

Ensemble

☑ Trắc nghiệm 3 ☑ Thực hành 1 ⬇ PDF bài

Ensemble là gộp dự đoán của nhiều model. Gộp chỉ có lợi khi các model sai ở những chỗ khác nhau, để lỗi của model này được model kia bù. Nếu các model cùng sai một kiểu, ví dụ cùng dựa vào một dấu hiệu tình cờ trong ảnh, thì thêm bao nhiêu model cũng không giúp. Bài này học các cách gộp (bỏ phiếu, lấy trung bình, trung bình có trọng số, stacking) và cách chọn trọng số trên một tập rồi kiểm trên tập khác, để điểm báo cáo không bị đẹp giả.

MỤC TIÊU

- Tính các cách gộp: bỏ phiếu, trung bình xác suất, trung bình hạng, trung bình có trọng số.
- Biết khi nào gộp có lợi.
- Chọn trọng số trên một tập, kiểm trên tập khác.
- Làm stacking bằng dự đoán out-of-fold.

Gộp nhãn khác gộp xác suất

Hai cách gộp phân loại:

- **Hard voting** chọn lớp được nhiều model dự đoán nhất.
- **Soft voting** lấy trung bình các xác suất từng lớp rồi chọn lớp có xác suất cao nhất.

Hai cách có thể cho kết quả khác, vì hard voting bỏ mức tự tin. Ta lấy mẫu đầu tiên: ba xác suất lớp 1 là 0.51, 0.51, 0.01. Hai model hơi nghiêng về lớp 1, một model nghiêng mạnh về lớp 0. Hard voting cho lớp 1, còn mean $1.03/3 \approx 0.343$ cho lớp 0.

Python

```
import numpy as np

# Mỗi hàng là một mẫu; mỗi cột là xác suất lớp 1 của một model.
probabilities = np.array([
```

```

[0.51, 0.51, 0.01],
[0.90, 0.60, 0.60],
[0.20, 0.80, 0.70],
[0.80, 0.10, 0.10],
])
hard_labels = (probabilities >= 0.5).astype(int)
majority = (hard_labels.sum(axis=1) >= 2).astype(int)
mean_probability = probabilities.mean(axis=1)
soft_labels = (mean_probability >= 0.5).astype(int)
print("shape", probabilities.shape)
print("hard", majority.tolist())
print("mean probabilities", mean_probability.round(3).tolist())
print("soft", soft_labels.tolist())

```

KẾT QUẢ

```

shape (4, 3)
hard [1, 1, 1, 0]
mean probabilities [0.343, 0.7, 0.567, 0.333]
soft [0, 1, 1, 0]

```

Đọc hàng đầu để thấy hai đáp án khác nhau. Soft voting không luôn đúng hơn: một model tự tin sai có thể kéo mean sang lớp sai.

Xác suất được hiệu chỉnh (calibrated) giúp mức tự tin có ý nghĩa; một model cho 0.9 cần đúng gần 90% trên các trường hợp tương tự để cách đọc ấy hợp lý. Tài liệu [VotingClassifier](#) phân biệt hai cơ chế và khuyến nghị soft voting cho các bộ phân loại có xác suất được hiệu chỉnh tốt.

Với K model và trọng số không âm w_k có tổng 1, xác suất gộp là

$$p(c \mid x) = \sum_{k=1}^K w_k p_k(c \mid x).$$

c chỉ lớp, x chỉ một mẫu. Với bài hồi quy, thay xác suất bằng giá trị dự đoán là trung bình dự đoán. Không gộp ID hay chỉ số lớp như một số thực: trung bình nhãn cat=0, dog=1, bird=2 thành 1 không có ý nghĩa “ở giữa mèo và chim”. Nếu model xuất logits, mean logits rồi softmax khác mean probabilities; chọn cách nào phải được kiểm đúng metric.

Hard voting với số model chẵn có thể hòa phiếu; khi hòa, `VotingClassifier` và `np.argmax(np.bincount(...))` đều chọn lớp có chỉ số nhỏ hơn, một quy tắc ngầm mà

người viết không chủ ý đặt. Số model lẻ tránh hòa với hai lớp; với nhiều lớp, ba model trả ba nhãn khác nhau vẫn hòa.

	model 1	model 2	model 3	hard	mean	soft
mẫu 1	0.51	0.51	0.01	1	0.343	0
mẫu 2	0.90	0.60	0.60	1	0.700	1
mẫu 3	0.20	0.80	0.70	1	0.567	1
mẫu 4	0.80	0.10	0.10	0	0.333	0

ô cam: nghiêng về lớp 1; mẫu 1 là chỗ hai cách gộp khác nhau

Bỏ phiếu chọn lớp 1; trung bình xác suất chọn lớp 0 vì một model rất chắc.

HỎI NHANH

Hai model cho xác suất lớp 1 là 0.2 và 0.8, với trọng số lần lượt là 0.75 và 0.25. Xác suất gộp bằng bao nhiêu?

A 0.5

B 0.35

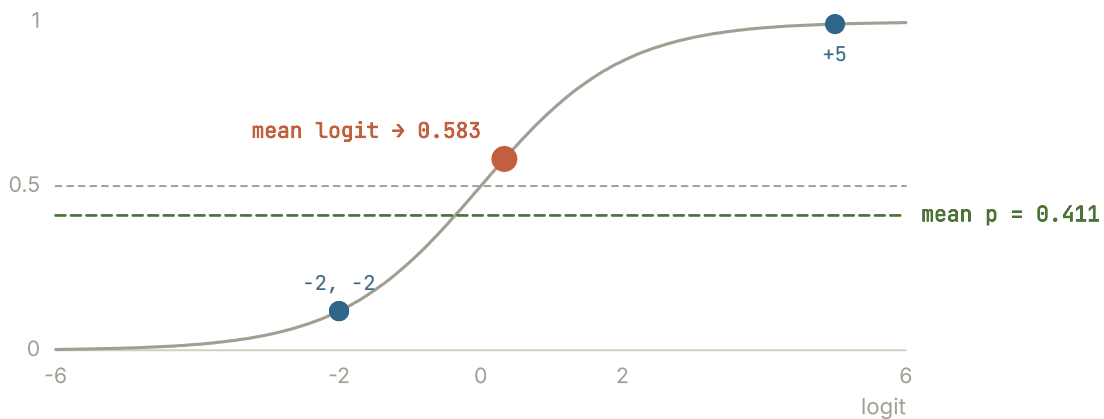
C 0.65

D 1.0

HỎI NHANH

Ba model nhị phân cho logit +5, -2, -2 ở cùng một mẫu, và mẫu thuộc lớp 1 nếu xác suất từ 0.5 trở lên. Lấy trung bình logit rồi qua sigmoid, và lấy trung bình các xác suất sigmoid, lần lượt cho lớp nào?

