# Production: Working with Training Data

```
$ echo "Data Science Institute"
```

# Agenda

### 3.1 Working with Training Data

- Sampling
- Labeling
- Class Imbalance
- Data Augmentation

### 3.2 A Training Pipeline

- Sampling in Python.
- An initial training pipeline.
- Modularizing the training pipeline.
- Decoupling settings, parameters, data, code, and results.

# Slides, Notebooks, and Code

- These notes are based on Chapter 4 of *Designing Machine Learning Systems*, by Chip Huyen.

### Notebooks

- `./02_notebooks/production_3_1_sampling.ipynb`
- `./02_notebooks/production_3_2_pipeline.ipynb`

### Code

- `./05_src/credit_experiment_nb.py`
- `./05_src/credit_preproc_ingredient.py`
- `./05_src/`
- `./05_src/.env`

# Our Reference Architecture

# The Flock Reference Architecture

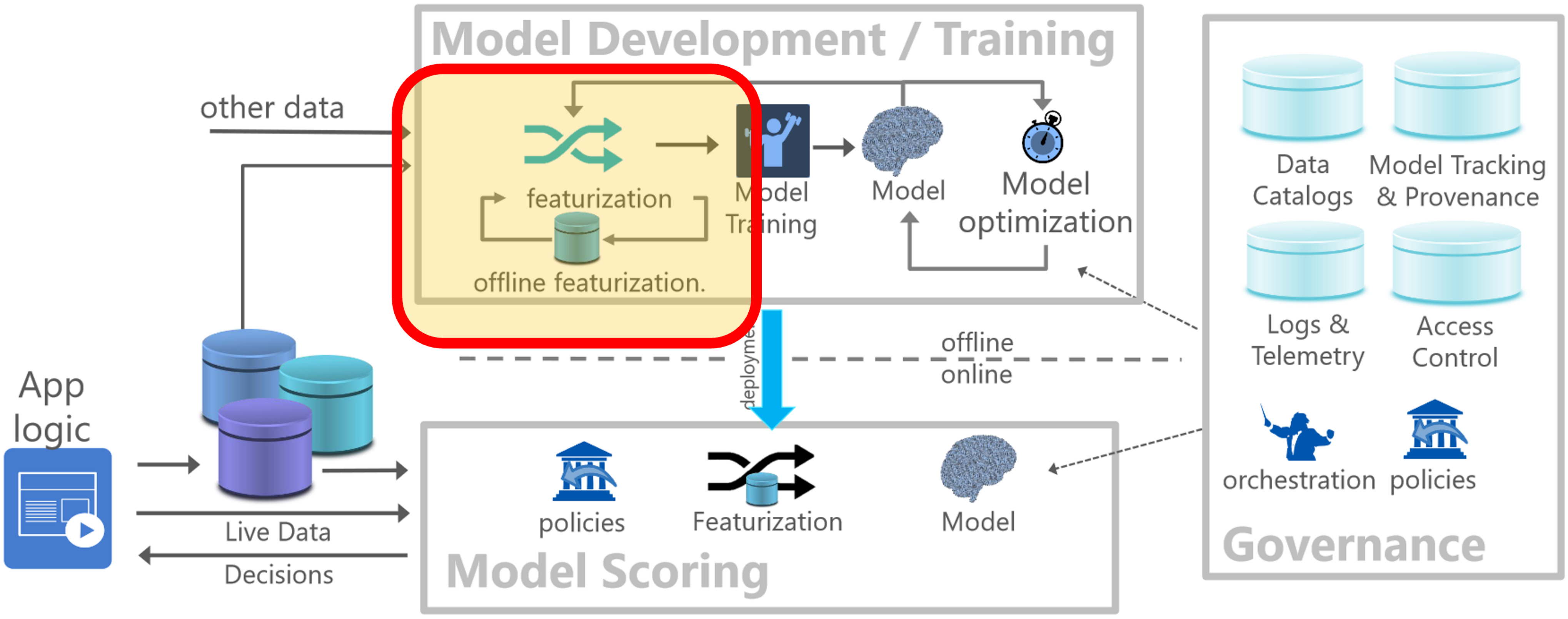


Figure 1: Flock reference architecture for a canonical data science lifecycle.

