KAZAKH-BRITISH TECHNICAL UNIVERSITY

Introduction to Machine Learning
Week 10

Olivier JAYLET

School of Information Technology and Engineering

Classification metrics

Classification metrics are used to evaluate the performance of classification models, which are machine learning models that predict categorical labels or classes for input data.

Common classification metrics include accuracy, precision, recall, and F1 score, each providing a unique insight into the model's performance, such as its ability to correctly predict the positive class or avoid false positives.

The choice of classification metric often depends on the specific problem being addressed, as some metrics may be more suitable for imbalanced datasets or specific business objectives

Confusion matrix

		Actual	
		Positive	**Negative**
Predicted	**Positive**	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

This matrix gives four different informations :

- TP : Event predicted **True** while it was actually **True**.
- TN : Event predicted **False** while it was actually **False**.
- FN : Event predicted **False** while it was actually **True**.
- FP : Event predicted **True** while it was actually **False**.

Accuracy

The most common metric for binary and multiclass classification which shows the fraction of correct predictions:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{1}$$

$$\text{Accuracy} = \frac{\sum TP + TN}{\sum TP + TN + FP + FN} \tag{2}$$

$$\text{acc}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}[y_i = \hat{y}_i]. \tag{3}$$

The missclassification rate (or error rate) is a similar metric and equals to $1 - \text{acc}$.

Recall

The recall compute the ratio of TP over the TP and the FN populations.

$$\text{Recall} = \frac{\sum TP}{\sum TP + FN} \tag{4}$$

A high recall rate indicates that we correctly identified most positive observations, minimizing the number of positive cases predicted as negative.

Lets say we have a model to predict whether a student is depressed. In this case, it might be more interesting to focus on the recall. In your opinion, why ?

Precision

Precision is the ratio of TP to the total of TP and FP. It assesses the proportion of predicted positive instances that are truly positive.

$$\text{Precision} = \frac{\sum TP}{\sum TP + FP} \tag{5}$$

A high precision score indicates that the model is accurate in its positive predictions, resulting in a lower number of FP.

Lets say we have a model to predict whether your baggage will be mishandled by the airport. In this case, it might be more interesting to focus on the precision. In your opinion, why ?

F1-Score

The F1-score is the harmonic mean of precision and recall. It provides a balanced measure by considering both false positives and false negatives, making it especially useful when the dataset is imbalanced.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6}$$

A high F1-score indicates that the model achieves a good balance between precision and recall, excelling at both identifying positive cases and minimizing incorrect positive predictions.

ROC-AUC

To map predicted probabilities to binary classes, you need to choose a threshold (classification rule). However, this threshold can vary according to your problem. The ROC curve evaluates the model's performance across all possible thresholds (from 0 to 1).