- A Trung bình logit cho lớp 0, trung bình xác suất cho lớp 0
- B Trung bình logit cho lớp 0, trung bình xác suất cho lớp 1
- C Trung bình logit cho lớp 1, trung bình xác suất cho lớp 0
- D Trung bình logit cho lớp 1, trung bình xác suất cho lớp 1



Trung bình logit và trung bình xác suất có thể cho hai lớp khác nhau.

Khi metric chỉ xét thứ tự: trung bình hạng

AUC và các metric xếp hạng chỉ dùng thứ tự điểm giữa các mẫu, không dùng độ lớn. Hai model thường cho điểm ở hai thang khác nhau: một model nén mọi điểm quanh 0.5, model kia trải từ 0.05 đến 0.95. Trung bình điểm thô để model thang rộng lấn át, dù hai model có giá trị như nhau.

Trung bình hạng (rank averaging) đổi điểm của mỗi model thành hạng của mẫu trong tập, rồi lấy trung bình các hạng. Ví dụ dựng sẵn dưới đây có sáu mẫu, ba âm và ba dương.

Python

```
from scipy.stats import rankdata
from sklearn.metrics import roc_auc_score

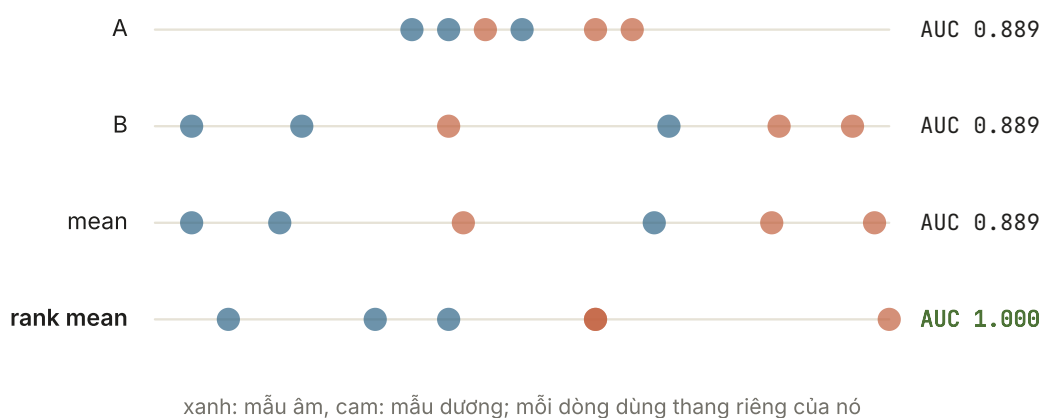
y_small = np.array([0, 0, 0, 1, 1, 1])
score_a = np.array([0.47, 0.48, 0.50, 0.49, 0.52, 0.53]) # thang hẹp quanh 0.5
score_b = np.array([0.20, 0.70, 0.05, 0.85, 0.40, 0.95]) # thang rộng
mean_score = (score_a + score_b) / 2
rank_score = (rankdata(score_a) + rankdata(score_b)) / 2
for name, score in [("A", score_a), ("B", score_b),
                    ("mean", mean_score), ("rank mean", rank_score)]:
    print(name, "AUC", f"{roc_auc_score(y_small, score):.3f}")
print("rank scores", rank_score.tolist())
```

KẾT QUẢ

```
A AUC 0.889
B AUC 0.889
mean AUC 0.889
rank mean AUC 1.000
rank scores [1.5, 3.0, 2.5, 4.0, 4.0, 6.0]
```

Mỗi model xếp đúng 8 trên 9 cặp (dương, âm), nên AUC là 0.889. A sai ở một cặp (mẫu dương 0.49 thấp hơn mẫu âm 0.50), B sai ở một cặp khác (mẫu âm 0.70 cao hơn mẫu dương 0.40). Trung bình điểm thô sửa được lỗi của A nhờ B, nhưng giữ lại lỗi của B: khoảng cách điểm của B ở cặp ấy là 0.30, của A chỉ 0.04, nên B lấn át.

Trung bình hạng cho mỗi model đóng góp theo thứ tự thay vì theo độ rộng thang, và cả hai lỗi đều bị model kia ghi đè: AUC lên 1.000.



Trung bình điểm thô để một model lấn át; trung bình hạng thì không.

Hạng chỉ có nghĩa trong một tập dự đoán đầy đủ, vì hạng của một mẫu phụ thuộc mọi mẫu khác. Test chia thành nhiều file phải được gộp trước khi xếp hạng. Hạng cũng không phải xác suất: log loss, F1 theo ngưỡng và mọi metric cần độ lớn không tính được trên hạng.

HỎI NHANH

Test gồm hai file, mỗi file 5000 mẫu, và metric là AUC trên cả 10 000 mẫu. Hai model đã dự đoán xong. Lấy trung bình hạng thế nào cho đúng?

- A Gộp hai file, xếp hạng chung, rồi tách lại theo ID
- B Xếp hạng riêng từng file, rồi nối hai bảng hạng lại
- C Xếp hạng file đầu, giữ điểm thô cho file sau
- D Với mỗi mẫu, xếp hạng hai model rồi lấy trung bình

Nhiều model giảm phần lỗi nào?

Hãy tưởng tượng ba người cùng đo một chiều dài. Nếu sai số đo độc lập, lấy trung bình giảm dao động. Nếu cả ba dùng một thước bị lệch thêm 2 cm, lấy trung bình vẫn lệch 2 cm. Ensemble cũng vậy: khác seed có thể giảm dao động huấn luyện, nhưng cùng dùng một nhãn sai, một split rò rỉ hoặc một đường tắt thì lỗi chung còn nguyên.

Một công thức nhỏ cho trực giác. Giả sử các sai số ϵ_k đều có trung bình 0, phương sai σ^2 và mọi cặp có cùng tương quan ρ . Phương sai của trung bình là

$$\text{Var}(\bar{\epsilon}) = \frac{\sigma^2}{K} (1 + (K - 1)\rho).$$

Có K hạng phương sai trên đường chéo và $K(K - 1)$ hạng hiệp phương sai, mỗi hạng bằng $\rho\sigma^2$; chia tổng cho K^2 cho công thức. Nếu $\rho = 0$, phương sai giảm còn σ^2/K . Nếu $\rho = 1$, nó vẫn là σ^2 . Nếu $\rho > 0$, tăng model mãi còn giới hạn $\rho\sigma^2$. Đây là mô hình sai số đơn giản, không phải định luật dự báo accuracy của mọi ensemble.

