Life Sciences and
Facility Management
Institute of
Computational Life Sciences

Imbalanced data

Machine Learning

Norman Juchler

Learning objectives

- Understand what data imbalance is...
- ...and how to deal with it.

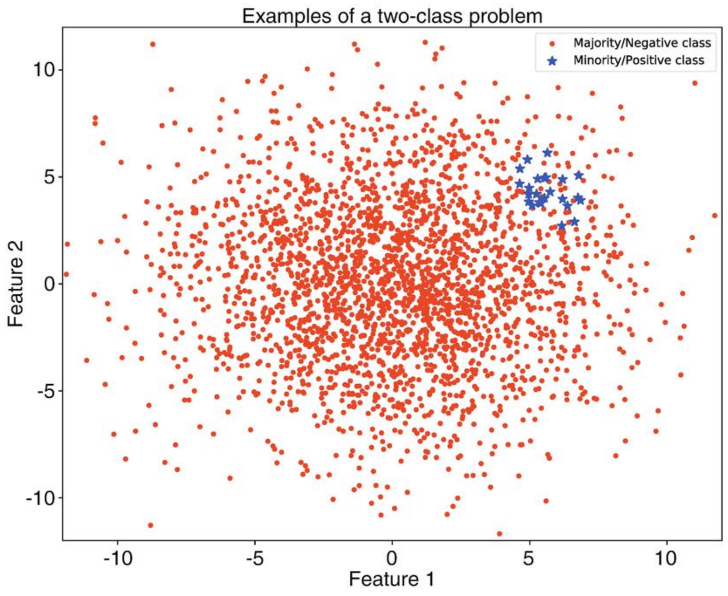
Image source: Adobe Stock / 948685604 / licensed

Imbalanced data

- **Definition**: Imbalanced data refers to a situation where certain categories, values, or groups in a dataset are underrepresented compared to others.

- **Examples**:
  - Credit card fraud
  - Natural disasters
  - Tumor cells
  - ...

- Many (if not most) real-world datasets exhibit imbalance
- Often, we are interested in the outlier (or rare) events



Consequences of data imbalance

- **Poor minority class performance**: It becomes hard for the model to learn about the minority class.
- **Bias toward the majority class**: Models tend to predict the majority class more frequently, neglecting the minority class.
- **Difficulty in learning patterns**: The model may struggle to learn meaningful patterns for the minority class due to insufficient representation.
- **Unreliable predictions in critical scenarios**: For problems like fraud detection or disease diagnosis, errors on the minority class can have severe consequences.
- **Misleading evaluation metrics**: Some metrics, like accuracy, can be unreliable with imbalanced data, masking poor performance on the minority class.
- **Challenges in optimization**: Gradient-based methods can be skewed by the imbalance, leading to suboptimal convergence.

Example: Fraud detection and accuracy

- Scenario:
  - Task: Develop a classification model to detect fraudulent transactions
  - Data set: Data from 100,000 transactions, of which 1,000 are fraudulent
- Candidate model: Naive model that predicts every transaction as legitimate!

> **Task**: Compute the accuracy for this naive model on the entire dataset.

Image source: Adobe Stock / 92627697 / licensed

Example: Fraud detection and accuracy

- Scenario:
  - Task: Develop a classification model to detect fraudulent transactions
  - Data set: Data from 100,000 transactions, of which 1,000 are fraudulent
- Candidate model: Naive model that predicts every transaction as legitimate!
- Performance assessment looks great:

$$Accuracy = \frac{Correct\ classifications}{All\ classifications} = \frac{99'000}{100'000} = 0.99$$

