VEILLE SUR CROSS VALIDATION(VALIDATON CROISEE)

YAO KOUAME ARNAUD,
ETUDIANT A IGS EN DATA IA.

SOMMAIRE

I-INTRODUCTION

Chaque fois que nous construisons un modèle d'apprentissage automatique, nous l'alimentons avec des données initiales pour entraîner le modèle. Et puis nous alimentons des données inconnues (données de test) pour comprendre les performances du modèle et généralisées sur des données invisibles. Si le modèle fonctionne bien sur les données invisibles, il est cohérent et est capable de prédire avec une bonne précision sur une large gamme de données d'entrée ; alors ce modèle est stable.

Mais ce n'est pas toujours le cas ! Les modèles d'apprentissage automatique ne sont pas toujours stables et nous devons évaluer la stabilité du modèle d'apprentissage automatique. C'est là qu'intervient la validation croisée

« En termes simples, la validation croisée est une technique utilisée pour évaluer les performances de nos modèles d'apprentissage automatique sur des données invisibles »

Selon Wikipedia, la validation croisée est le processus d'évaluation de la façon dont les résultats d'une analyse statistique se généraliseront à un ensemble de données indépendant.

Il existe de nombreuses façons d'effectuer une validation croisée et nous allons découvrir 4 méthodes dans notr exposé.

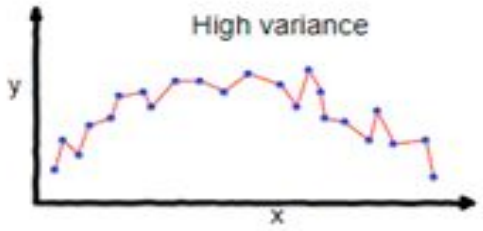
Comprenons d'abord la nécessité d'une validation croisée !

II-POURQUOI AVONS-NOUS BESOINS D'UNE VALIDATION CROISEE ?

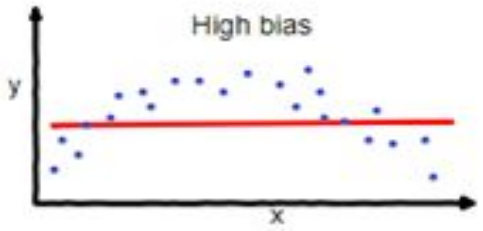
Supposons que vous construisiez un modèle d'apprentissage automatique pour résoudre un problème et que vous ayez entraîné le modèle sur un ensemble de données donné. Lorsque vous vérifiez la précision du modèle sur les données d'entraînement, elle est proche de 95%. Cela signifie-t-il que votre modèle s'est très bien entraîné et qu'il s'agit du meilleur modèle en raison de sa grande précision ?

Non ce n'est pas! Parce que votre modèle est entraîné sur les données données, il connaît bien les données, a capturé même les variations infimes (bruit) et s'est très bien généralisé sur les données données. Si vous exposez le modèle à des données totalement nouvelles et invisibles, il peut ne pas prédire avec la même précision et il peut échouer à généraliser sur les nouvelles données. Ce problème est appelé sur-ajustement.

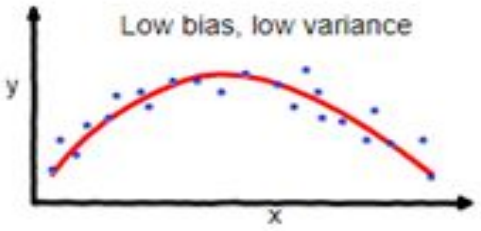
Parfois, le modèle ne s'entraîne pas bien sur l'ensemble d'entraînement car il n'est pas capable de trouver des modèles. Dans ce cas, il ne fonctionnerait pas non plus correctement sur l'ensemble de test. Ce problème est appelé sous-ajustement.



overfitting



underfitting



Good balance

Source de l'image: fireblazeaischool.in

Pour surmonter les problèmes de sur-ajustement, nous utilisons une technique appelée validation croisée.

