Data Engineering for Machine Learning

Sampling, Class Imbalance, Feature Engineering, Distribution Shift

Acknowledgements: These lecture materials are adapted from the original notes provided by Prof. Vijaya Reddi and Chip Huyen. Any deviations, errors, or supplementary content in these slides are solely my responsibility.

Data is Source Code for an ML System

The following analogy for ML Systems: Dataset = source code. Pipeline = compiler. Weights = binary/executable.
Deleting a row of training data is deleting a line of code[1]

Conventional software	**ML data**
Fails loudly at a known line	Degrades silently on a subgroup
Stack trace localizes the fault	Symptom is an accuracy number
Fix is local	Fix may require reprocessing history
Tests assert on behavior	Tests must assert on *distributions*

[1] One may consider this as an extreme view but users do not typically uderstand data as well as code

Example

Example. A ZIP code field changes from string to integer.

Leading zeros are lost: $07102 \to 7102$.

Every record is still valid. The model rejects a whole region silently.

Median time to discovery: $\approx$ 4 weeks. (Sambasivan et al., CHI 2021)

Engineering an ML system: Model-centric vs. Data-centric

- **Model-centric:** hold data fixed. If performance stalls or assertions fail, swap architectures (e.g., ResNet to ViT, GBDT to XGBoost), tweaks learning rates, or adds regularization.
- **Data-centric:** hold the model fixed, systematically improve the data — sample, relabel, rebalance, de-duplicate, fill coverage gaps.

Roadmap

- Identify representative dataset (Sampling; BT)
- Label it. (Labeling; BT)
- Determine if its complete (Missingness; BT/AT)
- Determine if it is contaminated (Outliers; BT)
- Determine if it is skewed (Imbalance; BT)
- Determine if it is encoded once or twice (Features; BT)
- Determine if its distribution has changed (Shift; AT)

Sampling

Definition: selecting a subset of data from a larger population in order to make an inference about the population.
Two types of sampling:

- Non-probability sampling
- Probability-based sampling

In statistics: the population is given, the sample is chosen.
In ML: the sample arrives first, and the "population" is whatever the model will see in production.

Non-probability Sampling

- **Convenience:** samples selected by ease of access
- **Snowball:** future samples selected from existing ones.
  e.g. to scrape legitimate Twitter accounts, start with seed accounts and follow their following
- **Judgment:** experts decide what to include

Examples of Convenience-based Non Probability Sampling

Data in ML is almost always convenience-based. Some examples:

- Language models: BookCorpus, CommonCrawl, Wikipedia, Reddit links
- Sentiment analysis: IMDB, Amazon. Only users with Internet access who were motivated enough to write a review
- Self-driving cars: most data from the Bay Area (CA) and Phoenix (AZ). Very little rain or snow

In each case the sampling frame is not the population of interest. The system is least trained where it is most likely to fail.

Disadvantages of Non-probability Sampling

- Cannot compute probabilites of selection
- Cannot compute unbiased estimates of population quantities
- Cannot compute confidence intervals

In short: you cannot know what you do not know.

Probability Sampling

Every unit i, chosen from a population U has a *known* inclusion probability $\pi_i > 0$.

Lets say:

The Target (T): The true total sum across all units in the full population U:

$$T = \sum_{i \in U} y_i$$

The Sample (S): The subset of items you actually observe.

The Estimator ($\hat{T}$): A function of only the data you observed in S:

$$\hat{T} = \sum_{i \in S} \frac{y_i}{\pi_i}$$

Types of Probability Sampling

- Random sampling: simple, weighted, importance, reservoir ; Uniform: $\pi_i = \frac{n}{N}$ for everyone.

Types of Probability Sampling

- Random sampling: simple, weighted, importance, reservoir ; Uniform: $\pi_i = \frac{n}{N}$ for everyone.
- Stratified sampling: divide into strata, sample within each; Non-uniform: $\pi_i = \frac{n_h}{N_h}$ varies across strata h.

Types of Probability Sampling

- Random sampling: simple, weighted, importance, reservoir ; Uniform: $\pi_i = \frac{n}{N}$ for everyone.
- Stratified sampling: divide into strata, sample within each; Non-uniform: $\pi_i = \frac{n_h}{N_h}$ varies across strata h.
- Cluster sampling: divide into clusters, sample whole clusters; Non-uniform: Units in larger clusters often have different overall chances of being selected than units in smaller clusters.