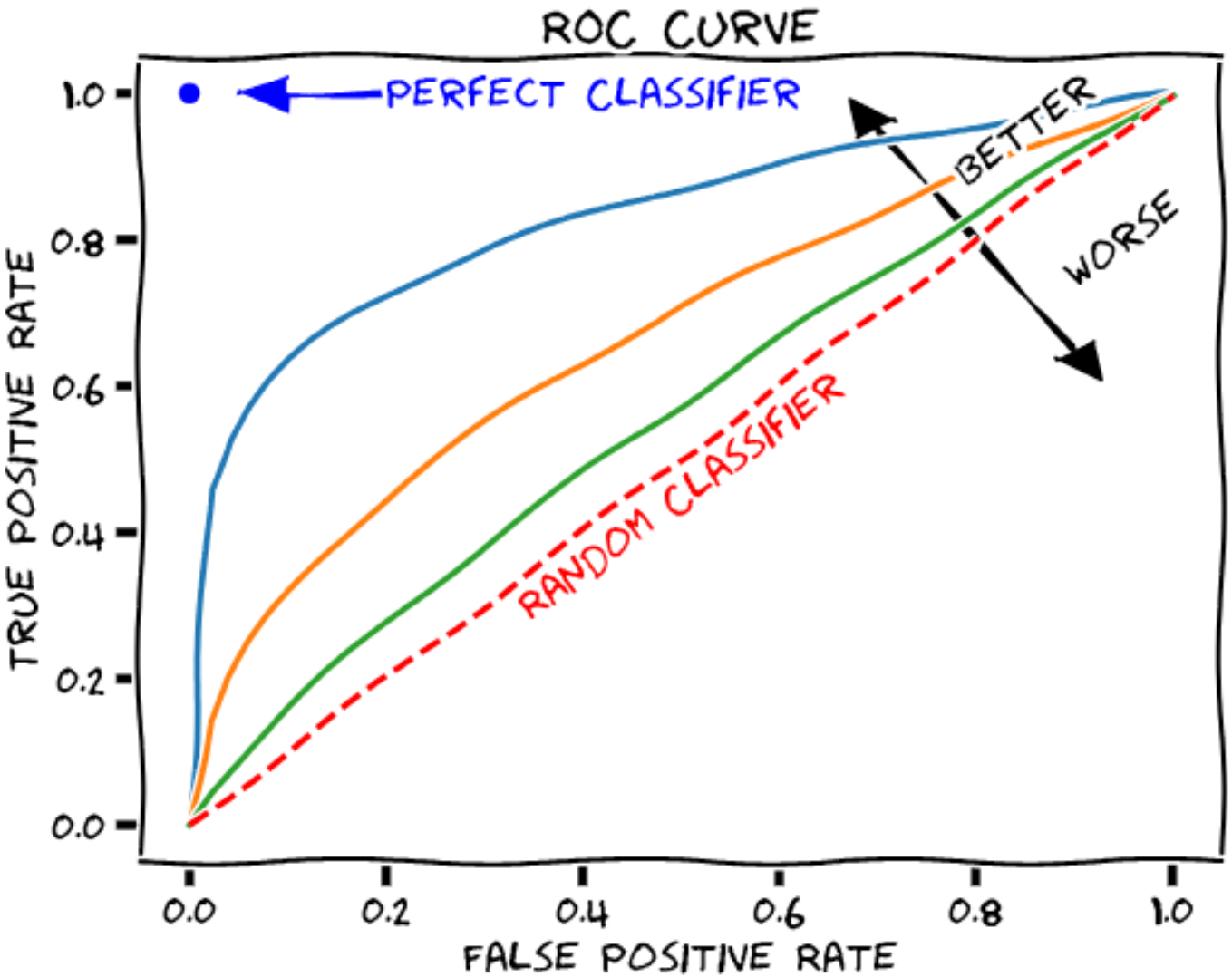
ROC-AUC

The ROC-AUC curve is built by plotting the True Positive Rate against the False