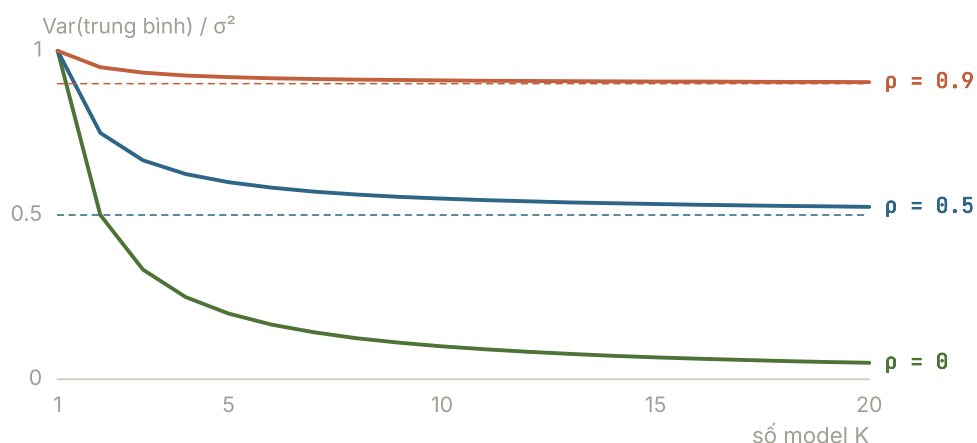
Python

```
for correlation in [0.0, 0.5, 1.0]:  
    k = 4  
    variance_ratio = (1 + (k - 1) * correlation) / k  
    print("rho", correlation, "variance ratio", f"{variance_ratio:.3f}")
```

KẾT QUẢ

```
rho 0.0 variance ratio 0.250  
rho 0.5 variance ratio 0.625  
rho 1.0 variance ratio 1.000
```

Bốn model độc lập cho tỉ lệ 0.25; tương quan 0.5 cho 0.625; giống hệt nhau cho 1.0. Đó là lý do ta xem **mẫu sai**, không chỉ xem metric mỗi model. Model thứ hai có thể hơi yếu hơn tổng thể nhưng sửa đúng một nhóm model thứ nhất thường sai. Nó cũng có thể bất đồng vì quá yếu; bất đồng tự nó chưa chứng minh ích lợi.



Thêm model chỉ kéo phương sai sai số về mức sàn ρ ; muốn thấp hơn phải giảm tương quan.

HỎI NHANH

Mười bản cùng kiến trúc, cùng dữ liệu, chỉ khác seed, và sai số của mọi cặp có tương quan $\rho = 0.9$. Theo mô hình của bài, phương sai của sai số trung bình bằng bao nhiêu so với một bản?

A $0.10\sigma^2$

B $0.90\sigma^2$

C $1.00\sigma^2$

D $0.91\sigma^2$

Bỏ phiếu cũng theo logic ấy. Giả sử K model nhị phân, mỗi model đúng với xác suất p trên mỗi mẫu và sai độc lập nhau; đa số đúng khi có quá nửa số model đúng, xác suất ấy là một tổng nhị thức. Hàm dưới đây tính nó, rồi thử một phần số mẫu mà cả ba model luôn cùng đúng hoặc cùng sai (lỗi trùng hoàn toàn), phần còn lại độc lập.

Python

```
from math import comb

def majority_accuracy(p, k):
    """k model độc lập, mỗi model đúng với xác suất p; đa số đúng khi hơn nửa số model
    đúng."""
    return sum(comb(k, j) * p**j * (1 - p)**(k - j) for j in range(k // 2 + 1, k + 1))

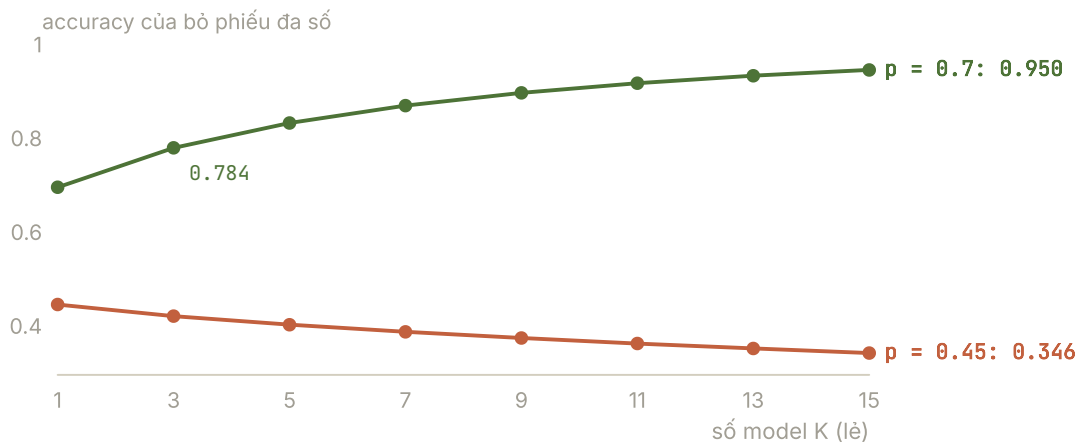
print("p=0.7", "3 models", f"{majority_accuracy(0.7, 3):.3f}",
      "15 models", f"{majority_accuracy(0.7, 15):.3f}")
for shared in [0.0, 0.5, 1.0]:
    vote = shared * 0.7 + (1 - shared) * majority_accuracy(0.7, 3)
    print("shared", shared, "vote of 3", f"{vote:.3f}")
```

KẾT QUẢ

```
p=0.7 3 models 0.784 15 models 0.950
shared 0.0 vote of 3 0.784
shared 0.5 vote of 3 0.742
shared 1.0 vote of 3 0.700
```

Ba model độc lập, mỗi model đúng 70%, cho 0.784; mười lăm model cho 0.950. Khi một nửa số mẫu có lỗi trùng hoàn toàn ($\text{shared}=0.5$), ba model chỉ còn 0.742; khi mọi lỗi trùng ($\text{shared}=1.0$), bỏ phiếu bằng đúng một model, 0.700.

Điều kiện $p > 0.5$ cũng bắt buộc: khi mỗi model đúng ít hơn nửa số mẫu, thêm model làm bỏ phiếu tệ hơn nữa. Boosting đặt điều kiện tương tự: weak learner phải hơn đoán ngẫu nhiên.



Model đúng 70%: càng nhiều model, bỏ phiếu càng tốt. Model đúng 45%: càng tệ.

HỎI NHANH

Ba model đều đúng 70%. Trên một nửa số mẫu, cả ba luôn cùng đúng hoặc cùng sai. Trên nửa còn lại, ba model sai độc lập với nhau. Bỏ phiếu đa số cho accuracy gần mức nào nhất?