Agrawal et al (2019)

# Sampling

# Why Sample?

- Sampling is embedded across the ML lifecycle: data exploration, train/validation/test split, etc.
- Sometimes, sampling is necessary:
  - We cannot access all possible data in the real world.
  - Using all data is unfeasible, costly, or otherwise impractical.
  - Accomplish a task faster and cheaper: experiment with a new model, explore data, etc.

There are two families of sampling:

- Nonprobability sampling.
- Random sampling.

# Nonprobability Sampling

- Generally, selecting data to train ML methods using this family of sampling methods is a bad idea, but some of them are popular.
- Convenience sampling
    - Select data based on their availability.
    - Popular and convenient: fast, inexpensive, practical.
    - Not scientific and does not offer guarantees.
- Snowball sampling
    - Future samples are selected based on existing samples.
    - Sampling in social media (or other) networks: select a base sample of accounts, then expand the sample by adding the accounts they follow, and so on.

# Nonprobability Sampling

- Judgement sampling
    - Experts decide what samples to include.
    - AKA: risk-based, SME, subjective, etc.
- Quota sampling
    - Select samples based on predefined and heuristic quotas.
    - Example: select 100 responses from all age groups without considering the proportional representation of age groups.

# Random Sampling

- Simple Random Sampling
    - All potential samples in the population have equal probabilities of being selected.
    - Advantage: Easy to implement.
    - Disadvantage: Rare categories of data may not appear in the selection: if a class appears in 0.01% of the data and we randomly select 1% of the population, we may not get a representation of this minority class.

# Random Sampling

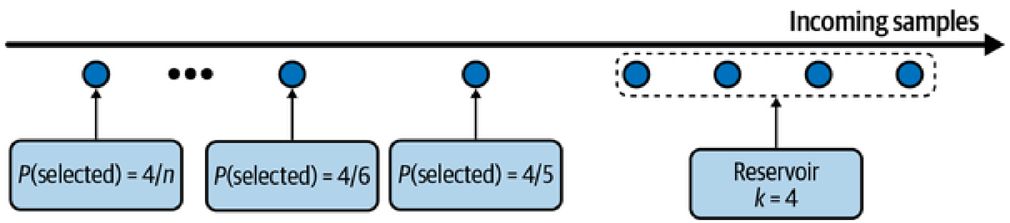
- Stratified Sampling
  - First, divide the population into groups we care about, then sample from each group separately.
  - Each group is called a *stratum* and this method is called *stratified sampling*.
  - Advantage: the distribution of groups in the population is reflected in the sample.
  - Particularly important for selecting training, validation, and test sets.
  - This method is only sometimes possible (multilabel cases, for example, may not be treated).

# Random Sampling

- Weighted Sampling
    - Each sample is given a weight, determining the probability of being selected.
    - This method allows us to leverage domain expertise.
    - Can be used to adjust samples that are coming from a different distribution than the original data:
        - Assume the data contains 25% red samples and 75% blue samples.
        - We know the actual distribution is closer to 50% red and 50% blue.
        - We can apply red weights that are three times higher than blue weights.

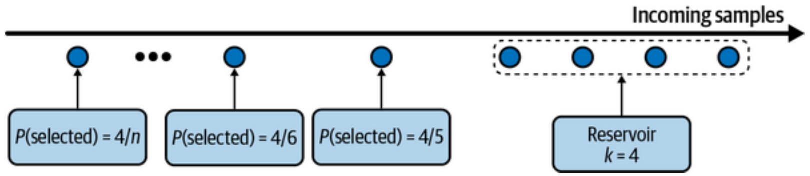
# Reservoir Sampling

- Ideal for streaming data with complex universe concept.
- Aim: unbiased sampling from Twitter feed.
- Goals:
    - Ensure equal probability for each tweet.
    - Ability to halt algorithm with correct sampling distribution.



# Reservoir Sampling

- Reservoir sampling:
  - Put the first k elements into the reservoir.
  - For each incoming nth element, generate a random number i such that 1 ≤ i ≤ n.
  - If 1 ≤ i ≤ k: replace the ith element in the reservoir with the nth element. Else, do nothing.
- Each incoming nth element has a k/n probability of being in the reservoir.