Simple Random Sampling

Simple Random Sampling

- Each unit has an equal chance of selection, $\pi_i = n/N$
- E.g. select 10% of all samples in the population
- The baseline other designs are judged against
- What "shuffle and split" implicitly assumes

Where it fails?:

Simple Random Sampling

- Each unit has an equal chance of selection, $\pi_i = n/N$
- E.g. select 10% of all samples in the population
- The baseline other designs are judged against
- What "shuffle and split" implicitly assumes

Where it fails?: at 1% prevalence, a 1,000-unit sample gives you about *ten* positives.

Weighted Random Sampling

- Each element is assigned a weight by importance or relevance
- Higher weight $\Rightarrow$ higher chance of selection
- E.g. in fraud detection, upweight fraudulent transactions

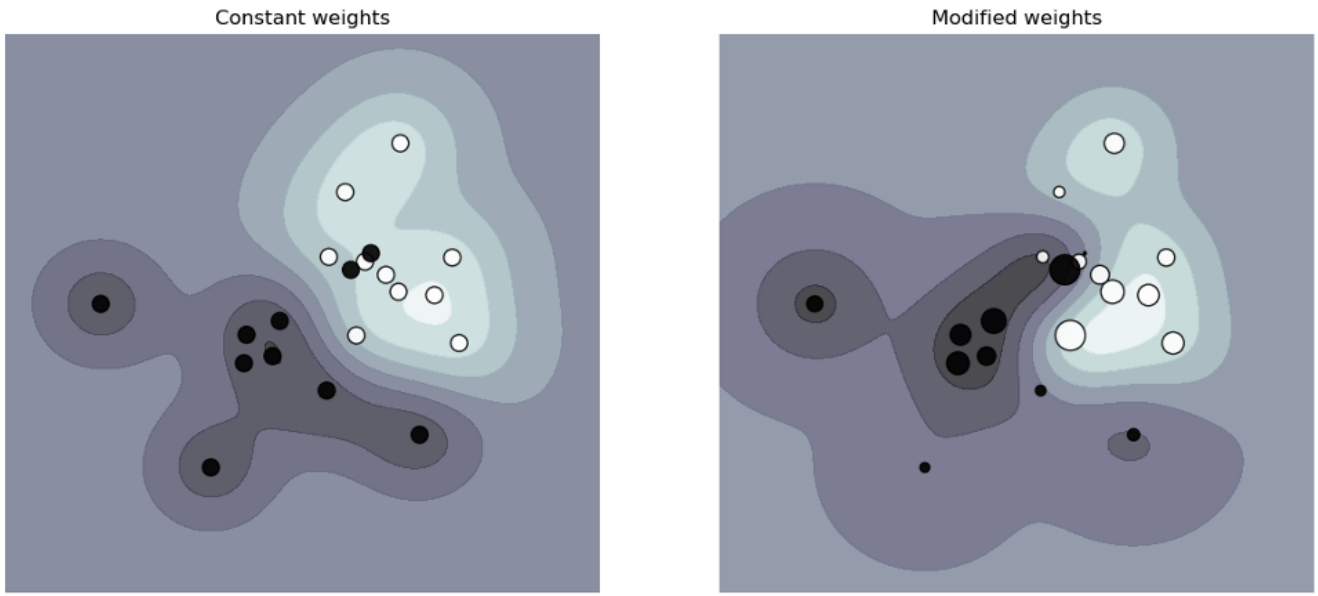
Choose two items so that 1, 2, 3, 4 each have a 20% chance and 100, 1000 each have 10%.

```
random.choices(population=[1, 2, 3, 4, 100, 1000],
                weights=[0.2, 0.2, 0.2, 0.2, 0.1,
0.1], k=2)
```

```
random.choices(population=[1, 1, 2, 2, 3, 3, 4, 4,
100, 1000], k=2)
```

Note: this samples *with* replacement. Without-replacement weighted designs do *not* have π_i proportional to the weights.