A 0.700

B 0.742

C 0.784

D 0.850

Chuẩn bị ba phần dữ liệu có vai trò riêng

Dữ liệu là hai hình bán nguyệt nhiễu từ `make_moons`, sinh tại chỗ. Mỗi mẫu có hai đặc trưng, nhãn 0 hoặc 1. Ta dành 360 mẫu để fit model, 120 mẫu tune để chọn trọng số, 120 mẫu audit để kiểm phương án đã khóa. Tên audit chỉ gọi tập kiểm giữ riêng trong notebook.

Python

```
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import log_loss, accuracy_score

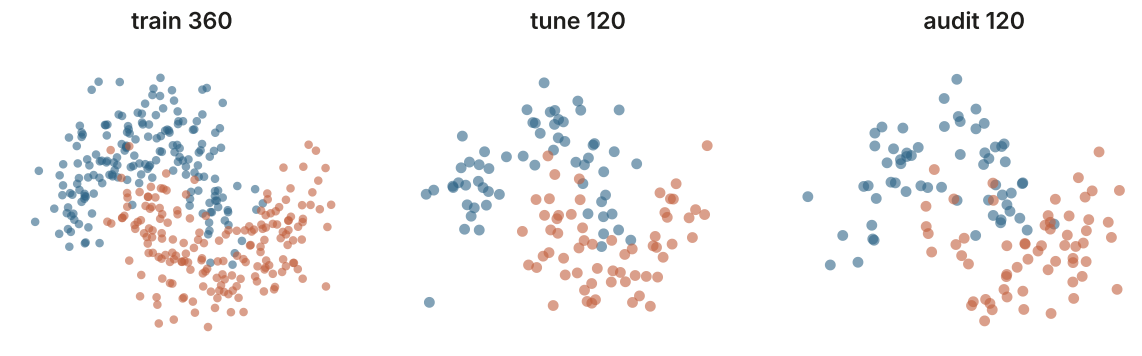
X, y = make_moons(n_samples=600, noise=0.28, random_state=84)
X_train, X_hold, y_train, y_hold = train_test_split(
    X, y, test_size=0.4, random_state=1, stratify=y)
X_tune, X_audit, y_tune, y_audit = train_test_split(
    X_hold, y_hold, test_size=0.5, random_state=2, stratify=y_hold)
models = {
    "linear": make_pipeline(StandardScaler(), LogisticRegression(max_iter=500)),
    "forest": RandomForestClassifier(n_estimators=40, max_depth=5,
                                    min_samples_leaf=3, random_state=0, n_jobs=1),
    "knn": make_pipeline(StandardScaler(), KNeighborsClassifier(n_neighbors=15)),
}
tune_probs = []
audit_probs = []
for name, model in models.items():
    model.fit(X_train, y_train)
    assert np.array_equal(model.classes_, [0, 1])
    p_tune = model.predict_proba(X_tune)
    p_audit = model.predict_proba(X_audit)
    tune_probs.append(p_tune)
    audit_probs.append(p_audit)
    print(name, "tune logloss", f"{log_loss(y_tune, p_tune):.4f}")
tune_probs = np.stack(tune_probs)
audit_probs = np.stack(audit_probs)
```

```
print("train/tune/audit", len(y_train), len(y_tune), len(y_audit))
print("probability shape", tune_probs.shape)
```

KẾT QUẢ

```
linear tune logloss 0.3229
forest tune logloss 0.2240
knn tune logloss 0.4201
train/tune/audit 360 120 120
probability shape (3, 120, 2)
```

Shape (3,120,2) có thứ tự **model, mẫu, lớp**. predict_proba cột đầu là lớp 0, cột sau là lớp 1 vì đã kiểm classes_. Khi dùng pandas ghép dự đoán, ta giữ cùng thứ tự ID; một tensor cùng shape nhưng hàng lệch vẫn cho một mean vô nghĩa. Fit scaler chỉ trên train nhờ Pipeline; tập audit chưa tham gia chọn trọng số.



cùng make_moons, chia một lần: tune chọn trọng số, audit chỉ mở cho phương án đã khóa

Dữ liệu chia ba phần: fit model, chọn trọng số, và giữ riêng để đánh giá.

HỎI NHANH

`P.shape == (3, 120, 2)`. Muốn lấy trung bình qua các model mà vẫn giữ mẫu và lớp thì dùng gì?

A `P.mean(axis=0)`, cho shape (120, 2)

B `P.mean(axis=1)`, cho shape (3, 2)

C `P.mean(axis=2)`, cho shape (3, 120)

D `P.mean()`, cho một số

Chọn trọng số bằng một tập, đánh giá bằng tập khác

Ta thử một lưới nhỏ: trọng số là bội của 0.25, không âm và tổng 1. Viết trọng số là $(a, b, c)/4$ với $a + b + c = 4$ nguyên không âm thì có $\binom{6}{2} = 15$ bộ. Đây là một tập phương án định trước, đủ nhỏ để đọc.

Với K model và lưới bước $1/g$, số bộ là $\binom{g+K-1}{K-1}$, tăng rất nhanh: bốn model và bước $1/20$ đã có 1771 bộ. Thử hàng nghìn trọng số trên 120 mẫu, ta có thể fit nhiều của chính tune dù không fit thêm model nào.

Python

```
weight_grid = []
for first in range(5):
    for second in range(5 - first):
        third = 4 - first - second
        weight_grid.append(np.array([first, second, third]) / 4)

def combine(weights, probabilities):
    return np.tensordot(weights, probabilities, axes=(0, 0))

losses = [log_loss(y_tune, combine(w, tune_probs)) for w in weight_grid]
best_index = int(np.argmin(losses))
chosen_weights = weight_grid[best_index]
equal_weights = np.full(3, 1 / 3)
```

```
print("candidate weights", len(weight_grid))
print("chosen weights", chosen_weights.tolist())
print("chosen tune loss", f"{losses[best_index]:.4f}")
print("equal tune loss", f"{log_loss(y_tune, combine(equal_weights, tune_probs)):.4f}")
```

KẾT QUẢ

```
candidate weights 15
chosen weights [0.0, 0.25, 0.75]
chosen tune loss 0.1687
equal tune loss 0.2190
```

Ta khóa `chosen_weights` sau khối này. Trọng số 0 có nghĩa bỏ model đó, nên lưới đã gồm cả model đơn. Nếu equal weights không thắng model đơn, đó là một kết quả cần giữ, không phải lỗi phải che. Model mạnh trên tune có thể không mạnh ở một miền khác; kiểm lỗi theo nhóm từ bài 8.2 vẫn cần thiết.