# Labeling

# Hand Labels

- Getting hand-labelled data takes a lot of work.
- It is expensive, particularly if subject matter expertise is required. For instance, compare:
    - Hand label a sentiment data set.
    - Hand label a medical diagnosis data set.
- It may be invasive: hand labelling data requires someone to see the data.
- Hand labelling is slow.

# Hand Labels

- Label ambiguity or label multiplicity occurs when multiple conflicting labels exist for a data instance.
- Label multiplicity may occur when labels are input by multiple annotators or data comes from different sources.
- Disagreements among annotators are common, particularly as the need for subject matter expertise increases.
- A potential solution is to have a clear problem definition and task guidance.

# Examples of Label Multiplicity

| Annotator | # entities | Annotation |
|---|---|---|
| 1 | 3 | [Darth Sidious], known simply as the Emperor, was a [Dark Lord of the Sith] who reigned over the galaxy as [Galactic Emperor of the First Galactic Empire]. |
| 2 | 6 | [Darth Sidious], known simply as the [Emperor], was a [Dark Lord] of the [Sith] who reigned over the galaxy as [Galactic Emperor] of the [First Galactic Empire]. |
| 3 | 4 | [Darth Sidious], known simply as the [Emperor], was a [Dark Lord of the Sith] who reigned over the galaxy as [Galactic |

# Natural Labels

- Natural ground truth labels or natural labels occur when the system can automatically evaluate or partially predict.
- Examples: time travelled on a particular route on Google Maps, stock return, etc.
- Natural labels are inexpensive to obtain and motivate many ML projects.

# Natural Labels

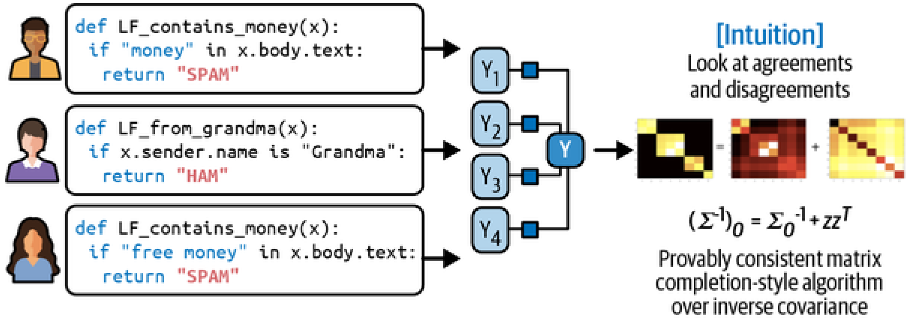
- Recommender systems are the prime example of natural labels: we will know if the recommendation was good, if it was acted on.
- Many tasks can be framed as recommendation tasks; for example, predicting an ad's clickthrough rate can be reframed as recommending the best ads.
- Natural labels that are inferred from user behaviours like clicks and ratings are known as behavioural labels.
- Behavioural labels can be:
    - Explicit labels are observed from user behaviour (click, upvote, rating, etc.)
    - Implicit labels are inferred by non-behaviour, for example, ads that are not clicked.
- Inferring an implicit label depends on the feedback loop length, which is the time between serving a prediction and the feedback on it provided.

# Handling the Lack of Labels

| Method | How | Ground truths required? |
|---|---|---|
| Weak supervision | Leverages (often noisy) heuristics to generate labels | No, but a small number of labels are recommended to guide the development of heuristics |
| Semi-supervision | Leverages structural assumptions to generate labels | Yes, a small number of initial labels as seeds to generate more labels |
| Transfer learning | Leverages models pretrained on another | No for zero-shot learning. Yes for fine-tuning, though the number of ground truths required is often much smaller than what would be needed |

# Weak Supervision

- Can we automate costly hand labeling?
- Weak supervision relies on codified heuristics using Labeling Functions (LF) such as keyword heuristics, regular expressions, database lookup, and outputs from other models.
- LFs are combined, reweighted, and denoised to generate labels.
- While ideally no hand labels are necessary, a few may be needed initially to assess LF accuracy.



```
def LF_contains_money(x):
 if "money" in x.body.text:
  return "SPAM"
```

```
def LF_from_grandma(x):
 if x.sender.name is "Grandma":
  return "HAM"
```

```
def LF_contains_money(x):
 if "free money" in x.body.text:
  return "SPAM"
```