Weighted Sampling



How sample weights can affect the decision boundary.
On the left is when all samples are given equal weights. On the right is when samples are given different weights. Source: SVM: Weighted samples (sklearn).

Stratified Random Sampling

Used to highlight differences among groups in a population.

- Population is divided into subgroups (strata) based on certain characteristics
- Random samples are drawn from each stratum in proportion to its size in the population
- E.g. if population has 70% positive reviews and 30% negative reviews, then select 70% positive and 30% negative reviews in sample

Issues?

Cluster Sampling

- Divide the population into clusters
- Select *some* clusters at random; take *all* units inside them
- E.g. sample cities, then take every user in those cities

This is a **cost** design, not a precision design.

	Stratified	**Cluster**
Groups should be	homogeneous within	heterogeneous within
You sample	from all groups	some groups
Effect on variance	reduces	increases

Reservoir Sampling

- Select k samples from a "stream" of n samples, equal probability
- n is unknown; impossible or inefficient to hold all in memory
- Can stop the stream at any moment and hold a valid sample

Reservoir Sampling: Algorithm

Use case: For streaming data, we want to sample k items from a stream of unknown size n with equal probability.

- First k elements go into the reservoir
- For each incoming ith element, draw random j uniform in $[1, i]$
- If $1 \leq j \leq k$: replace the jth reservoir slot with element i; else discard
- Each element seen so far is in the reservoir with probability k/i

Why is this true?

Reservoir Sampling: Proof by Induction

Claim. After i items, each is in the reservoir w.p. k/i.

- **Base:** $i = k$. Immediate.
- **Step:** item $i+1$ enters w.p. $\frac{k}{i+1}$ by construction.
- An item already in the reservoir survives if either
  - the new item is rejected: $1 - \frac{k}{i+1}$, or
  - it is accepted but evicts one of the *other* $k-1$ slots: $\frac{k}{i+1} \cdot \frac{k-1}{k}$
- Sum: $1 - \frac{1}{i+1} = \frac{i}{i+1}$
- So its probability becomes $\frac{k}{i} \cdot \frac{i}{i+1} = \frac{k}{i+1}$. ■

Advantages of Reservoir Sampling

- Works for streams of unknown size
- Works for streams too large to hold in memory
- Can stop the stream at any moment and hold a valid sample
- If the stream is non-stationary (e.g., ordered chronologically), this sample only represents the historical distribution up to time M, containing zero representation of future temporal shifts.

Importance Sampling

Use case: Used when direct sampling is infeasible.
$p(x)$ is the target we want; we can only sample from $q(x)$.

- Draw from a target distribution that is *different* from the sampling target
- Each sample is assigned a weight based on the ratio of the target distribution to the sampling distribution

$$\mathbb{E}_p[f(X)] = \int f(x)\, p(x)\, dx = \int f(x)\, \frac{p(x)}{q(x)}\, q(x)\, dx = \mathbb{E}_q[f(X)\, w(X)]$$

$$w(x) = \frac{p(x)}{q(x)}, \qquad \hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} f(x_i)\, w(x_i), \quad x_i \sim q$$

Importance Sampling: An Illustration

- Compute $\mathbb{E}[X^2]$ under $p(x) = \mathcal{N}(0, 1)$
- Suppose we cannot sample p directly, but can sample $q(x) = \mathcal{N}(0, 4)$ (a *normal* with $\sigma = 2$)
- The weight is the *ratio* of the two densities:

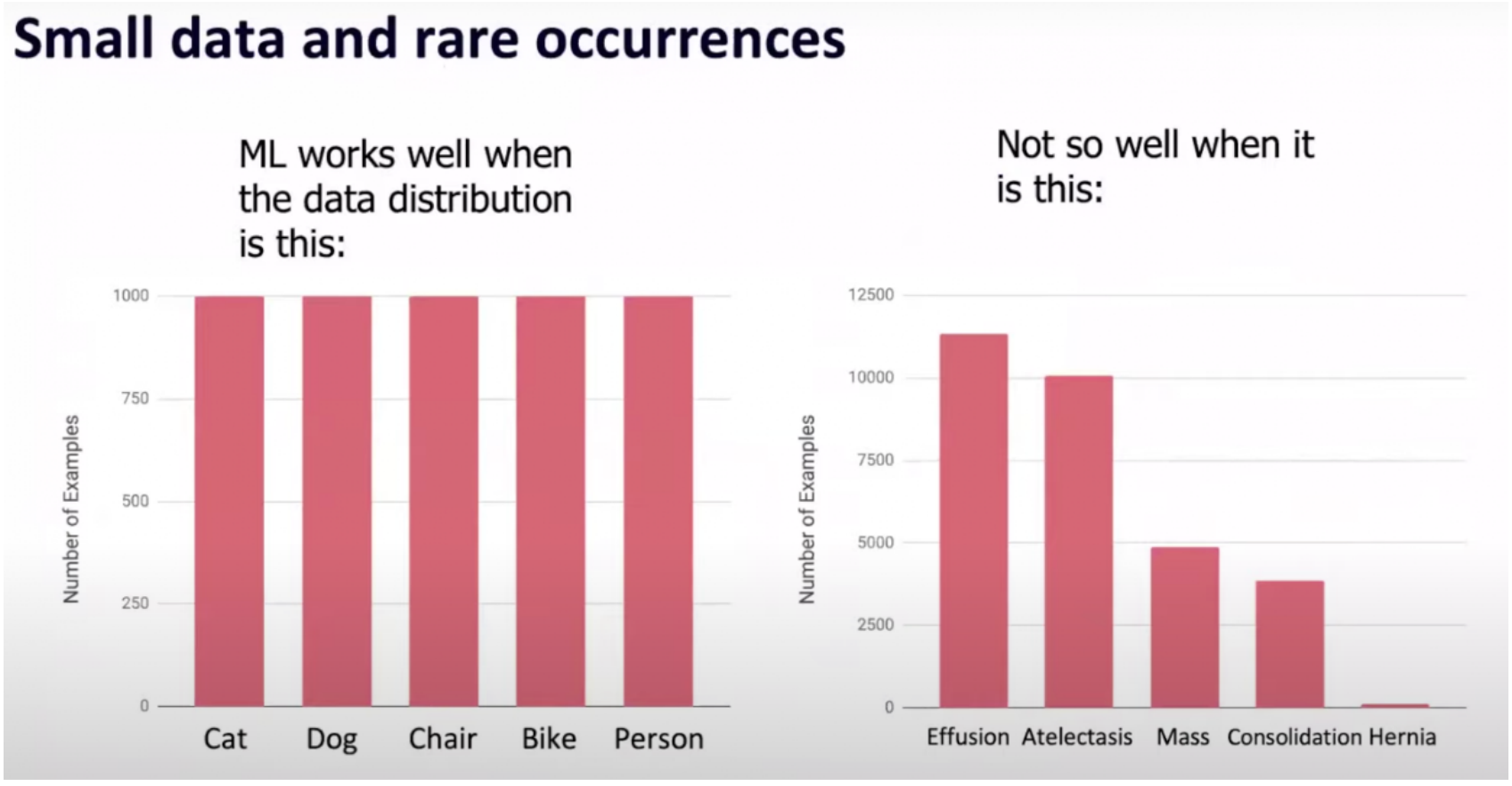
$$w(x) = \frac{p(x)}{q(x)} = \frac{\frac{1}{\sqrt{2\pi}} e^{-x^2/2}}{\frac{1}{2\sqrt{2\pi}} e^{-x^2/8}} = 2\, e^{-3x^2/8}$$

- Draw $x_1, \ldots, x_n \sim q$, then $\hat{\mathbb{E}}[X^2] = \frac{1}{n} \sum_i x_i^2\, w(x_i)$
- As $n \to \infty$, $\hat{\mathbb{E}}[X^2] \to 1$

More Data Is Not Automatically Better

Collecting data without a deliberate sampling design is expensive, ineffective, and potentially harmful.

Class Imbalance



Class Imbalance: Definition

- A large disparity in class representation during training
- Real examples: ?

Class Imbalance: Definition

- A large disparity in class representation during training
- Real examples: ?

Fraud $\approx$ 0.1–0.2% positive. Rare-disease screening, defect detection, intrusion detection, adverse events — all the same regime. Note: these are exactly the applications where the minority class is the entire reason the ML system is being developed.