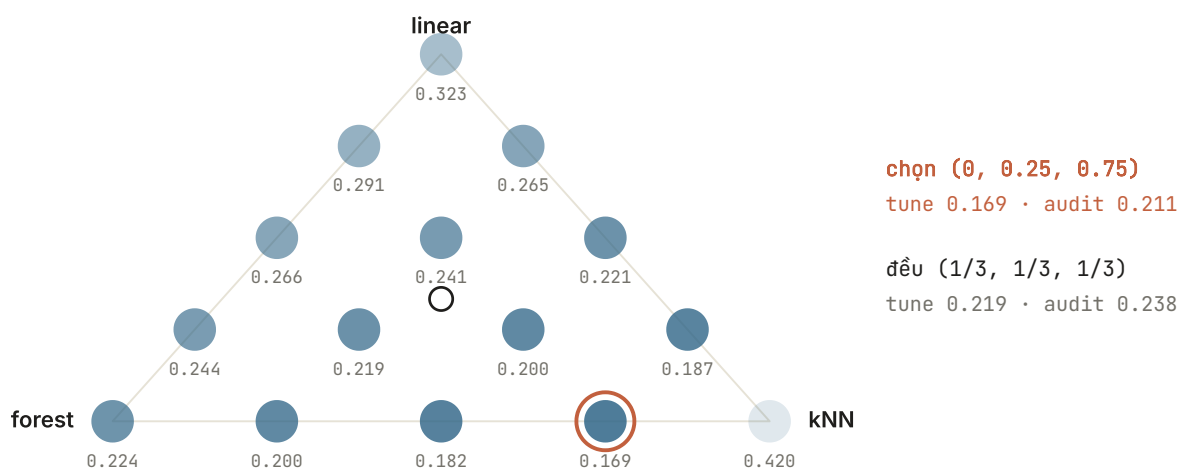
Giờ mở tập audit **một lần cho phương án đã chọn**, đồng thời báo mốc model đơn và equal weights để hiểu hệ quả. Trong bài luyện thật, nếu dùng kết quả audit để sửa lại phương án, tập này đã trở thành validation; cần một tập đánh giá khác hoặc ghi đúng giới hạn phép đo.

Python

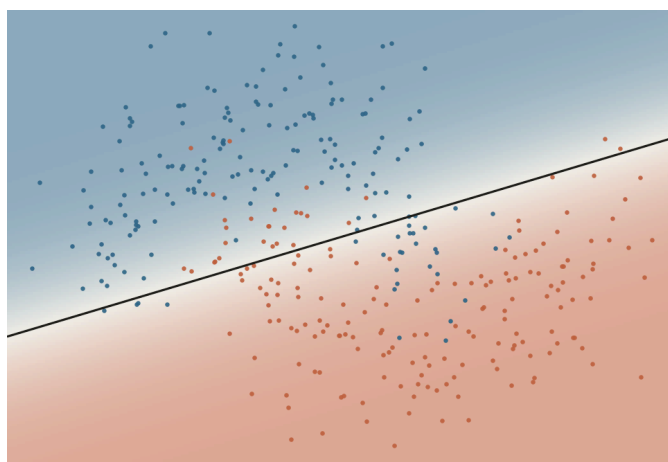
```
audit_ensemble = combine(chosen_weights, audit_probs)
for name, p in list(zip(models, audit_probs)) + [
    ("equal", combine(equal_weights, audit_probs)),
    ("chosen", audit_ensemble),
]:
    labels = np.argmax(p, axis=1)
    print(name, "audit logloss", f"{log_loss(y_audit, p):.4f}",
          "accuracy", f"{accuracy_score(y_audit, labels):.3f}")
```

KẾT QUẢ

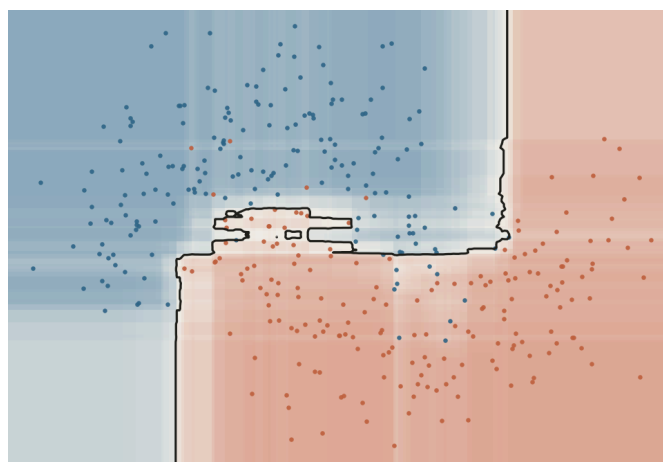
```
linear audit logloss 0.3124 accuracy 0.842
forest audit logloss 0.2571 accuracy 0.892
knn audit logloss 0.4809 accuracy 0.925
equal audit logloss 0.2384 accuracy 0.908
chosen audit logloss 0.2113 accuracy 0.925
```



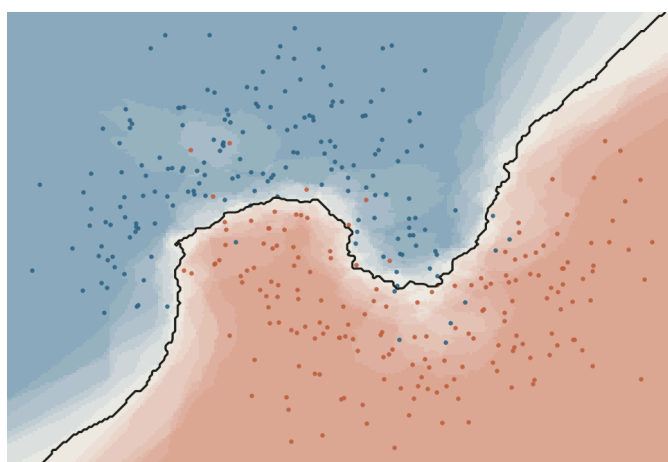
Trọng số chọn trên tập tune đạt 0.169, trên tập giữ riêng 0.211.



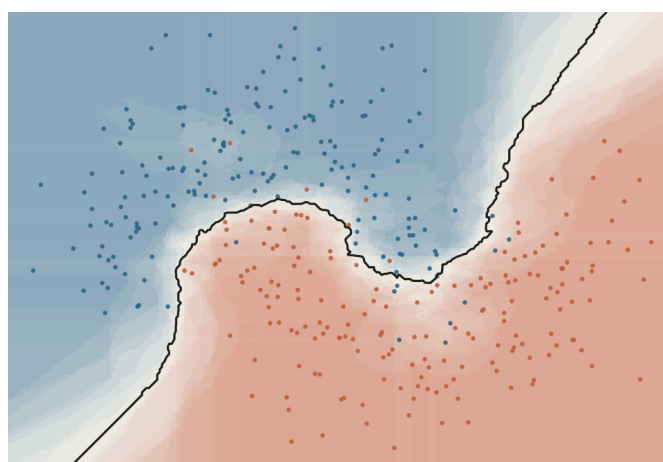
logistic regression: log loss tune 0.3229



random forest: log loss tune 0.2240



kNN 15 láng giềng: log loss tune 0.4201



trọng số đã chọn trên tune (0.0, 0.25, 0.75): log loss tune 0.1687

Xác suất lớp 1 của ba model, rồi của trung bình đều và trung bình có trọng số.