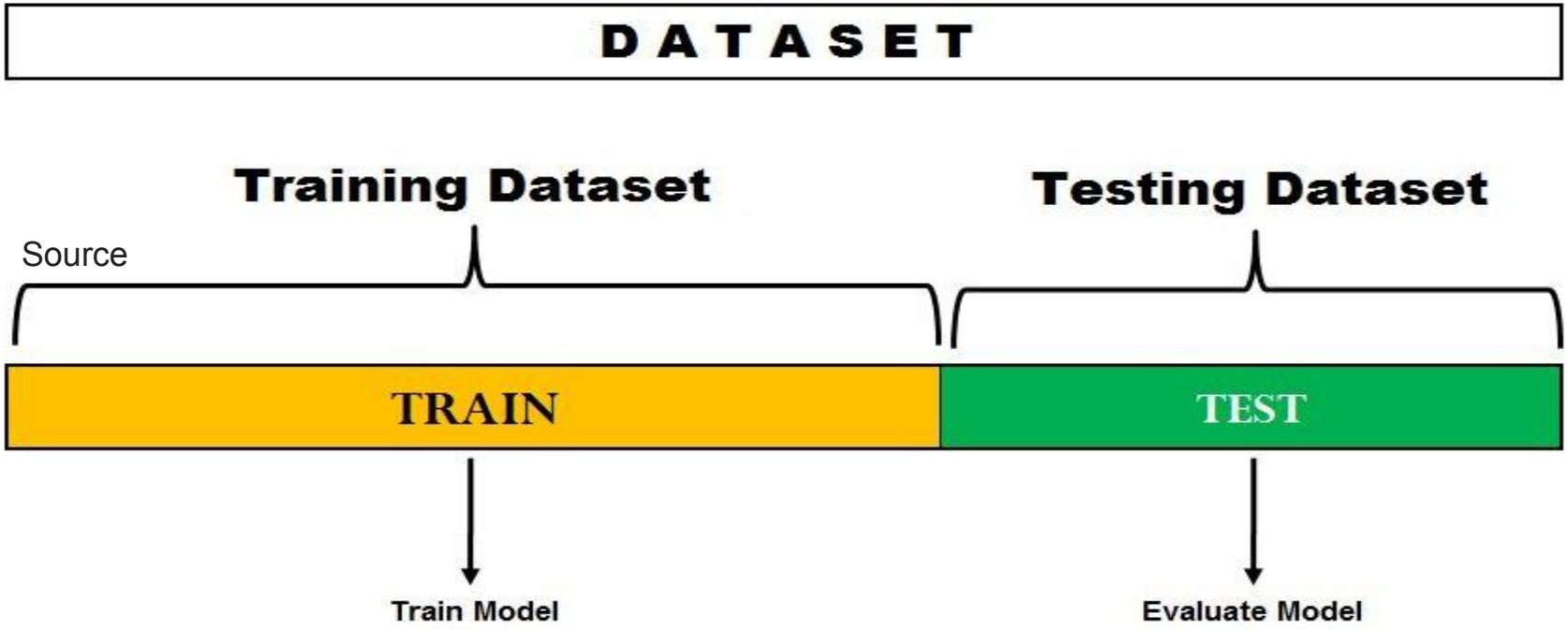
La validation croisée est une technique de rééchantillonnage avec l'idée fondamentale de diviser l'ensemble de données en 2 parties : les données d'entraînement et les données de test. Les données d'entraînement sont utilisées pour entraîner le modèle et les données de test invisibles sont utilisées pour la prédiction. Si le modèle fonctionne bien sur les données de test et donne une bonne précision, cela signifie que le modèle n'a pas surajusté les données d'entraînement et peut être utilisé pour la prédiction.

Approfondissons et découvrons certaines des techniques d'évaluation de modèle.

III-METHODES DE VALIDATION CROISEE

1-METHODE D’ATTENTE

C'est la méthode d'évaluation la plus simple et elle est largement utilisée dans les projets de Machine Learning. Ici, l'ensemble de données (population) est divisé en 2 ensembles - ensemble de train et ensemble de test. Les données peuvent être divisées en 70-30 ou 60-40, 75-25 ou 80-20, voire 50-50 selon le cas d'utilisation. En règle générale, la proportion de données d'entraînement doit être plus importante que les données de test.