$(\Sigma^{-1})_O = \Sigma_O^{-1} + zz^T$

Provably consistent matrix completion-style algorithm over inverse covariance

# Semi-Supervision

- Leverages structural assumptions to generate new labels based on a small set of initial labels.
- Useful when the number of labels is limited.
- Approach 1: self-training.
    - Train a model on an existing set of labelled data.
    - Make predictions for unlabeled samples; keep only the ones with high raw probability scores.
    - Train a new model on an expanded set of labels.

# Semi-Supervision

- Approach 2: similarity.
    - Assume that data samples that share similar characteristics share the same labels.
    - Similarity is established by more complex methods (clustering, k-nn, etc.)
- Approach 3: perturbation.
    - Assume that small perturbations to a sample do not change its label.
    - Apply small perturbations to your training instances to obtain new training instances.

# Class Imbalance

# What is Class Imbalance?

- Class imbalance occurs when one or more classes have significantly lower proportions in the data than other classes.
- The majority class dominates, but interest is generally in the minority class (e.g., default, fraud, or market crash).
- Models trained on imbalanced data will tend to be under-fitted; they will not be able to classify the minority class successfully.
- ML (particularly deep learning) works well when the class distribution is balanced. At the same time, performance decreases with class imbalance because:
  - There is insufficient signal for the model to learn to detect the minority class.
  - It is easier for a model to find a suboptimal solution by exploiting a simple heuristic instead of learning anything useful about the underlying pattern.
  - Asymmetric costs of error.

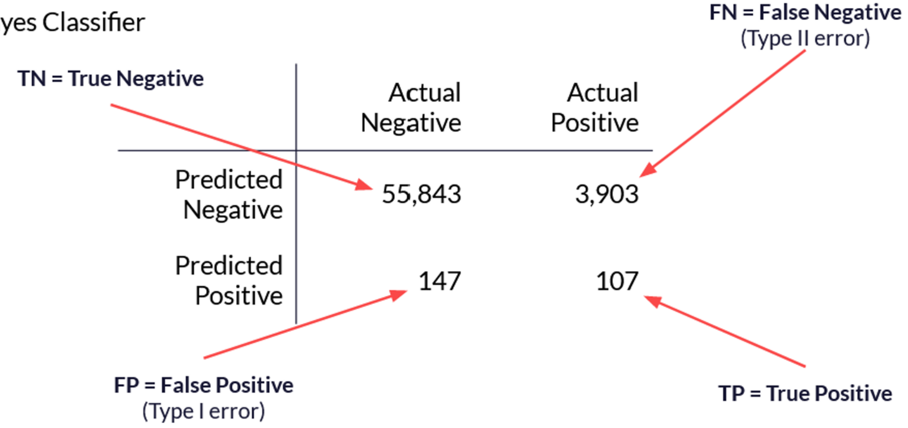
# What is Class Imbalance?

- Class imbalance is the norm in many subject domains.
- To handle class imbalance:
  - Choose the right performance metric.
  - Data-level methods: change the data distribution to reduce the imbalance.
  - Algorithm level methods: change the learning method to make it more robust to class imbalance.

# Performance Metrics

## Confusion Matrix

Naïve Bayes Classifier



|  | Actual Negative | Actual Positive |
|---|---|---|
| Predicted Negative | 55,843 | 3,903 |
| Predicted Positive | 147 | 107 |