Positive Rate at every possible thresholds.

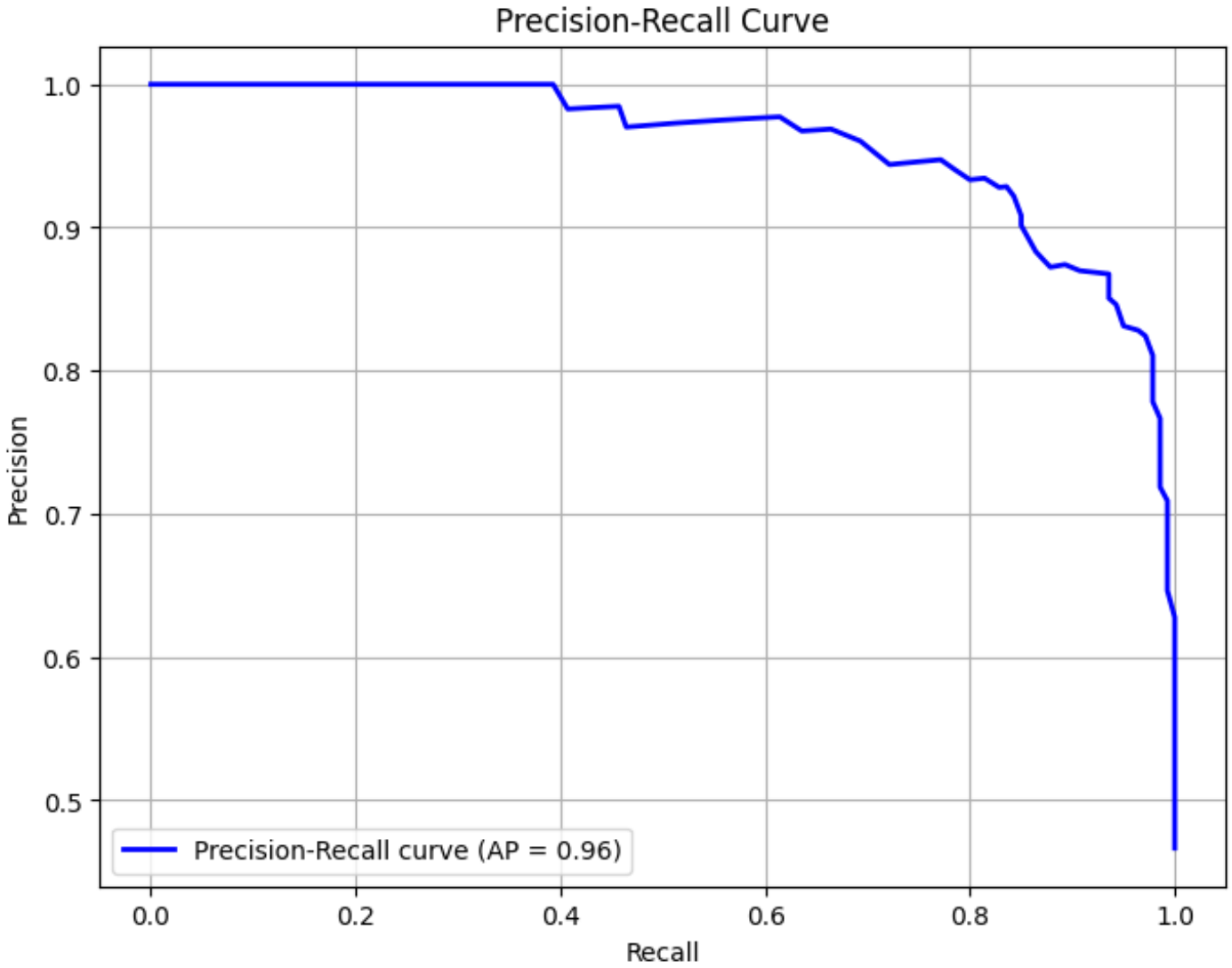
- $TPR = TP/(TP + FN)$
- $FPR = FP/(FP + TN)$