Ratio Vs absolute number

ratio is less important than the **absolute number** of minority examples.

- 1000:1 with ten million positives — manageable
- 10:1 with forty positives — not

It is important to determine the absolute count before reaching for a resampling method.

Why Is Class Imbalance a Problem?

- A highly accurate model may do worse on the imbalanced class
- A model that predicts "not fraud" 99.8% of the time is 99.8% accurate
- Simply print it?

Your model can be beaten by a non-model such as `print("negative")`. So it is not predicting the way you want.

Why Cs class Imbalance a problem?

1. The overall accuracy and error rate are the most frequently used metrics to report the performance of ML models.
2. However, they are insufficient metrics for tasks with class imbalance because they treat all classes equally, which means the performance of your model on the majority class will dominate the accuracy.

Consider 2 models and a baseline

Model A	Actual CANCER	Actual NORMAL
Predicted CANCER	10	10
Predicted NORMAL	90	890

Model B	Actual CANCER	Actual NORMAL
Predicted CANCER	90	90
Predicted NORMAL	10	810

Baseline: Simply print"healthy"

Accuracy?

What is the accuracy of each model?

Accuracy?

What is the accuracy of each model?

1. Global accuracy treats all errors equally.
2. But in the real world, confusing a healthy patient for a sick one (FP) and missing a sick patient (FN) carry wildly asymmetric costs.
3. To evaluate these systems properly, we must consider alternate metrics such as Recall (did we catch the sickness?) and Precision (when there is cancer, does the model actually detect cancerf?).

Confusion Matrix

Model A	Actual Positive (Cancer)	Actual Negative (Normal)
Predicted Positive (Cancer)	TP	FP
Predicted Negative (Normal)	FN	TN

- True Positive (TP): Reality is sick, model correctly alarms sick. (Success)
- False Positive (FP): Reality is healthy, model incorrectly alarms sick. (Type I Error: "False Alarm")
- False Negative (FN): Reality is sick, model incorrectly claims healthy. (Type II Error: "Missed Disease")
- True Negative (TN): Reality is healthy, model correctly claims healthy. (Success)

Precision vs. Recall vs. F1

- Precision $= \frac{TP}{TP+FP}$
- Recall $= \frac{TP}{TP+FN}$
- F1 $= 2 \cdot \frac{P \cdot R}{P+R} = \frac{2TP}{2TP+FP+FN}$

Precision vs. Recall vs. F1

- Precision $= \frac{TP}{TP+FP}$
- Recall $= \frac{TP}{TP+FN}$
- F1 $= 2 \cdot \frac{P \cdot R}{P+R} = \frac{2TP}{2TP+FP+FN}$

Model	CANCER (1)	NORMAL (0)	Acc.	Prec.	Recall	F1
A	10/100	890/900	0.90	0.50	0.10	0.17
B	90/100	810/900	0.90	0.50	0.90	0.64

Identical accuracy. Identical precision. One model misses 90 of 100 cancers; the other catches 90.

Recall separates them.

When to Use Which Metric?

No one size fits all. Which metric in each case? Infectious disease; Crime prevention; Automated bug finding; Medical imaging

When to Use Which Metric?

No one size fits all. Which metric in each case? Infectious disease; Crime prevention; Automated bug finding; Medical imaging

Setting	**Emphasis**	**Reasoning**
Infectious disease	Recall	A missed case seeds transmission; a false positive costs one confirmatory test
Crime prevention	Precision	A false positive is a serious harm imposed on an individual
Automated bug finding	Precision	Developers abandon a linter that cries wolf. Unused tool $\Rightarrow$ zero recall
Medical imaging	Stage-dependent	Screening favours recall; confirmation favours precision

Classification as Continuous Scoring & Thresholding

- Most classifiers output a continuous score or calibrated probability:

$$\hat{p}(x) \in [0, 1]$$

- A hard label requires choosing an operating threshold τ:

$$\hat{y} = \mathbf{1}[\hat{p}(x) > \tau]$$

- The default $\tau = 0.5$ is completely arbitrary:
  - **Decrease τ (0.1):** Flag everyone with slight suspicion of disease; Recall shoots up but false alarms (FPR) also rises.
  - **Increase τ (0.9):** High-confidence alerting (only flag patients when certain); suppresses false alarms ($\downarrow \mathrm{FPR}$), but misses edge cases ($\downarrow \mathrm{Recall}$).