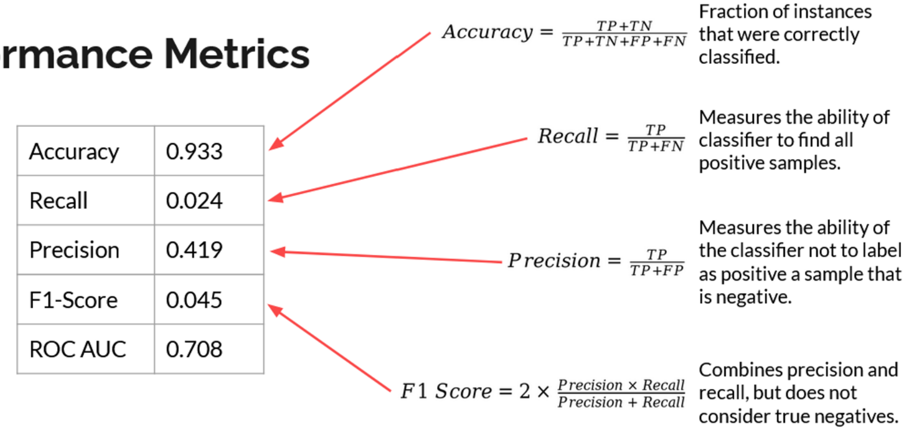
**TN = True Negative**

**FN = False Negative**
(Type II error)

**FP = False Positive**
(Type I error)

**TP = True Positive**

## Performance Metrics



|  |  |
|---|---|
| Accuracy | 0.933 |
| Recall | 0.024 |
| Precision | 0.419 |
| F1-Score | 0.045 |
| ROC AUC | 0.708 |

$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$ Fraction of instances that were correctly classified.

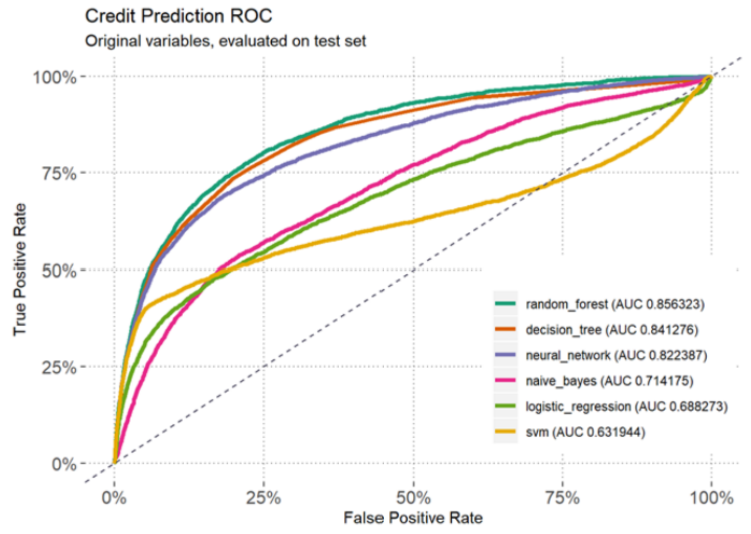
$Recall = \frac{TP}{TP+FN}$ Measures the ability of classifier to find all positive samples.

$Precision = \frac{TP}{TP+FP}$ Measures the ability of the classifier not to label as positive a sample that is negative.

$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$ Combines precision and recall, but does not consider true negatives.

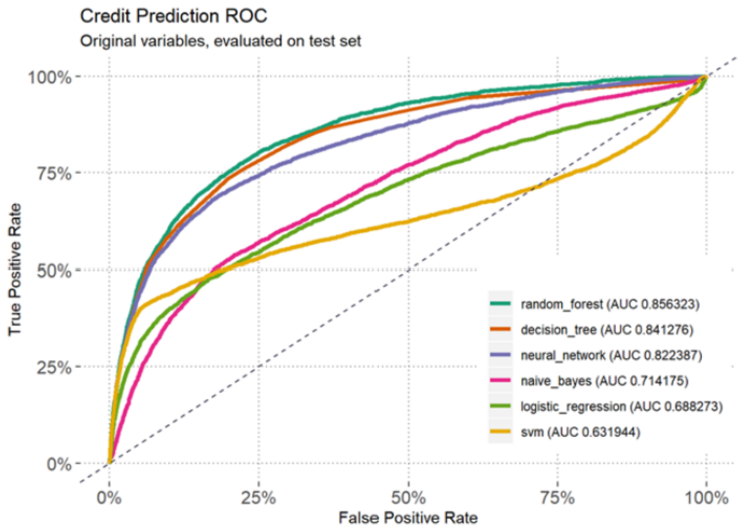
# Class Probabilities Carry Information

- Class probabilities offer more information about model predictions than the simple class value.
- Given class probabilities, one could decide to predict a class by comparing them to a threshold.
- A Receiver Operating Characteristic (ROC) curve shows the relationship between the True Positive Rate (TPR) and the False Positive Rate (FPR) for a variety of thresholds.