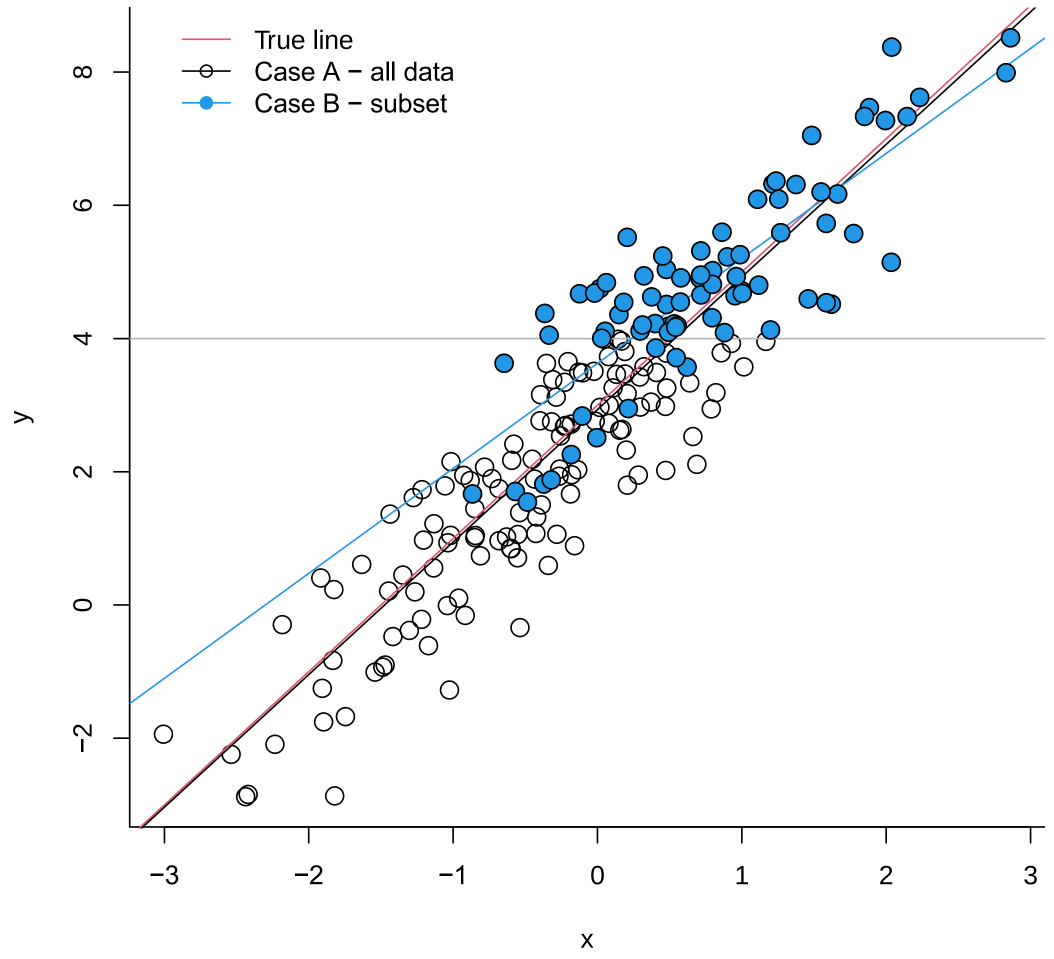
- Despite high accuracy, the model is completely useless for identifying fraud.
- **Solution**: Metrics like precision, recall, F1-score, or area under the ROC curve (AUC-ROC) are more meaningful

Image source: Adobe Stock / 92627697 / licensed

Data imbalance and regression problems

Two regression models that illustrate the effects of unbalanced data on regression. Case A: Regression model fitted to the entire data set. Case B: Model fitted to an unbalanced variant of data A, where only 10% of the points with a target value y<4 were included. Due to the imbalance of the data, the model becomes less accurate in the under-sampled region.

Such a situation can occur if, for example, a sensor is not able to measure reliably in certain value ranges.