Đọc cả log loss và accuracy. Trong lần chạy này, kNN đơn và bản chọn đều đạt accuracy 0.925 trên audit, nhưng log loss giảm từ 0.4809 xuống 0.2113: một model có thể cho cùng nhãn mà xác suất bất cực đoan.

Lý do nằm ở chính kNN với 15 hàng xóm: khi cả 15 hàng xóm cùng nhãn, nó trả xác suất đúng 0 hoặc 1, và một lần tự tin sai bị log loss phạt gần như vô hạn. Ta đếm trên tune.

Python

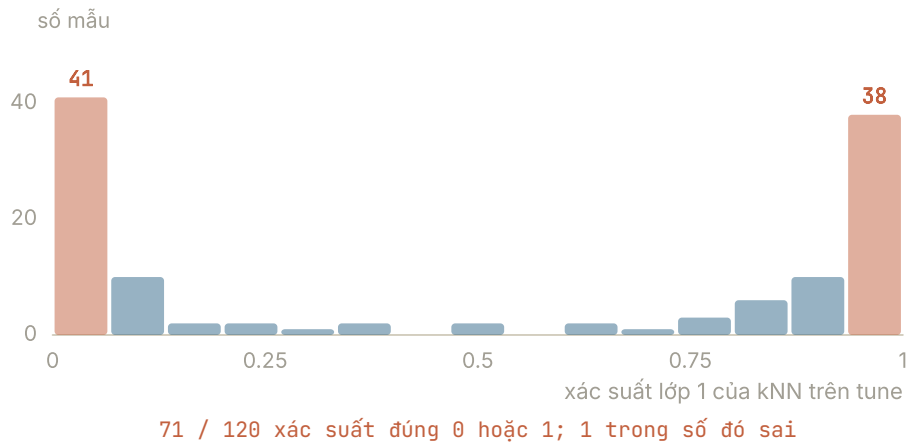
```
knn_tune = tune_probs[2, :, 1]
extreme = (knn_tune == 0) | (knn_tune == 1)
wrong_extreme = extreme & ((knn_tune >= 0.5) != y_tune)
clipped = np.clip(tune_probs[2], 0.01, 0.99)
print("kNN extreme probabilities", int(extreme.sum()), "of", len(y_tune))
print("extreme and wrong", int(wrong_extreme.sum()))
print("kNN tune logloss raw", f"{log_loss(y_tune, tune_probs[2]):.4f}",
      "clipped", f"{log_loss(y_tune, clipped):.4f}")
```

KẾT QUẢ

```
kNN extreme probabilities 71 of 120
extreme and wrong 1
kNN tune logloss raw 0.4201 clipped 0.1639
```

71 trong 120 xác suất của kNN là đúng 0 hoặc 1, và một mẫu trong số đó sai: riêng mẫu ấy góp khoảng 36 trên tổng 50.4 của log loss tune (0.4201×120). Chỉ cắt xác suất vào đoạn $[0.01, 0.99]$ đã đưa log loss về 0.1639, gần bản trộn forest và kNN (0.1687). Trọng số 0.25 cho forest, vốn không cho xác suất đúng 0 hoặc 1, làm đúng việc đó: chặn dưới các xác suất của kNN.

Vậy trên tập tune, phần lớn lợi ích của trọng số đã chọn đến từ hiệu chỉnh xác suất, không phải từ bù lỗi giữa các model; muốn biết điều này có giữ trên dữ liệu mới, so hai cách trên audit chứ không kết luận từ tune. Bộ trọng số (0, 0.25, 0.75) gắn với dữ liệu và metric của bài; bài khác phải chọn lại.



kNN cho nhiều xác suất đúng 0 hoặc 1; một mẫu tự tin sai chiếm phần lớn log loss.

Cũng cần đọc độ lớn của các con số. Bản chọn có log loss 0.1687 trên tune nhưng 0.2113 trên audit: một phần của chênh lệch 0.04 ấy là cái giá của việc chọn trên chính tune.

Accuracy trên 120 mẫu có độ lệch chuẩn cỡ $\sqrt{0.9 \cdot 0.1/120} \approx 0.027$, tức hơn ba mẫu; chênh 0.892 và 0.925 (bốn mẫu) giữa forest và bản chọn chưa đủ để xếp hạng hai phương án. Ngược lại, metric F1 dùng ngưỡng và phụ thuộc phân bố lớp; trọng số tốt cho log loss chưa chắc tốt cho F1. Chọn theo metric của bài, các metric khác chỉ giúp chẩn đoán.

Còn câu hỏi về tương quan sai số. Ta đo hai thứ trên tune: tỉ lệ nhãn khác nhau (disagreement) và tương quan của sai số $p_1 - y$, tức xác suất lớp 1 trừ nhãn thật.

Python

```
predicted_labels = np.argmax(tune_probs, axis=2)
tune_errors = tune_probs[:, :, 1] - y_tune      # shape (3, 120)
error_corr = np.corrcoef(tune_errors)
names = list(models)
for first, second in [(0, 1), (0, 2), (1, 2)]:
    disagreement = np.mean(predicted_labels[first] != predicted_labels[second])
    print(names[first], names[second], f"disagree {disagreement:.3f}",
          f"corr {error_corr[first, second]:.3f}")
```

KẾT QUẢ

```
linear forest disagree 0.067 corr 0.915
linear knn disagree 0.100 corr 0.791
forest knn disagree 0.033 corr 0.896
```

Hai con số đo hai điều khác nhau. Disagreement chỉ đếm nhãn khác nhau, từ 3% đến 10%. Tương quan sai số gần với ρ của công thức hơn: 0.79 đến 0.92, cao vì cả ba model cùng đối mặt với nhiễu của dữ liệu và cùng sai ở vùng hai bán nguyệt chồng lên nhau. Với $K = 3$ và $\rho \approx 0.9$, công thức cho tỉ lệ $(1 + 2 \cdot 0.9)/3 \approx 0.93$: gộp có lợi nhưng ít. Cùng hướng ấy, equal weights chỉ hạ log loss tune từ 0.2240 (forest) xuống 0.2190; mức chênh 0.005 này nhỏ hơn dao động giữa các seed của riêng forest, như mục sau cho thấy. Khi hai model bất đồng, một có thể đúng hoặc cả hai sai; với nhiều lớp, trường hợp cả hai sai nhưng khác nhãn rất thường gặp.

Xem các dòng bất đồng cùng nhãn validation cho biết model nào giúp nhóm nào. Không dùng nhãn test kín để làm bước này.