# Class Probabilities Carry Information

- A greater Area Under the ROC Curve (AUC ROC) indicates a better model: AUC ROC can be interpreted as the probability that the classifier ranks a randomly chosen positive instance above a randomly chosen negative one.
- AUC ROC measures the ranking order of a model's prediction: it is useful when costs are unavailable and class distributions are unknown.



# Cross-entropy, Negative Log-Loss, and Log-Likelihood

- Log loss or cross-entropy loss is a performance metric that quantifies the difference between predicted and actual probabilities.
- In a two-class setting, it is given by:

$$H(p,q) = -\sum_{i=1}^{n} \left(y_i log(\hat{y}_{\theta,i}) + (1-y_i) log(1-\hat{y}_{\theta,i})\right)$$

- Formulation is related to maximum likelihood: minimizing negative log-likelihood is the "same" as minimizing log loss.

# Cross-entropy, Negative Log-Loss, and Log-Likelihood

- Assume the actual value is 1.
- If the model is confident and correctly predicted 0.9, then

  > `Loss = −(1*log(0.9)) = 0.10536`

- If the model is unsure and predicted 0.5, then

  > `Loss = −(1*log(0.5)) = 0.6931`

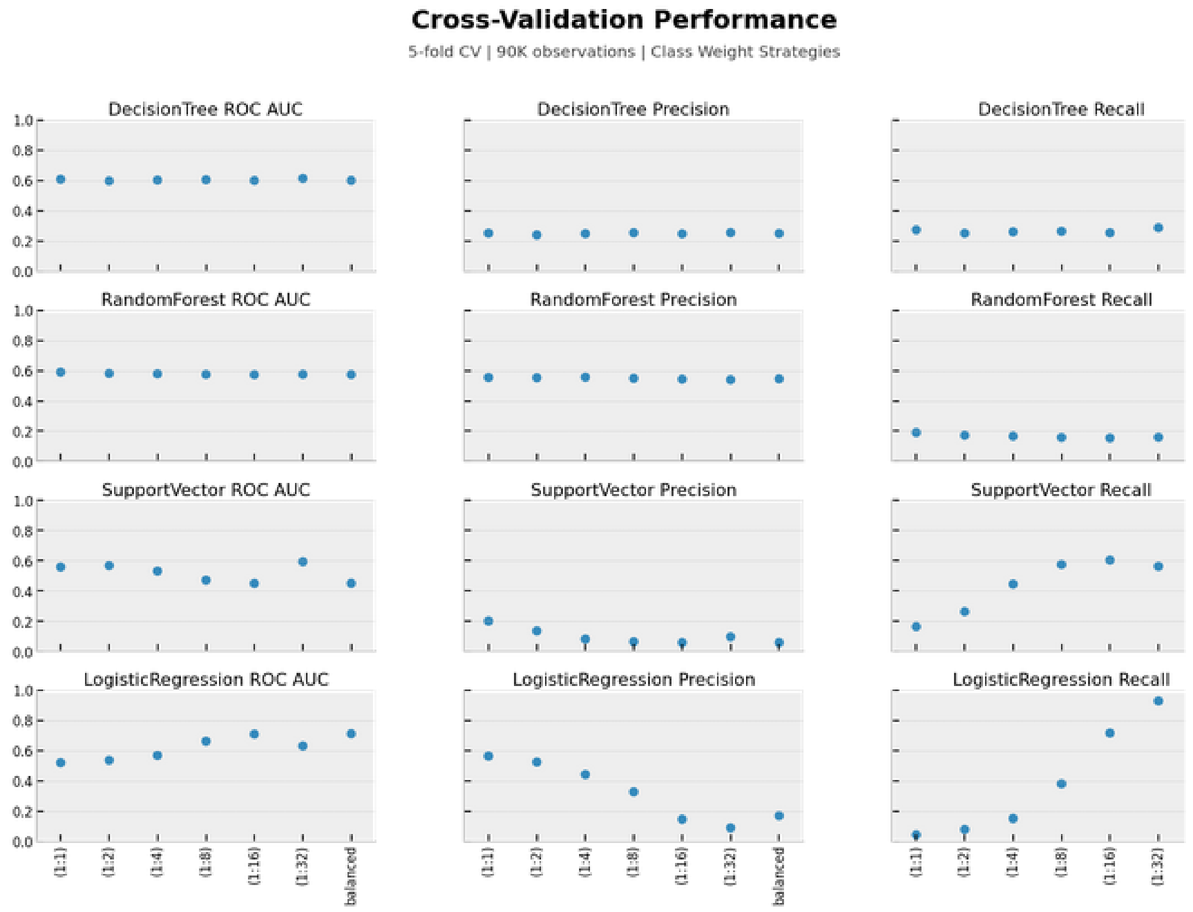
- If the model is confident but incorrectly predicted 0.1, then