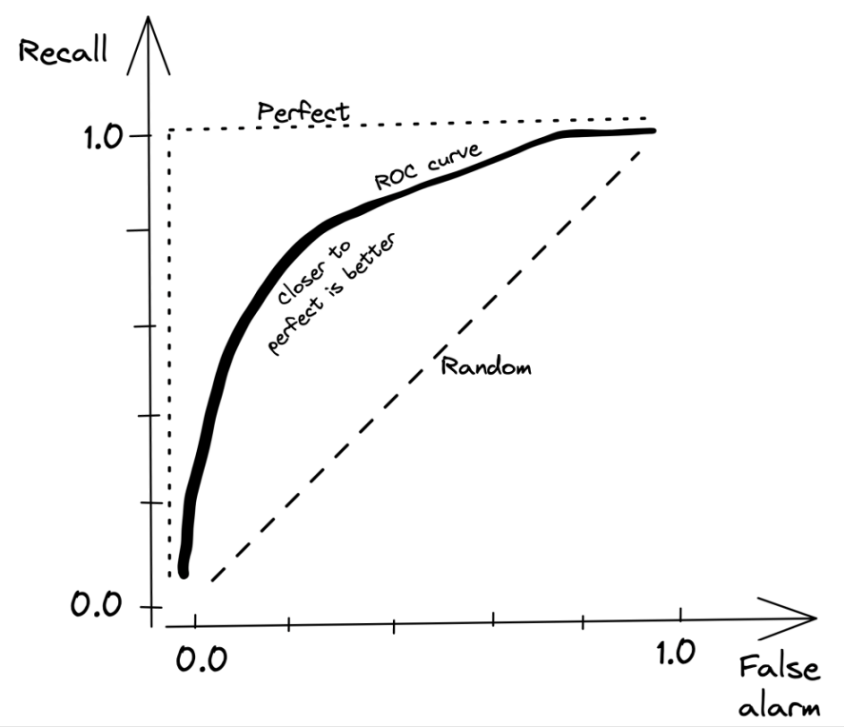
Evaluating at a single fixed threshold obscures the underlying ranking capability.

The Receiver Operating Characteristic (ROC)

- Sweeps threshold $\tau \in [0, 1]$ and tracks the trade-off across coordinates:

$$\text{Y-axis: } \mathrm{TPR/Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \qquad \text{X-axis: } \mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}$$

ROC-AUC



- **ROC-area under the curve(AUC)** summarizes performance across all operational operating points:
  - $\mathrm{AUC} = 0.5 \implies$ Random guessing (diagonal line).
  - $\mathrm{AUC} = 1.0 \implies$ Separable distributions (hugs top-left corner).

ROC Is Misleading Under Severe Imbalance

- Inspect the denominator of the False Positive Rate:

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathbf{TN}} = \frac{\mathrm{FP}}{\mathbf{N}^-}$$

- When $N^- \gg N^+$ (e.g., 100 positives (cancer) vs. 1,000,000 negatives (normal)):
  - Admitting 1,000 false alarms yields $\mathrm{FPR} \approx 0.001$ (0.1%).
  - The ROC curve hugs the vertical axis, posting an impressive $\mathrm{AUC} > 0.98$.
- **The Operational Reality:** A clinician receives 1,000 false alarms for only 100 real cases. An overwhelming majority of alerts are false, completely masked by massive *TN* support.

A tiny FPR times a massive negative population produces an avalanche of false alarms.

Precision–Recall (PR) Curves & Evaluation Policy

- **Isolate the Target Class (Davis & Goadrich, 2006):** Plot Precision vs. Recall by dropping True Negatives (TN) entirely:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \mathbf{FP}}, \qquad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- In the previous 1,000-FP scenario, Precision drops to $\leq 9.1\%$: the PR curve collapses visibly and communicates operational failure.