HỎI NHANH

Ta dùng audit để chọn lại trọng số, vì một bộ khác có điểm cao hơn trên audit. Vai trò của audit đã thay đổi thế nào?

- A Vẫn độc lập, vì ta không fit thêm model nào
- B Đã thành tập train cho forest, vì nó có nhãn
- C Đã thành tập dùng để chọn, nên điểm trên nó lạc quan
- D Vẫn độc lập, nếu chỉ thử đúng hai lần

HỎI NHANH

Ta chọn trọng số cho bốn model bằng một lưới bước 1/20, gồm 1771 bộ, trên tập tune có 120 mẫu, rồi báo log loss tune của bộ tốt nhất. So với log loss trên audit, con số ấy thường thế nào?

- A Xấp xỉ audit, vì cả hai tập đều không dùng để fit model
- B Cao hơn audit, vì lưới lớn phạt các bộ trọng số
- C Cao hơn audit, vì tune chỉ có 120 mẫu
- D Thấp hơn audit, vì bộ được chọn khớp cả nhiều của tune

Đổi seed và trung bình nhiều lần chạy

Nhiều seed tạo đa dạng khi thuật toán có thành phần ngẫu nhiên: bootstrap, thứ tự batch, khởi tạo. Với solver xác định, đổi seed có thể cho đúng cùng dự đoán. Ta đo cả hai trường hợp: forest 40 cây với năm seed, và logistic regression với hai seed.

Python

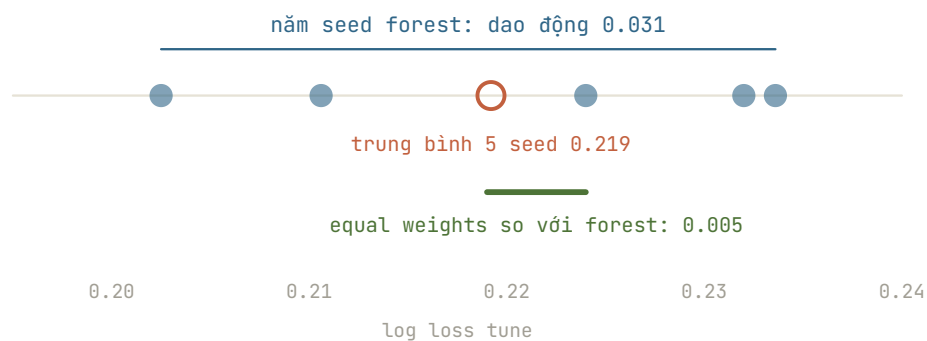
```
seed_probs = []
for seed in range(5):
    forest = RandomForestClassifier(n_estimators=40, max_depth=5,
                                   min_samples_leaf=3, random_state=seed, n_jobs=1)
    forest.fit(X_train, y_train)
    seed_probs.append(forest.predict_proba(X_tune))
seed_losses = [log_loss(y_tune, p) for p in seed_probs]
print("single seeds", [round(v, 4) for v in seed_losses])
print("mean of singles", f"{np.mean(seed_losses):.4f}",
      "seed average", f"{log_loss(y_tune, np.mean(seed_probs, axis=0)):.4f}")
lr_probs = [LogisticRegression(max_iter=500, random_state=s).fit(X_train, y_train)
             .predict_proba(X_tune) for s in (0, 1)]
print("logistic regression, two seeds identical:", np.array_equal(lr_probs[0], lr_probs[1]))
```

KẾT QUẢ

```
single seeds [0.224, 0.2336, 0.232, 0.2106, 0.2025]
mean of singles 0.2205 seed average 0.2192
```

Năm seed của cùng một cấu hình cho log loss tune từ 0.2025 đến 0.2336, dao động 0.031. Mức này lớn hơn sáu lần khoảng chênh 0.005 giữa equal weights và forest ở mục trước (0.2190 so với 0.2240), nên khoảng chênh ấy chưa phải bằng chứng. Trung bình năm bản cho 0.2192, sát trung bình các bản đơn (0.2205): gộp seed giảm dao động nhiều hơn là hạ mức lỗi.

Dao động giữa các seed là mốc để đọc mọi cải thiện nhỏ; chênh lệch dưới mốc ấy cần lặp nhiều seed trước khi kết luận. Logistic regression với solver mặc định cho cùng xác suất ở mọi seed, nên năm bản chỉ tốn thời gian chạy.



Chênh lệch giữa các seed lớn hơn chênh lệch giữa hai cách gộp.

Khi thời gian ít, kiểm hai seed nhỏ và so độ khác trước khi chạy nhiều bản. Nếu lỗi chung là đường tắt, đa dạng kiến trúc hoặc dữ liệu hợp lệ thường đáng khảo sát hơn thêm seed cùng công thức. Test-time augmentation (bài 5.3) là cùng ý ở phía đầu vào: một model, vài phiên bản hợp lệ của mẫu, trung bình dự đoán; chi phí chạy nhân lên theo số phiên bản.

HỎI NHANH

Hai cấu hình forest chênh nhau 0.004 log loss trên validation. Chạy lại mỗi cấu hình với 5 seed thì mỗi cấu hình dao động cỡ 0.03 giữa các seed. Ta kết luận được gì?

- A Chưa phân biệt được, cần so trung bình trên nhiều seed
- B Chọn cấu hình có điểm thấp hơn, vì mọi chênh lệch đều có ích
- C Chọn cấu hình có một seed đơn lẻ tốt nhất làm đại diện
- D Hai cấu hình giống nhau, nên nộp cấu hình nào cũng được

HỎI NHANH

Có năm bản LogisticRegression, với solver mặc định lbfgs, trên cùng dữ liệu, chỉ khác random_state. Trung bình xác suất của năm bản so với một bản thì thế nào?

- A Phương sai còn khoảng một phần năm, như với forest
- B Giống hệt một bản, vì solver chạy xác định
- C Tệ hơn, vì mỗi seed thêm nhiều khởi tạo
- D Khác về xác suất nhưng cho cùng nhãn

Stacking và dự đoán out-of-fold

Stacking dùng một model khác (meta-model) để học cách gộp. Feature của meta-model là dự đoán của các base model, nên mỗi mẫu train cần một dự đoán từ base model chưa fit trên mẫu ấy: dự đoán out-of-fold (OOF).

Blending là biến thể đơn giản hơn: giữ riêng một tập hold-out, fit base model trên phần còn lại và fit meta-model trên hold-out; không cần K lần fit, nhưng meta-model chỉ có

ít mẫu. Lưới trọng số ở mục trước là một blending mà meta-model chỉ có K tham số.

Ta thêm một cây quyết định không giới hạn độ sâu làm base model thứ tư vì nó khớp train 100%, làm rõ chỗ khác nhau giữa in-sample và OOF. Meta-model là logistic regression trên bốn xác suất lớp 1.