DATASET
Training Dataset
Testing Dataset
Source
TRAIN
TEST
Train Model
Evaluate Model

Source de l'image : DataVedas

Le fractionnement des données se produit de manière aléatoire et nous ne pouvons pas être sûrs des données qui se retrouvent dans le compartiment de train et de test pendant le fractionnement, à moins que nous ne spécifiions random_state. Cela peut conduire à une variance extrêmement élevée et à chaque fois que la division change, la précision changera également.

Il y a quelques inconvénients à cette méthode :

- Dans la méthode Hold out, les taux d'erreur de test sont très variables (variance élevée) et cela dépend totalement des observations qui se retrouvent dans l'ensemble d'apprentissage et l'ensemble de test.
- Seule une partie des données est utilisée pour entraîner le modèle (biais élevé), ce qui n'est pas une très bonne idée lorsque les données ne sont pas énormes et cela conduira à une surestimation de l'erreur de test.

L'un des principaux avantages de cette méthode est qu'elle est peu coûteuse en termes de calcul par rapport à d'autres techniques de validation croisée.

Implémentation rapide de la méthode Hold Out en Python

```
de sklearn.model_selection importer train_test_split
```

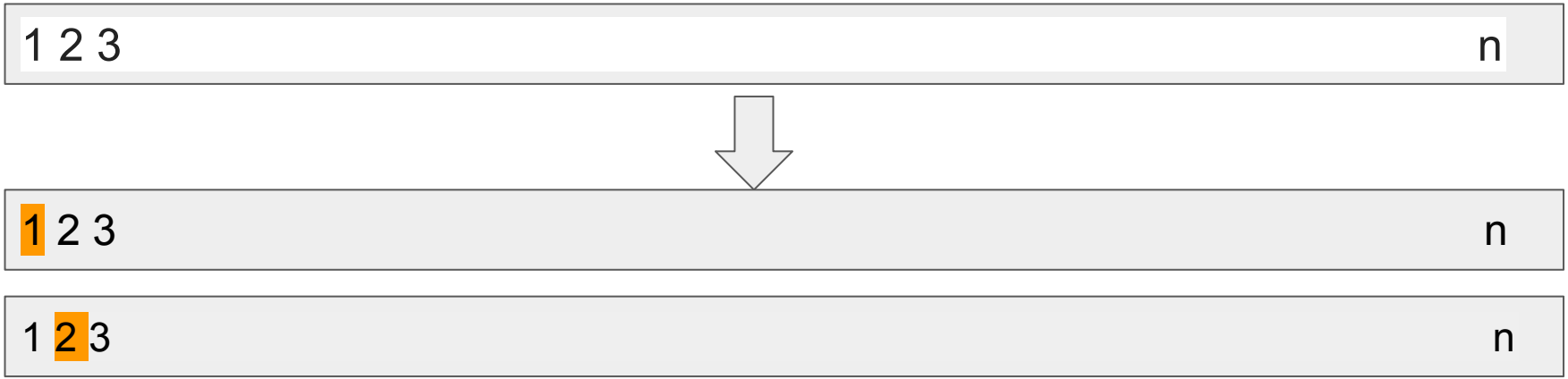
```
X = [10,20,30,40,50,60,70,80,90,100]

train, test= train_test_split(X,test_size=0.3, random_state=1)

print("Train : », X_train, « Test : » ,X_test)
```