1. Start with a threshold of 0, which means all predictions are classified as positive. In this case:
   - TPR = 1 (all true positives are identified),
   - FPR = 1 (all true negatives are misclassified as positives).
2. Gradually increase the threshold (e.g., 0.1, 0.2, ..., 1.0).
3. For each threshold, calculate the TPR and FPR values.
4. At a threshold of 1, everything is classified as negative, so TPR = 0 and FPR = 0.

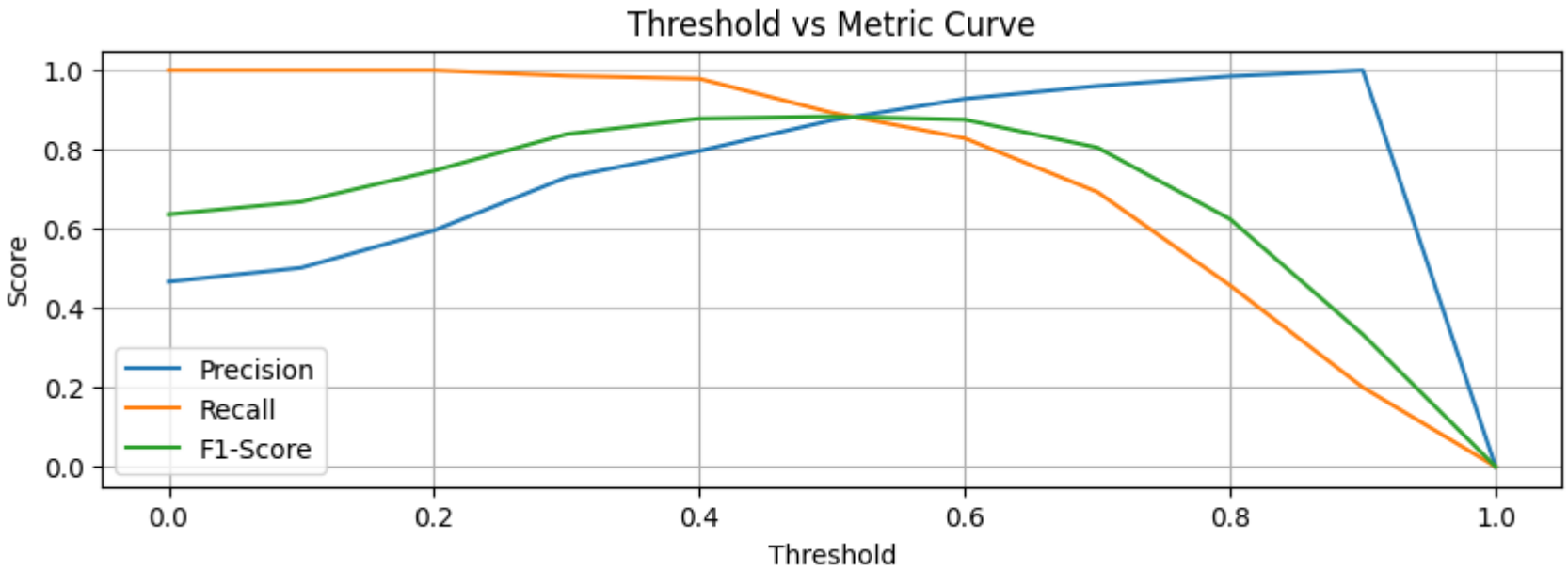
ROC-AUC Curve



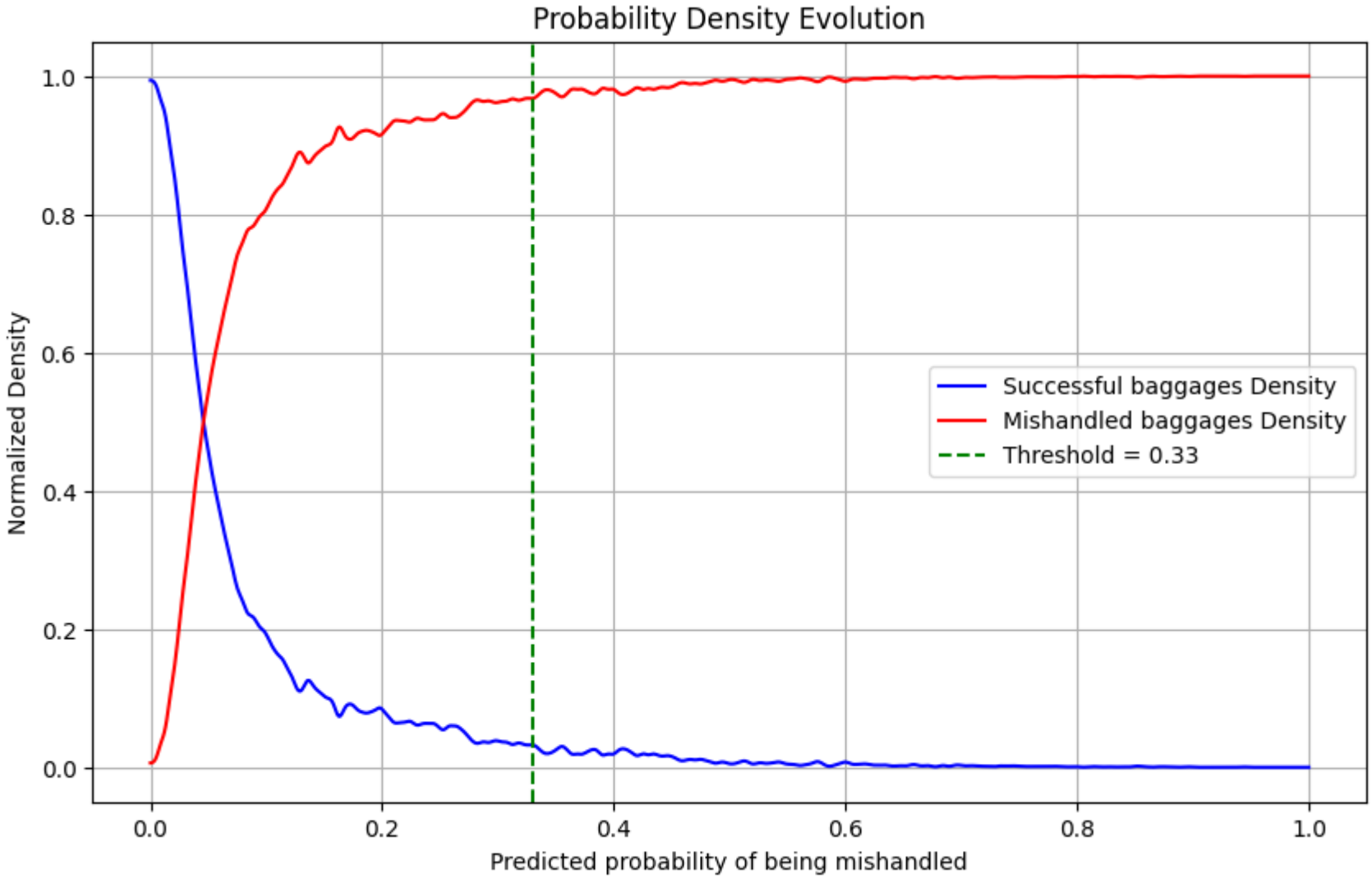
Precision-Recall Curve



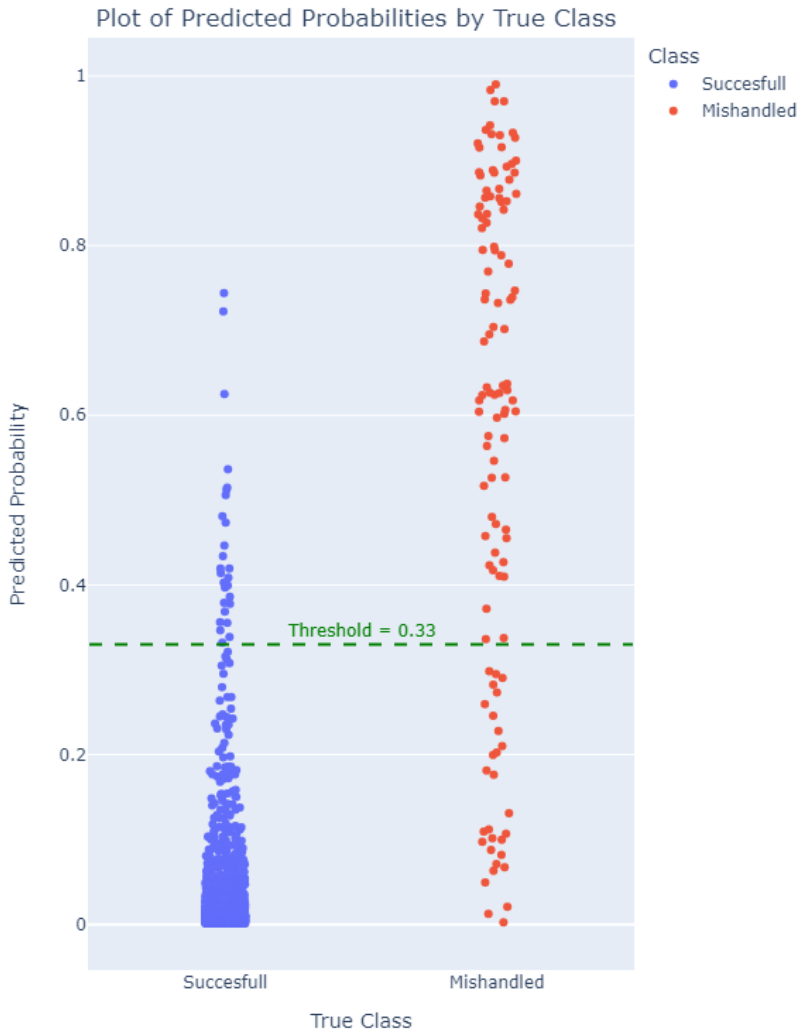
Threshold Vs. Metrics curves



Probability density



Probability distribution



Cross-Validation

Cross-Validation : Technique used for assessing the performance and generalization of a predictive model.

