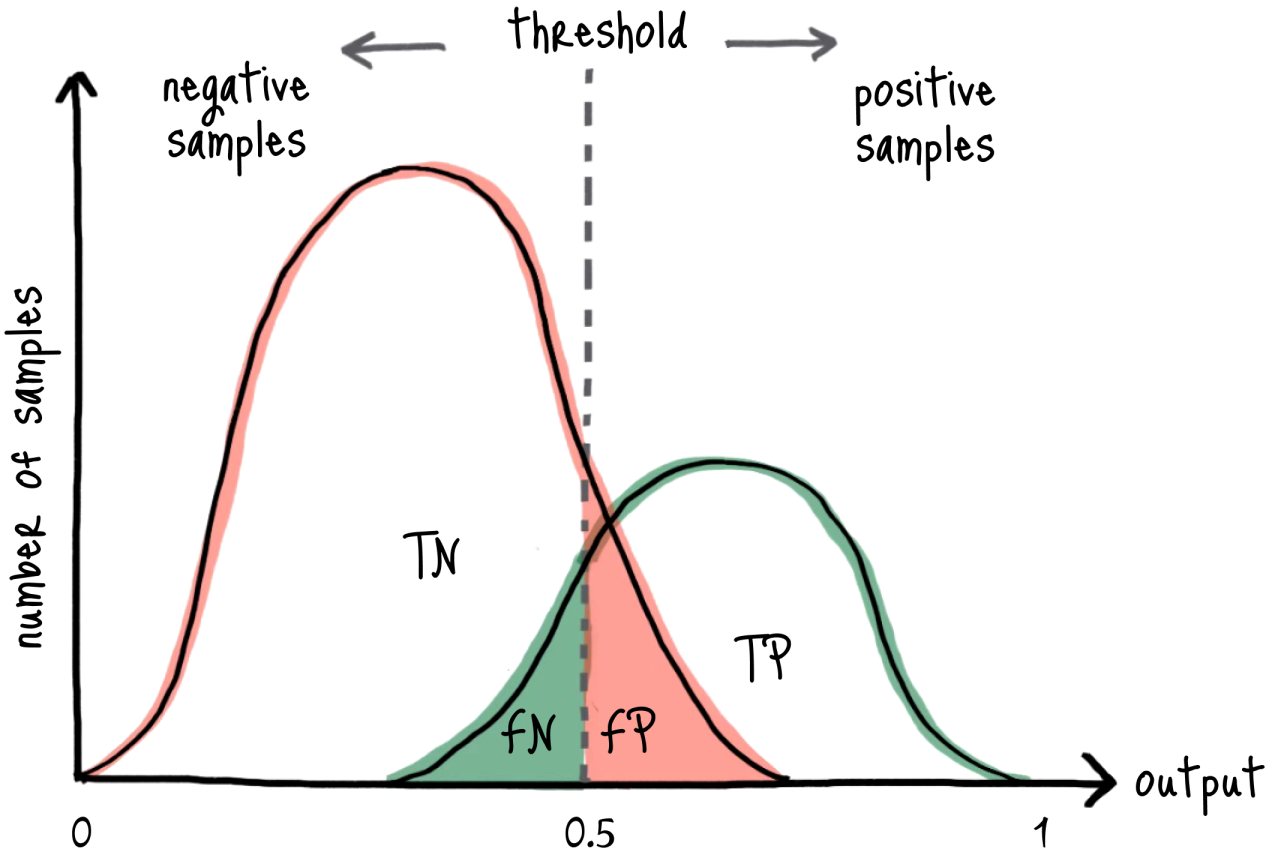
Resampling techniques





Threshold moving adjusts the **decision boundary** to classify samples based on their predicted probability. Thresholds can be learned using validation or test data.

threshold moving

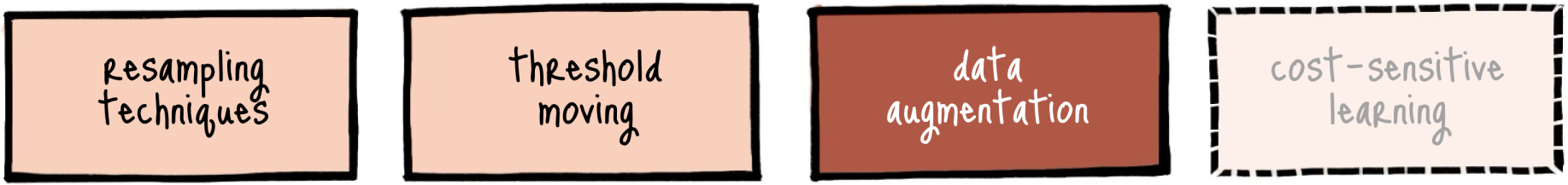


← threshold

- increases tp and fp
- decreases tn and fn
- higher sensitivity
- lower specificity

threshold →

- increases tn and fn
- decreases tp and fp
- lower sensitivity
- higher specificity