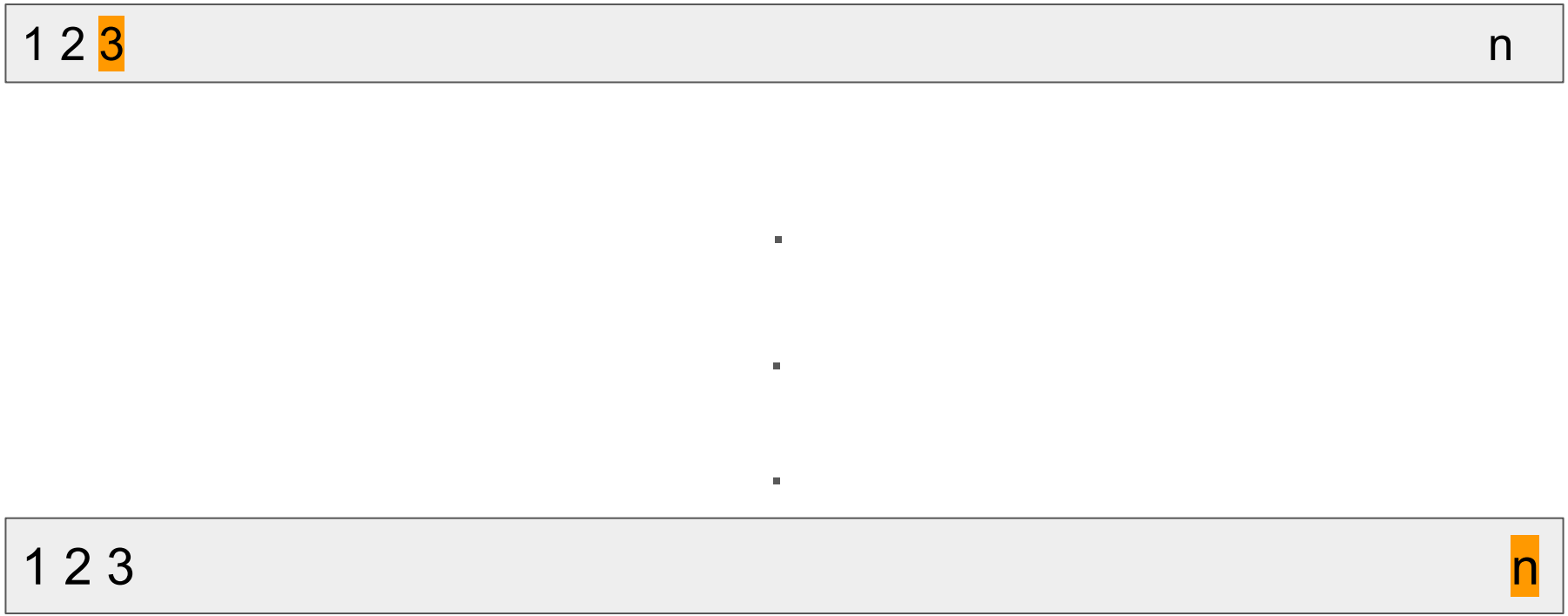
sortie

```
Entraînement : [50, 10, 40, 20, 80, 90, 60] Test : [30, 100, 70]
```

2-LAISSEZ UNE VALIDATION CROISEE DE COTE

Dans cette méthode, nous divisons les données en ensembles de train et de test - mais avec une torsion. Au lieu de diviser les données en 2 sous-ensembles, nous sélectionnons une seule observation comme données de test, et tout le reste est étiqueté comme données d'entraînement et le modèle est entraîné. Maintenant, la 2e observation est sélectionnée comme données de test et le modèle est entraîné sur les données restantes.





Ce processus se poursuit 'n' fois et la moyenne de toutes ces itérations est calculée et estimée en tant qu'erreur de l'ensemble de test.

$$\mathrm{CV}_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\mathrm{MSE}_i.$$

En ce qui concerne les estimations d'erreur de test, LOOCV donne des estimations non biaisées (faible biais). Mais le biais n'est pas le seul sujet de préoccupation dans les problèmes d'estimation. Nous devrions également considérer la variance.

LOOCV a une variance extrêmement élevée car nous faisons la moyenne des sorties de n-modèles qui sont ajustés sur un ensemble d'observations presque identique, et leurs sorties sont fortement corrélées les unes aux autres.

Et vous pouvez clairement voir que cela coûte cher en calcul car le modèle est exécuté 'n' fois pour tester chaque observation dans les données. Notre prochaine méthode s'attaquera à ce problème et nous donnera un bon équilibre entre biais et variance.

Implémentation rapide de la validation croisée Leave One Out en Python

```
de sklearn.model_selection importer LeaveOneOut
```

```
X = [10,20,30,40,50,60,70,80,90,100]

l = LeaveOneOut()

pour train, test dans l.split(X):

    print("%s %s"% (train, test))
```

Sortir

```
[1 2 3 4 5 6 7 8 9] [0]
[0 2 3 4 5 6 7 8 9] [1]
[0 1 3 4 5 6 7 8 9] [2]
[0 1 2 4 5 6 7 8 9] [3]
[0 1 2 3 5 6 7 8 9] [4]
[0 1 2 3 4 6 7 8 9] [5]
[0 1 2 3 4 5 7 8 9] [6]
[0 1 2 3 4 5 6 8 9] [7]
[0 1 2 3 4 5 6 7 9] [8]
[0 1 2 3 4 5 6 7 8] [9]
```

Cette sortie montre clairement comment LOOCV garde une observation de côté en tant que données de test et toutes les autres observations vont aux données d'apprentissage.