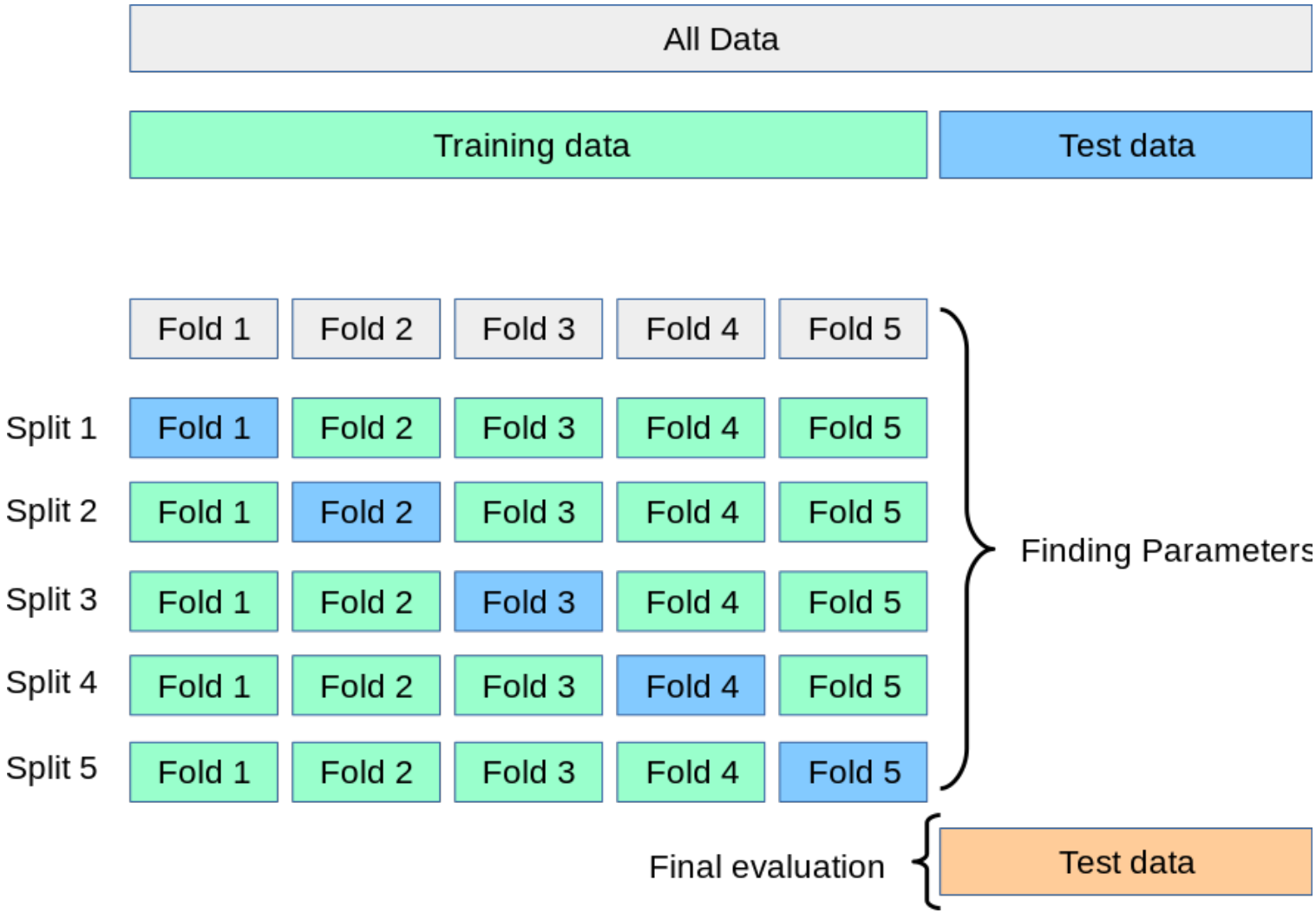
- Split the dataset into multiple subsets
- Train and test the model on different combinations of these subsets
- Aggregate the results to get a more comprehensive performance evaluation

Goal: Obtain a reliable estimate of a model's performance and ability to generalize to unseen data.

Different types of Cross-Validation

- **K-Fold Cross-Validation**: Divide the dataset into k subsets, train on $k-1$ subsets, and test on the remaining subset. Repeat for all k subsets.
- **Stratified K-Fold Cross-Validation**: Similar to k-fold cross-validation, but ensures that the class distribution is preserved in each subset.

K-Fold Cross Validation



```
import numpy as np
from sklearn.model_selection import KFold

X = np.arange(27).reshape(9, 3)
y = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
kf = KFold(n_splits=3, shuffle=True)

for train_index, valid_index in kf.split(X):
   print("TRAIN:", train_index, "VALID:", test_index)
```

Output :

```
TRAIN: [0 1 3 6 7 8] VALID: [2 4 5]
TRAIN: [0 1 2 3 4 5] VALID: [6 7 8]
TRAIN: [2 4 5 6 7 8] VALID: [0 1 3]
```

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(max_iter=5000)
scores = cross_val_score(clf, X_train, y_train, cv=5,
    scoring='f1')
print("Cross validation scores:", scores)
```

Output :