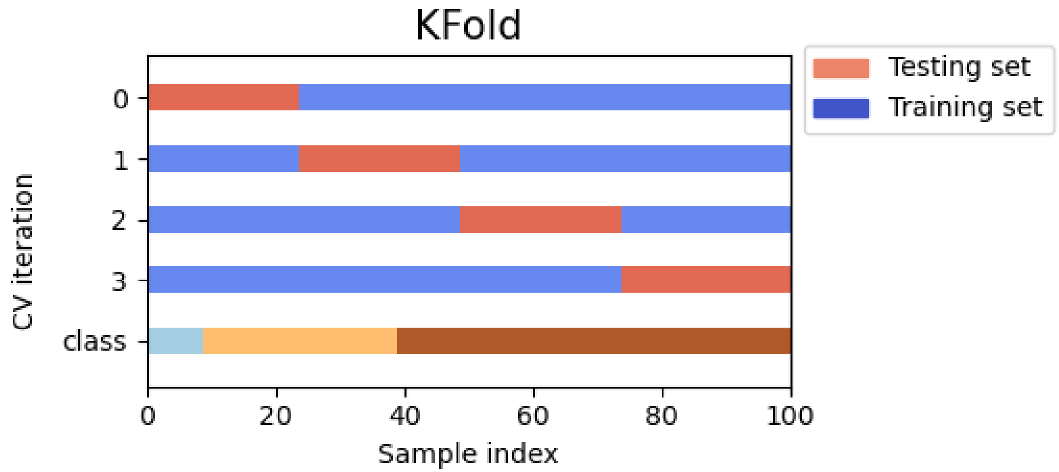


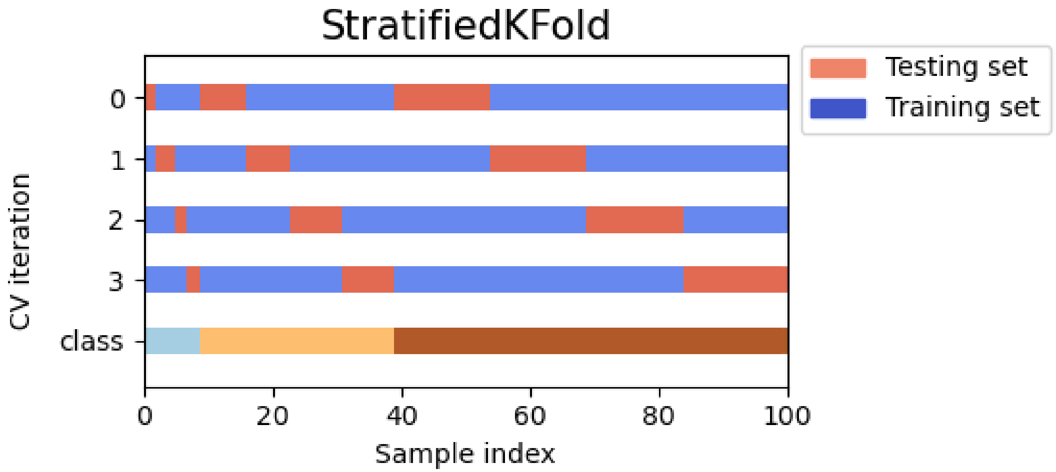
Methods

...for handling imbalanced data

Stratified sampling

- **Problem**: In imbalanced datasets, random sampling can lead to training and testing subsets that do not reflect the original class proportions.
- **Solution**: Use stratified sampling to ensure that relative class frequencies are approximately preserved in each train and validation fold.





Task: Try to understand these illustrations comparing the KFold and StratifiedKFold

Image source: scikit-learn

Stratified sampling

```python
import numpy as np
from sklearn.model_selection import KFold, StratifiedKFold

def count_classes(y):
    values, counts = np.unique(y, return_counts=True)
    return dict(zip(values, counts))

X, y = np.ones((50, 1)), np.hstack(([0] * 45, [1] * 5))
print("Summary of y:", count_classes(y))

# KFold does not respect the balance of classes
print("\nKFold:")
kf = KFold(n_splits=3)
for train, test in kf.split(X):
    print('train: %-13s | test - %-13s' % (count_classes(y[train]),
                                           count_classes(y[test])))

# StratifiedKFold respects the balance of classes
print("\nStratifiedKFold:")
skf = StratifiedKFold(n_splits=3)
for train, test in skf.split(X, y):
    print('train: %-13s | test - %-13s' % (count_classes(y[train]),
                                           count_classes(y[test])))
```