Addressing Class Imbalance

- Data-driven approaches
- Algorithmic approaches

Data-driven Approaches

- **Undersampling:** remove samples from the majority class.
  Loses information — but when the majority has millions of near-redundant examples the loss is often negligible, and training gets much faster
- **Oversampling:** duplicate minority examples.
  No new information; with flexible models it encourages memorization of the duplicated points

Issues: loss of information vs. can cause overfitting.

SMOTE

Synthesize minority samples as convex ($\approx$ linear) combinations of existing points and their nearest neighbors of the same class.

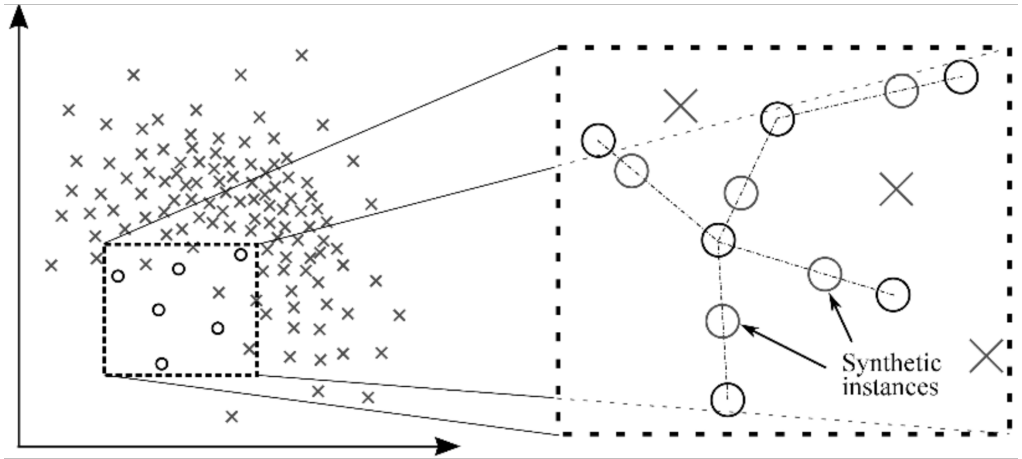


Figure: SMOTE: Synthetic Minority Over-sampling Technique (Chawla et al., 2002)

SMOTE: Algorithm

- Choose a random example from the minority class
- Find its k nearest minority neighbors (typically $k = 5$)
- Pick one neighbor at random; create a synthetic example at a random point on the segment between the two, in feature space
- Repeat until the desired balance is reached

Unlike duplication, this *expands* the region of feature space the model associates with the minority class.

Resample After Splitting. Never Before.

1. If you run SMOTE on your entire dataset before creating your train and test splits, you now have original point A, point B, and their synthetic clones scattered across your pool.
2. This leads to leaks; When you do your train/test split: Point A and a synthetic copy end up in the Training set. Another synthetic copy (or point B) ends up in the Test set.
3. The purpose of a test set is to evaluate whether the model can generalize to unseen data.
4. **Resample inside the training fold only**

SMOTE: Limits

- The segment between two minority points can pass through majority territory — synthesis can manufacture points inside the wrong class. Borderline-SMOTE and ADASYN concentrate synthesis near the boundary
- Linear interpolation is meaningful in low-dimensional continuous spaces, and much less so in high-dimensional or learned representations
- The original formulation is binary classification, continuous features only
- Authors' own recommendation: random undersampling of the majority, *then* SMOTE the minority

Algorithmic Approaches

- Naive loss: all samples contribute equally, $L(X;\theta) = \sum_x L(x;\theta)$
- Idea: the samples we care about should contribute more
- **Cost-sensitive learning:** assign different costs to different types of misclassification, and modify the loss to incorporate them
- Issues?

Cost-sensitive Learning

- C_{ij}: the cost if class i is classified as class j
- The loss for an instance x of class i becomes the weighted average over all possible classifications:

$$\text{Loss}(x \mid \theta) = \sum_j C_{ij}\, P(j \mid x; \theta)$$

- Minimizing expected *cost* rather than error rate
- The cost matrix need not be symmetric, or have a zero diagonal
- **Focal loss** takes another route: down-weight examples the model already classifies confidently

High cost for false negatives raises recall and lowers precision.