```
Cross validation scores: [0.73529412 0.66666667 0.672
    0.75590551 0.76335878]
```

Pro & cons of KFold CV

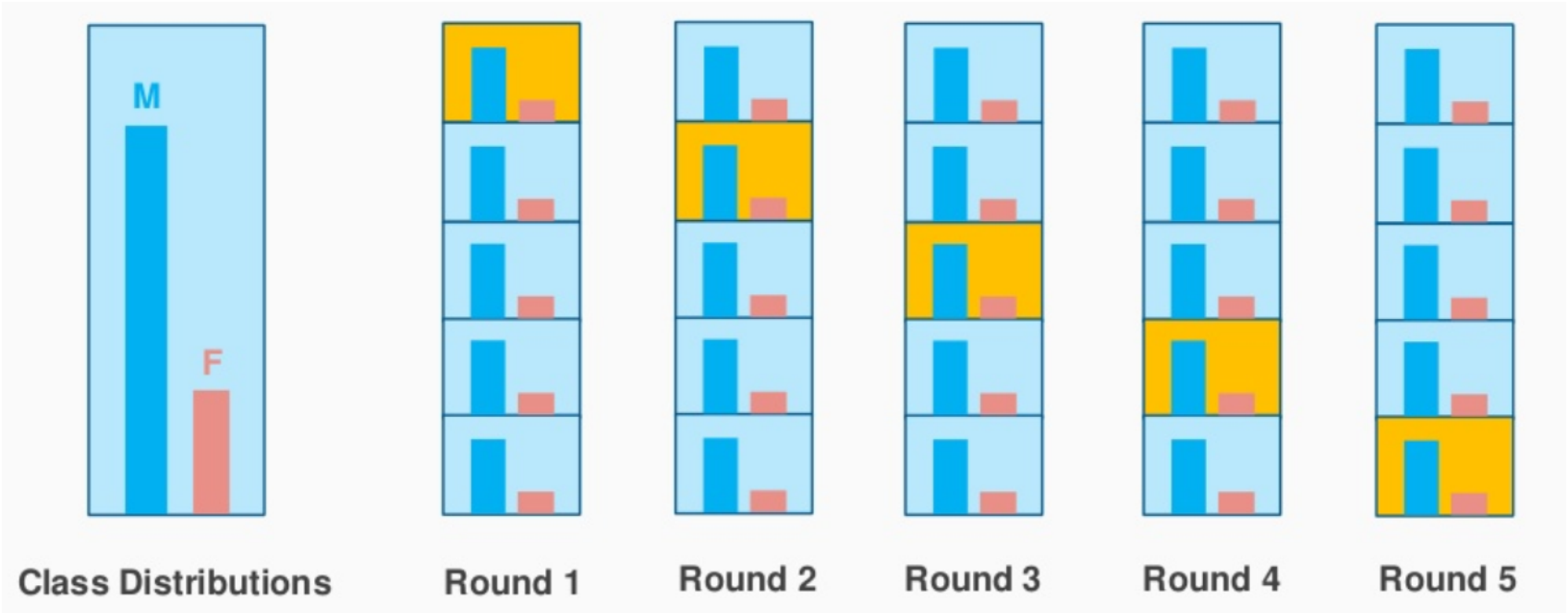
Pros :

- Reduces overfitting
- Improved model evaluation
- Handles small datasets

Cons :

- May not preserve class balance
- Sensitive to hyperparameter tuning
- May not capture temporal dependencies

Stratified K-Fold



```
from sklearn.model_selection import StratifiedKFold,
    cross_val_score
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(max_iter=5000)

n_folds = 5
skf = StratifiedKFold(n_splits=n_folds, shuffle=True,
    random_state=42)

scores = cross_val_score(clf, X_train, y_train, cv=skf,
    scoring='f1')
print("StratifiedKfold cross validation scores:", scores)
```

Output :