3-VALIDATION CROISEE K-FOLD

Dans cette technique de rééchantillonnage, l'ensemble des données est divisé en k ensembles de tailles presque égales. Le premier ensemble est sélectionné comme ensemble de test et le modèle est entraîné sur les ensembles k-1 restants. Le taux d'erreur de test est ensuite calculé après ajustement du modèle aux données de test.

Dans la deuxième itération, le 2e ensemble est sélectionné comme ensemble de test et les k-1 ensembles restants sont utilisés pour entraîner les données et l'erreur est calculée. Ce processus se poursuit pour tous les k ensembles.

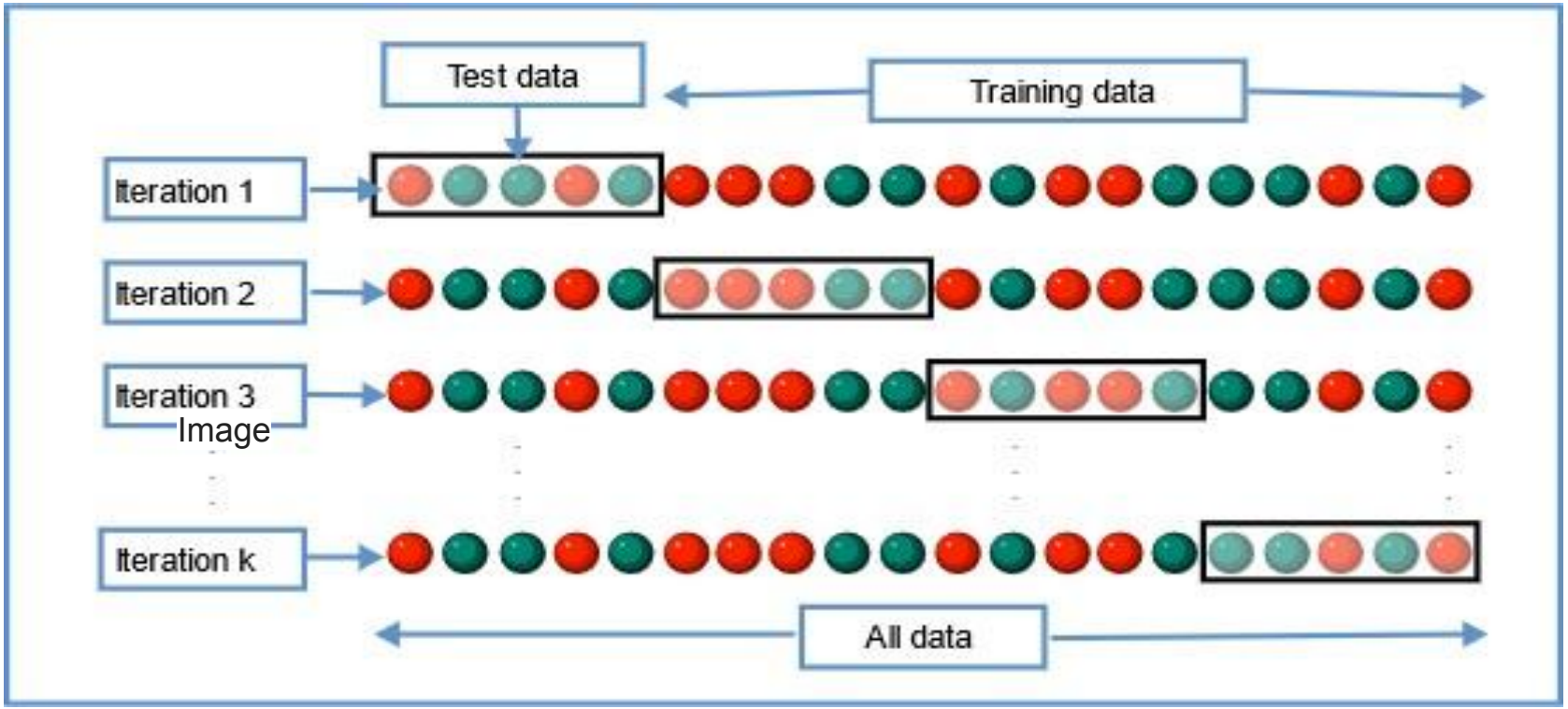

Test data
Training data
Iteration 1
Iteration 2
Iteration 3
Image
Iteration k
All data