Output

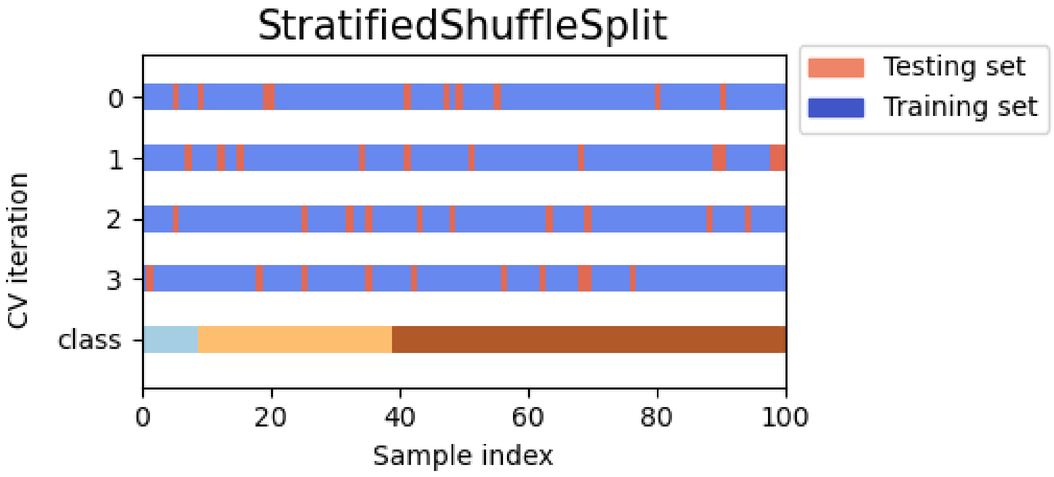
```
Summary of y: {0: 45, 1: 5}

KFold:
train: {0: 28, 1: 5} | test - {0: 17}
train: {0: 28, 1: 5} | test - {0: 17}
train: {0: 34}       | test - {0: 11, 1: 5}

StratifiedKFold:
train: {0: 30, 1: 3} | test - {0: 15, 1: 2}
train: {0: 30, 1: 3} | test - {0: 15, 1: 2}
train: {0: 30, 1: 4} | test - {0: 15, 1: 1}
```

Stratified sampling

- Related: StratifiedShuffleSplit...
- Goal: Create train-test splits that preserve the class ratios.
- (Related: One can set parameter stratify=True in train_test_split())



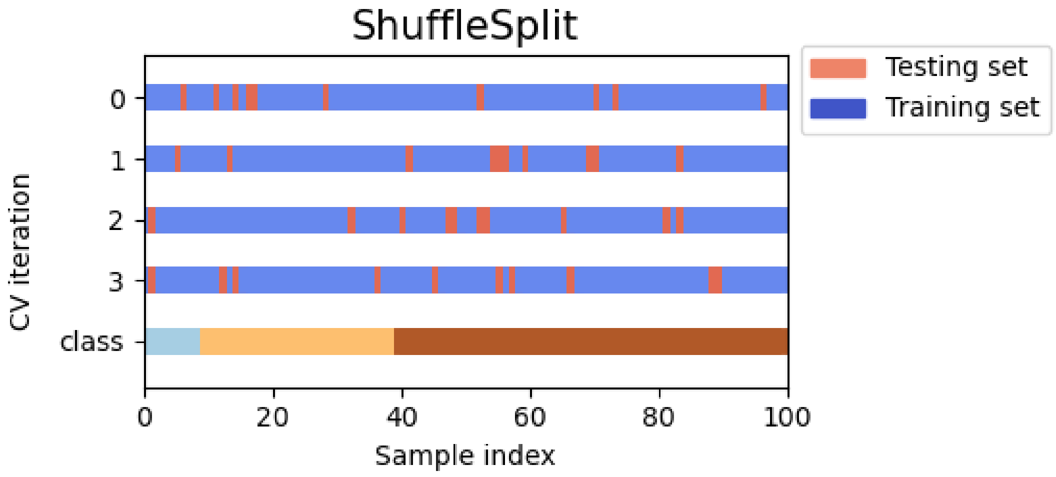
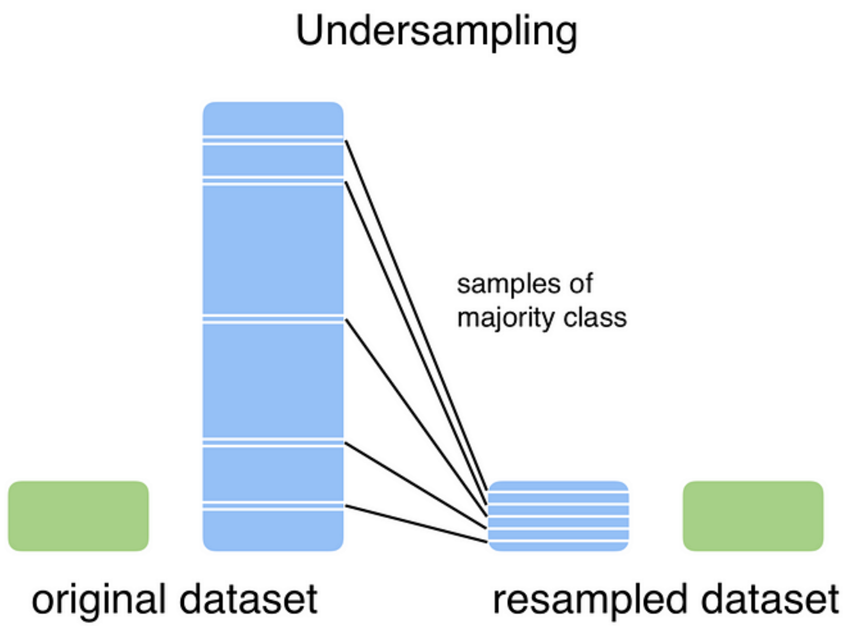