```
StratifiedKfold cross validation scores: [0.73529412
    0.66666667 0.672 0.75590551 0.76335878])
```

Pro & cons of Stratified KFold CV

Pros :

- Preserves class balance
- Useful for classification problems

Cons :

- Computationally more expensive
- Limited to classification problems

What are the hyper-parameters ?

Definition : In machine learning, a hyperparameter is a parameter whose value is used to control the learning process.

Example of hyperparameters (SKLearn Library)

- max_iter : The maximum number of passes over the training data (Epochs)
- tol : The stopping criterion (Tolerance).
- learning_rate : The learning rate schedule ('Constant', 'Adaptive', 'optimal' etc.)
- eta0 : initial learning rate value
- multi_class : 'OVR' or 'multinomial' (Softmax)
- Warm_start : reuse the solution of the previous call to fit as initialization. This parameter allows to setup the initial coefficients differently. (SKlearn isn't well optimized for this purpose)

Problem of hyperparameters

Who can answer the question : ”What is the best value of the learning rate for whatever problem ?”

Even if there are conventional range of values, we cant know which learning rate is perfect, mainly because it always change according to the data and the problem we try to solve.

This question become more complex when it comes to combination of hyperparameters.

However, we can try to answer this question by optimizing our model according to hyperparameters.

Hyperparameters tuning

The optimization encounter two main issues :

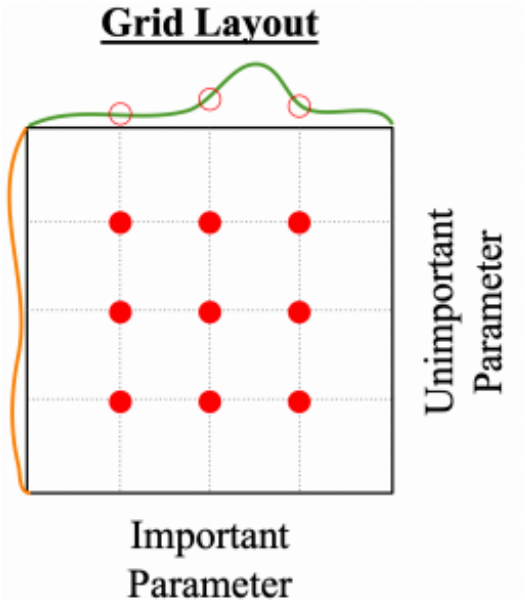
- High dimensionality: The number of hyperparameters can be large, making it difficult to search for the optimal combination.
- Computational cost: Evaluating different hyperparameters can be computationally expensive, making it difficult to search for the optimal combination.

Methods to find the optimal set of hyperparameters :

- Grid search
- Random Search
- (There exists other more advanced methods)

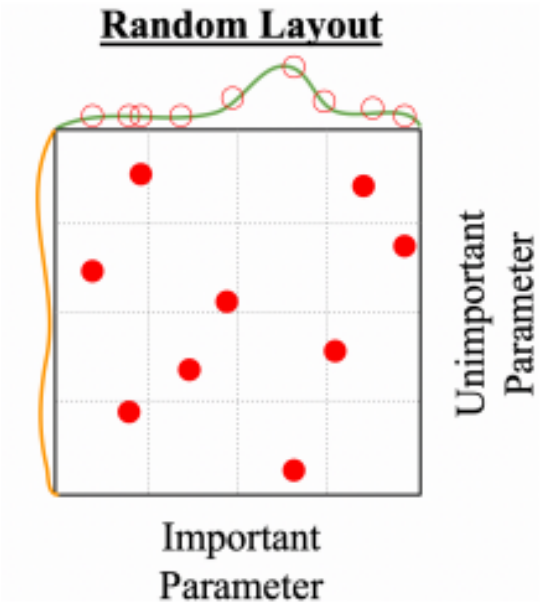
Grid Search Optimization

- Select a set of values for each hyperparameter
- For example,
  - Learning rate: 1×10^{-4}, 1×10^{-3}, 1×10^{-2}, 1×10^{-1}
  - Multiclass : 'OVR', 'multinomial'
  - max_iter : 100, 500, 1000
- Evaluate all possible values in a grid search.
- Exponential in number of hyperparameters.



Random Search optimization

- Select some ranges of values along which we want to search.
- For example,
  - Learning rate: $[1 \times 10^{-4}, 1 \times 10^{-1}]$
  - Multiclass : ['OVR', 'multinomial']
  - max_iter : [100, 1000]
- Select a random value for each of those types of parameters that fall within that range.
- More efficient strategy.



```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'alpha': [0.001, 0.01, 0.1],
    'eta0': [0.001, 0.01, 0.1],
    'max_iter': [100, 500, 1000]
}
sgdreg = SGDRegressor()
grid_search = GridSearchCV(sgdreg, param_grid, cv=5)

grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_

# Train a new model with the best hyperparameters and evaluate
    it
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
```

```
from sklearn.model_selection import RandomizedSearchCV

param_grid = {
    'alpha': np.logspace(-3, 0, 10),
    'eta0': np.logspace(-3, 0, 10),
    'max_iter': np.linspace(100, 1000, 10)
}
sgdreg = SGDRegressor()
random_search = RandomizedSearchCV(sgdreg, param_grid, cv=5,
    n_iter=100)

random_search.fit(X_train, y_train)

best_params = random_search.best_params_
best_score = random_search.best_score_

# Train a new model with the best hyperparameters and evaluate
    it
best_model = random_search.best_estimator_
y_pred = best_model.predict(X_test)
```