Image Source: Wikipedia

La moyenne des erreurs de toutes les itérations est calculée comme l'estimation de l'erreur de test du CV.

$$\mathrm{CV}_{(k)} = \frac{1}{k}\sum_{i=1}^{k}\mathrm{MSE}_i.$$

Dans K-Fold CV, le nombre de plis k est inférieur au nombre d'observations dans les données (k<n) et nous faisons la moyenne des sorties de k modèles ajustés qui sont un peu moins corrélés les uns aux autres depuis le chevauchement entre les ensembles dans chaque modèle est plus petit. Cela conduit à une faible variance puis LOOCV.

La meilleure partie de cette méthode est que chaque point de données se trouve dans l'ensemble de test exactement une fois et fait partie de l'ensemble d'apprentissage k-1 fois. Lorsque le nombre de plis k augmente, la variance diminue également (faible variance). Cette méthode conduit à un biais intermédiaire car chaque ensemble d'apprentissage contient moins d'observations (k-1)n/k que la méthode Leave One Out mais plus que la méthode Hold Out.

En règle générale, la validation croisée K-fold est effectuée en utilisant k=5 ou k=10 car il a été démontré empiriquement que ces valeurs donnent des estimations d'erreur de test qui n'ont ni biais ni variance élevées.

Le principal inconvénient de cette méthode est que le modèle doit être exécuté à partir de zéro k fois et qu'il est coûteux en temps de calcul que la méthode Hold Out, mais meilleur que la méthode Leave One Out.

```
à partir de sklearn.model_selection importer KFold
```

```
X = ["a",'b','c','d','e','f']

kf = KFold(n_splits=3, shuffle=False, random_state=Aucun)

pour le train, testez dans kf.split(X) :

    print("Données de train",train,"Données de test",test)
```

```
Sortir
```

```
Train: [2 3 4 5] Test: [0 1]
Train: [0 1 4 5] Test: [2 3]
Train: [0 1 2 3] Test: [4 5]
```

4-VALIDATION CROISEE STRATIFIEE DE K-FOLD

Il s'agit d'une légère variation par rapport à la validation croisée K-Fold, qui utilise un « échantillonnage stratifié » au lieu d'un « échantillonnage aléatoire ». Comprenons rapidement ce qu'est l'échantillonnage stratifié et en quoi est-il différent de

l'échantillonnage aléatoire.

Supposons que vos données contiennent des avis sur un produit cosmétique utilisé à la fois par la population masculine et féminine. Lorsque nous effectuons un échantillonnage aléatoire pour diviser les données en ensembles d'entraînement et de test, il est possible que la plupart des données représentant les hommes ne soient pas représentées dans les données d'entraînement mais puissent se retrouver dans les données de test. Lorsque nous entraînons le modèle sur des échantillons de données d'apprentissage qui ne sont pas une représentation correcte de la population réelle, le modèle ne prédit pas les données de test avec une bonne précision.

C'est là que l'échantillonnage stratifié vient à la rescousse. Ici, les données sont divisées de manière à représenter toutes les classes de la population.

Considérons l'exemple ci-dessus qui a une évaluation de produits cosmétiques de 1000 clients dont 60% sont des femmes et 40% sont des hommes. Je veux diviser les données en données de train et de test proportionnellement (80:20). 80% des 1000 clients seront 800 qui seront choisis de telle sorte qu'il y ait 480 avis associés à la population féminine et 320 représentant la population masculine. De la même manière, 20% des 1000 clients seront choisis pour les données de test (avec la même représentation féminine et masculine).