  > `Loss = −(1*log(0.1)) = 2.0258`

# Class Weights

- Some models can optimize a cost or loss function that differentiates for specific types of errors.
- In some instances, one can assume that misclassifying minority events (false negatives) is more costly than incorrectly predicting non-events (false positives).
- Relative cost or class weights can be determined by
  - Consulting a Cost Specialist or Subject Matter Expert
  - Balance function

$$W_y = \frac{N_{samples}}{M_{classes} Ny}$$



Cross-Validation Performance

5-fold CV | 90K observations | Class Weight Strategies

DecisionTree ROC AUC

DecisionTree Precision

DecisionTree Recall

RandomForest ROC AUC

RandomForest Precision

RandomForest Recall

SupportVector ROC AUC

SupportVector Precision

SupportVector Recall

LogisticRegression ROC AUC

LogisticRegression Precision

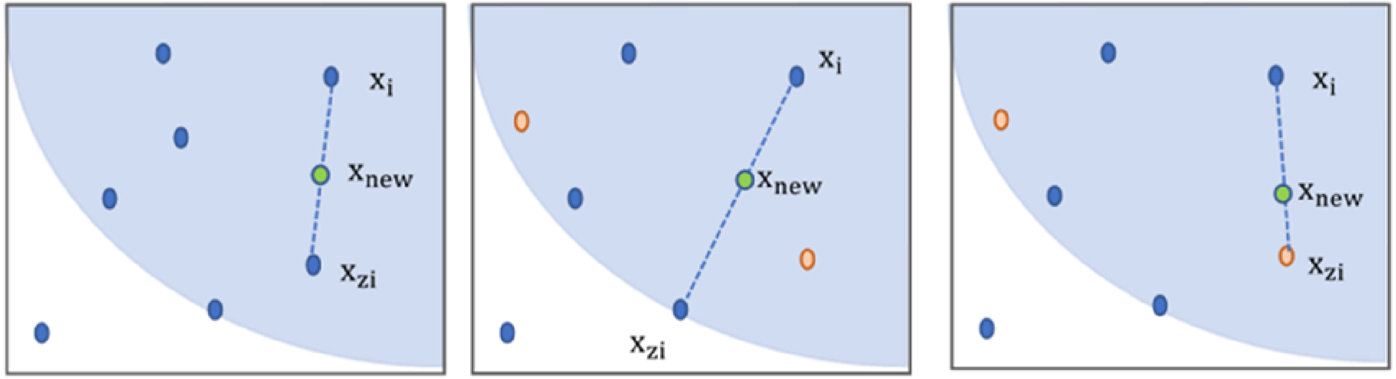
LogisticRegression Recall

# Class Weights and Performance

- Class weights (unequal costs) can affect model parameters and performance.
- Not every model will be equally affected by class weight strategies.

# SMOTE

- SMOTE: Synthetic Minority Oversampling TEchnique
  - Creates new instances based on random linear combinations of existing cases.
- ADASYN: Adaptive Synthetic Sampling Method
  - Similar to SMOTE, but new instances are generated based on density.
- With the availability of conformal prediction and advanced ML methods, synthetic oversampling is challenging to justify.



Source

# References

# References

- Agrawal, A. et al. "Cloudy with high chance of DBMS: A 10-year prediction for Enterprise-Grade ML." arXiv preprint arXiv:1909.00084 (2019).
- Huyen, Chip. "Designing machine learning systems." O'Reilly Media, Inc.(2022).