Image source: scikit-learn

Undersampling

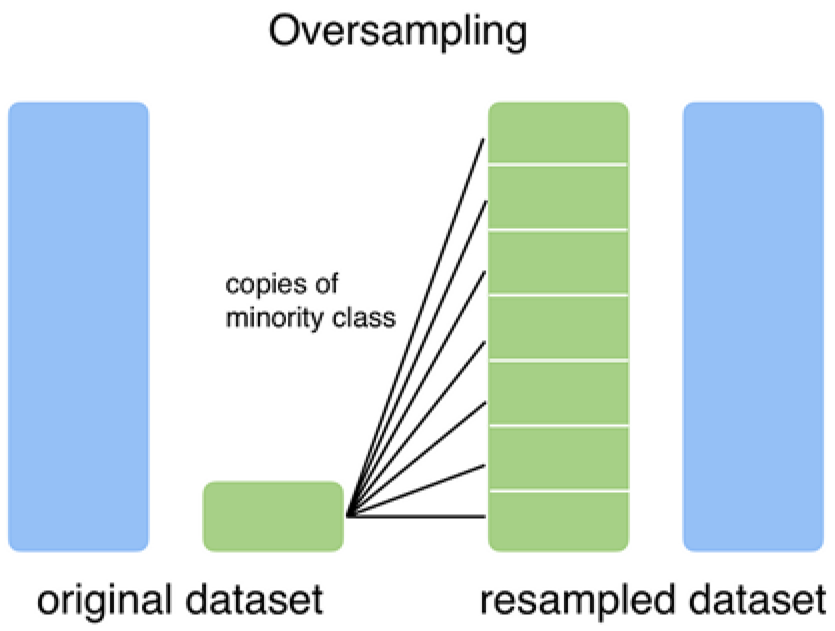
- Approach: Use only a fraction of the majority class samples



- Advantage: A smaller dataset means faster training
- Problem: Throwing away data removes information that could be useful

Oversampling

- Approach: Sample the minority samples with replacement for the training set.



- Advantage: No information is lost
- Problem: The minority class distribution is biased towards a few, potentially unrepresentative, samples.

Class weighting

- **Idea**: Apply different weights to samples from the minority and majority classes during the training process of a model.
- **Effect**: Class weights adjust the loss function so that misclassifying samples from underrepresented (minority) classes incurs a higher penalty
- **Common choice** for the weights is (per class)

$$w_c = \frac{Total\ samples}{Number\ of\ samples \times number\ of\ classes}$$

- Example: Logistic regression

```
lr = LogisticRegression(solver="newton-cg",
                        class_weight="balanced")
```