Data augmentation exposes a model to more aspects of the data, making it more **robust** to noise and improving its ability to **generalize** to **unseen** samples.

data augmentation

a cat

an upside down cat

a rotated cat

a flipped cat

an opaque cat

original sample

food was great

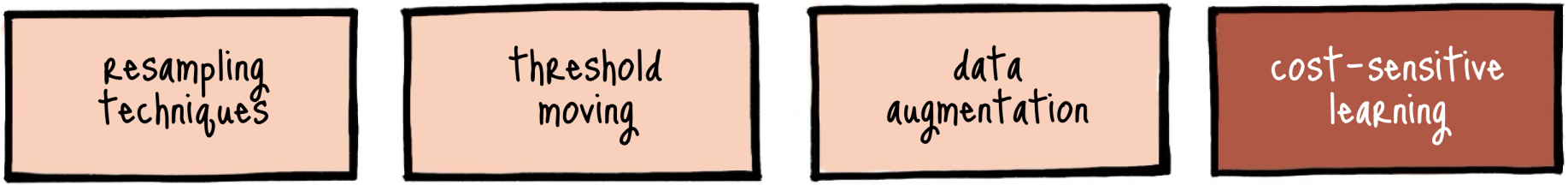
english to spanish

muy buena la comida

spanish to english

alternative sample

the food was fantastic!

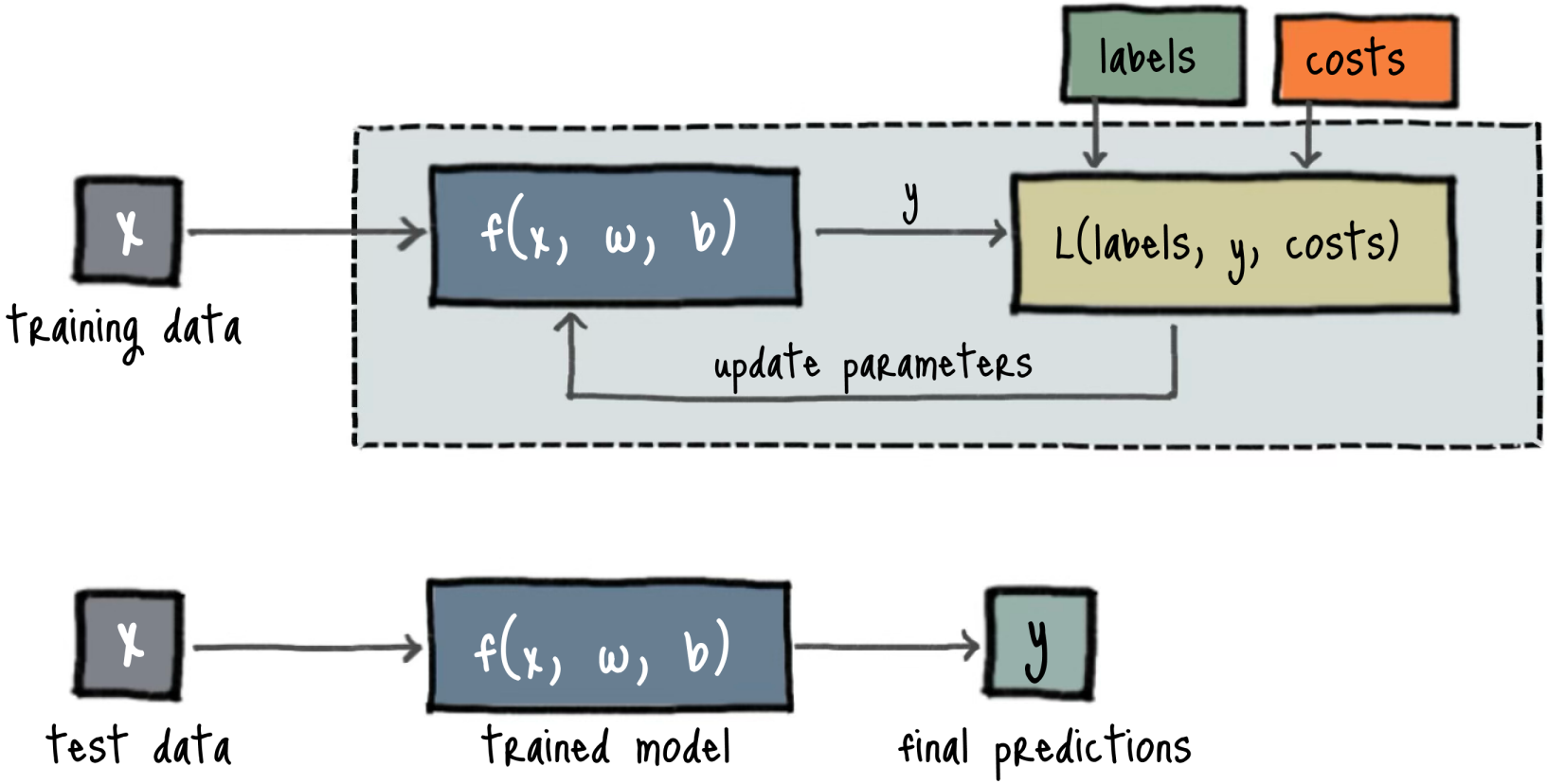


Cost-sensitive learning considers misclassification **costs** to help the model pay more **attention** to the minority class by **penalizing** mistakes more heavily.