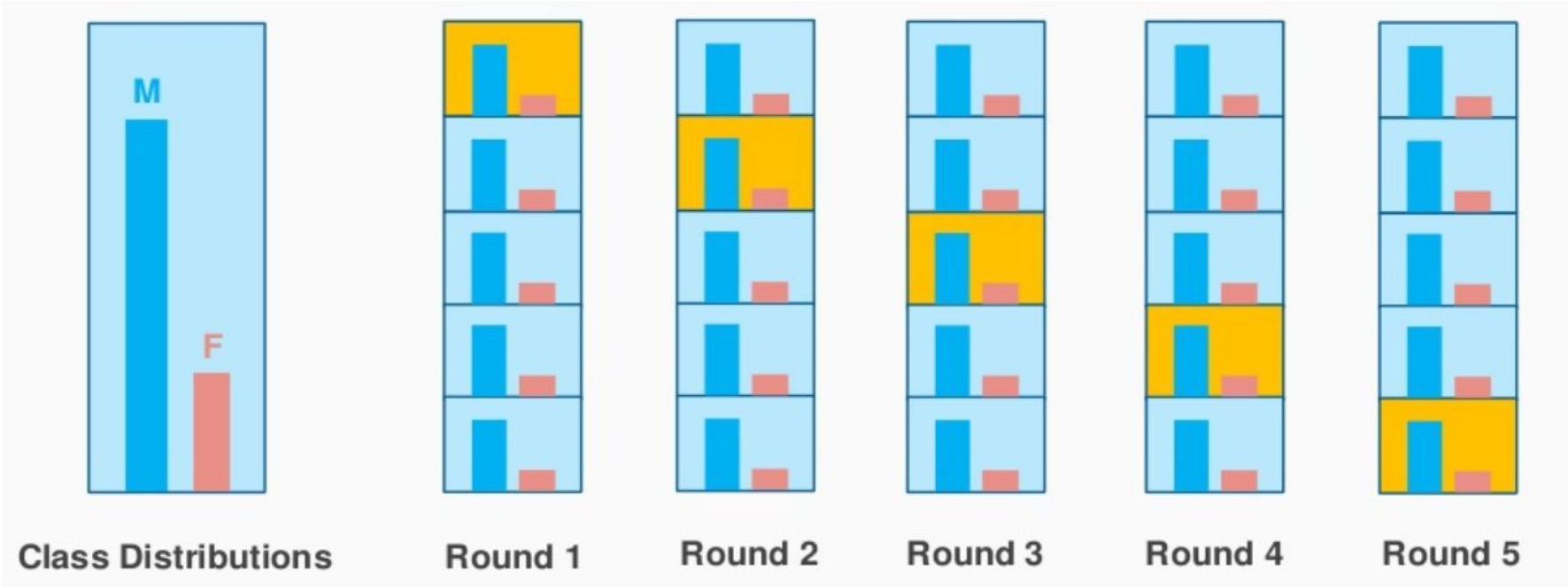

M
F
Class Distributions
Round 1
Round 2
Round 3
Round 4
Round 5

Image Source: stackexchange.com

C'est exactement ce que fait le CV stratifié K-Fold et il créera des K-Folds en préservant le pourcentage d'échantillon pour chaque classe. Cela résout le problème de l'échantillonnage aléatoire associé aux méthodes Hold out et K-Fold.

Implémentation rapide de la validation croisée stratifiée K-Fold en Python

```
à partir de sklearn.model_selection importer StratifiedKFold
```

```
X = np.array([[1,2],[3,4],[5,6],[7,8],[9,10],[11,12]])
y = np.tableau ([0,0,1,0,1,1])

skf =
StratifiedKFold(n_splits=3,random_state=None,shuffle=False)

pour train_index,test_index dans skf.split(X,y):
    print("Train :",train_index,'Test :',test_index)
    X_train,X_test = X[train_index], X[test_index]
    y_train,y_test = y[train_index], y[test_index]
```

```
Sortir
```

```
Former : [1 3 4 5] Tester : [0 2]
Former : [0 2 3 5] Tester : [1 4]
Former : [0 1 2 4] Tester : [3 5]
```

La sortie montre clairement la division stratifiée effectuée sur la base des classes « 0 » et « 1 » dans « y »