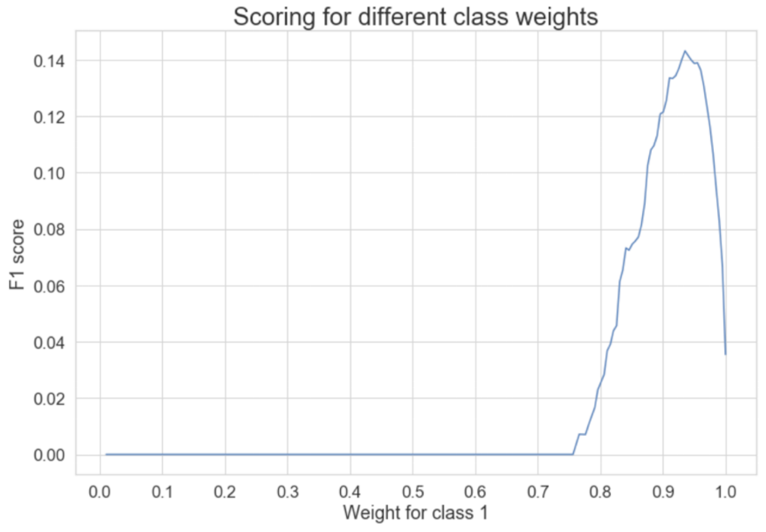


Illustration of model prediction scores as a function of different values of class weights

Image source: Analytics Vidhya

SMOTE

- **Synthetic Minority Oversampling Technique** (SMOTE): Synthesize new samples for the minority class.

- **Algorithm**:
  - select a minority class instance at random
  - find its k nearest minority class neighbors and select one at random
  - draw a line between these two points in the feature space
  - draw a new sample at a point along that line.

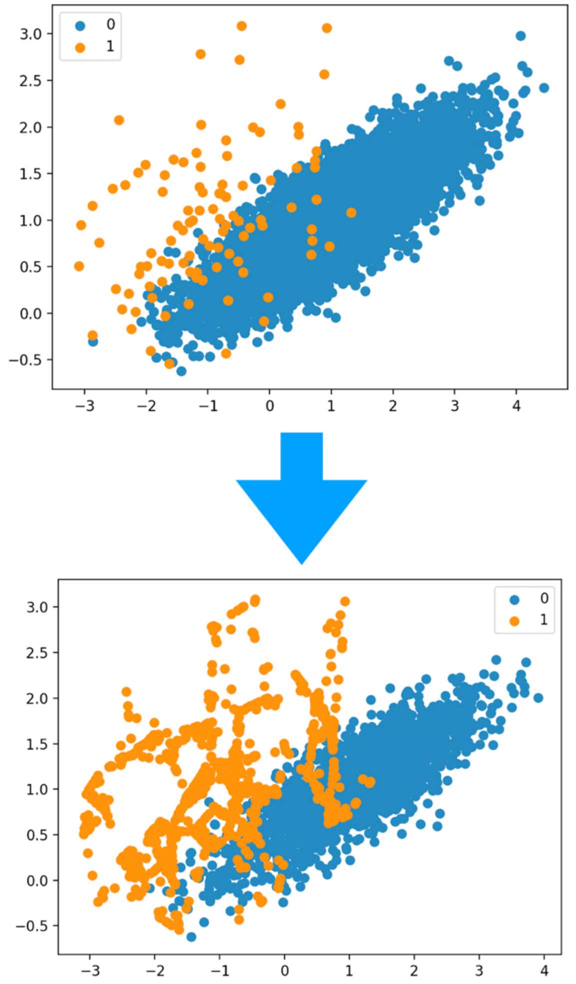


Image source: The machine learning mastery

Evaluation of imbalanced datasets

- Use imblearn's `classification_report_imbalanced()`
- Pay special attention to the performance of the minority class(es)

```
import numpy as np
from imblearn.metrics import classification_report_imbalanced

y_true = [0, 1, 2, 2, 2]
y_pred = [0, 0, 2, 2, 1]
target_names = ['class 0', 'class 1', 'class 2']
print(classification_report_imbalanced(y_true, y_pred,
                                       target_names=target_names))
```

Output

	pre	rec	spe	f1	geo	iba	sup
class 0	0.50	1.00	0.75	0.67	0.87	0.77	1
class 1	0.00	0.00	0.75	0.00	0.00	0.00	1
class 2	1.00	0.67	1.00	0.80	0.82	0.64	3
avg / total	0.70	0.60	0.90	0.61	0.66	0.54	5

imbalanced-learn (imblearn)

- An extension for scikit-learn: https://imbalanced-learn.org
- Designed specifically to handle imbalanced datasets