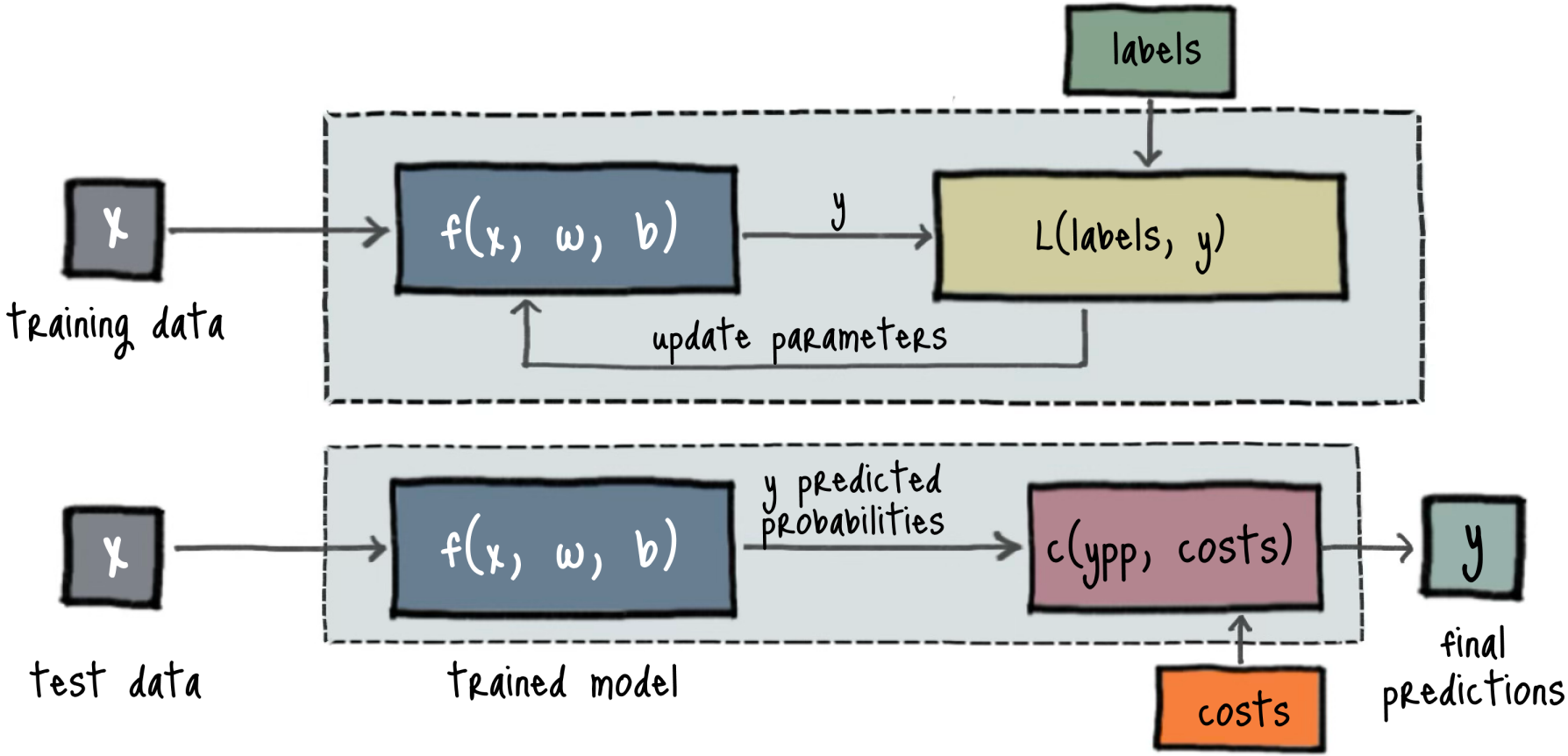
cost-sensitive learning

predicted class \ actual class	positive class	negative class
positive class	true positives (TP) cost(+, +)	false positives (FP) cost(+, -)
negative class	false negatives (FN) cost(-, +)	true negatives (TN) cost(-, -)

train-time cost-sensitive learning



test-time cost-sensitive learning



best practice

Treat class imbalances as a **signal**, not just a **problem**. Don’t rush to fix them—**analyze** how they **impact** your model first, as premature fixes can **bias** your model.

the end