Python

```
from sklearn.model_selection import StratifiedKFold, cross_val_predict
from sklearn.tree import DecisionTreeClassifier

base_models = {**models, "tree": DecisionTreeClassifier(random_state=0).fit(X_train,
y_train)}
folds = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
in_sample, out_of_fold, tune_features = [], [], []
for name, model in base_models.items():
    p_in = model.predict_proba(X_train)[: , 1]
    p_oof = cross_val_predict(model, X_train, y_train, cv=folds,
                             method="predict_proba")[: , 1]

    in_sample.append(p_in)
    out_of_fold.append(p_oof)
    tune_features.append(model.predict_proba(X_tune)[: , 1])
    print(name, "acc in-sample", f"{accuracy_score(y_train, p_in >= 0.5):.3f}",
          "OOF", f"{accuracy_score(y_train, p_oof >= 0.5):.3f}")
in_sample, out_of_fold, tune_features = (
    np.column_stack(f) for f in (in_sample, out_of_fold, tune_features))

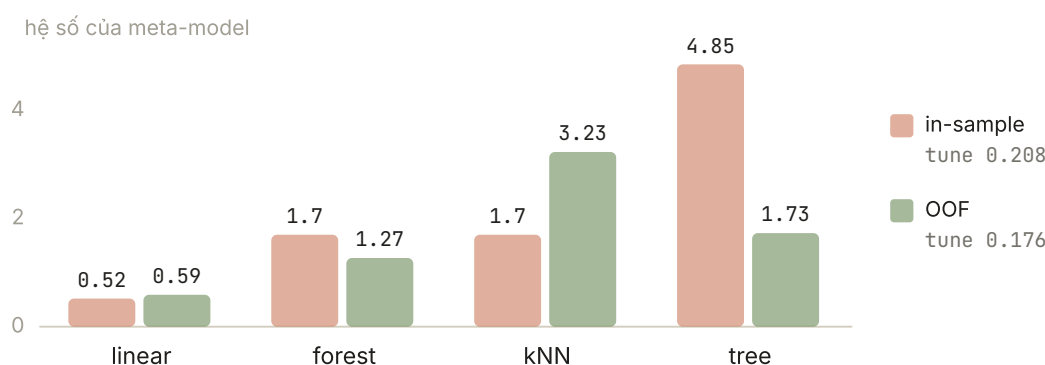
for label, features in [("in-sample", in_sample), ("OOF", out_of_fold)]:
    meta = LogisticRegression().fit(features, y_train)
    p_meta = meta.predict_proba(tune_features)
    print("meta", label, "tune logloss", f"{log_loss(y_tune, p_meta):.4f}")
    print("  coef", meta.coef_.round(2).tolist()[0])
```

KẾT QUẢ

```
linear acc in-sample 0.839 OOF 0.839
forest acc in-sample 0.961 OOF 0.925
knn acc in-sample 0.950 OOF 0.953
tree acc in-sample 1.000 OOF 0.922
meta in-sample tune logloss 0.2083
  coef [0.52, 1.7, 1.7, 4.85]
meta OOF tune logloss 0.1759
  coef [0.59, 1.27, 3.23, 1.73]
```

Cây đạt accuracy 1.000 trên train nhưng 0.922 khi mỗi mẫu được dự đoán bởi cây chưa thấy nó. Meta-model học trên in-sample thấy cây gần như hoàn hảo nên cho cây hệ số lớn nhất, 4.85; trên OOF, hệ số của cây chỉ còn 1.73 và kNN lên 3.23. Log loss tune là 0.2083 với in-sample và 0.1759 với OOF.

Một lần chia chưa đủ làm bằng chứng: đổi seed sinh dữ liệu ở ô đầu bài rồi chạy lại vài lần để thấy chiều chênh lệch có giữ nguyên không. Tập tune ở đây chỉ so hai cách tạo feature, meta-model không fit lên nó.



Meta-model học trên dự đoán in-sample tin quá mức vào cây quyết định; học trên OOF thì không.

Group và thời gian vẫn phải giữ đúng trong các fold: với bệnh nhân có nhiều ảnh, OOF theo từng ảnh vẫn cho base model thấy ảnh cùng người. Khi chạy test, base model là bản fit lại trên toàn train hoặc trung bình các bản của từng fold; meta-model nhận dự đoán của chúng. Với ít dữ liệu và ít giờ, mean hoặc một lưới trọng số nhỏ để kiểm hơn một meta-model chưa có đủ bằng chứng.

HỎI NHANH

Stacking có bốn base model, trong đó có một cây không giới hạn độ sâu. Meta-model được học trên dự đoán in-sample của các base model. Hệ quả có khả năng nhất là gì?

- A Cây bị giảm trọng số vì overfit, nên lỗi tự bù lại
- B Không ảnh hưởng gì, vì meta-model đơn giản hơn cây nhiều
- C Cây nhận hệ số lớn, rồi dự đoán kém ở mẫu mới
- D Meta-model nhận ra cây trùng dữ liệu train và loại nó đi

HỎI NHANH

Mỗi bệnh nhân có 6 ảnh gần giống nhau, và test là bệnh nhân mới. OOF của stacking được tạo bằng KFold xáo trộn theo từng ảnh. Meta-model có thể gặp vấn đề gì?

- A Không có vấn đề, vì không ảnh nào được dự đoán bởi model đã fit trên chính nó
- B OOF thiếu một số ảnh, nên meta-model bỏ qua những ảnh ấy
- C Meta-model overfit, vì chỉ có vài feature
- D Base model đã thấy ảnh của cùng bệnh nhân, nên OOF lạc quan

Chi phí chạy nhiều model

Chi phí inference thường gần tổng chi phí các model nếu chạy nối tiếp. Nếu nhiều model chia chung encoder, lưu sẵn (cache) embedding cắt phần lặp, đổi lại tốn đọc file và bộ nhớ.

Với encoder 40 giây và mỗi head 3 giây (số giả định cho phép tính), ba bản chạy encoder riêng mất $3(40 + 3) = 129$ giây, còn dùng một embedding chung mất $40 + 3 \cdot 3 = 49$ giây. Cache chỉ hợp lệ khi encoder, preprocessing và thứ tự mẫu thật sự giống nhau: chỉ cần một head dùng cách chuẩn hóa ảnh khác, embedding đã cache là đầu vào sai cho head ấy. Thời gian dùng để quyết định phải lấy từ lần đo trên đúng máy chấm, chưa phần dự phòng cho nạp trọng số và ghi file.

HỎI NHANH

Giới hạn chạy là 30 phút, và đội chưa 20% làm dự phòng. Đọc và tiền xử lý test mất 4 phút, làm một lần rồi dùng chung. Mỗi model mất thêm 1 phút nạp trọng số và 2 phút dự đoán. Chạy được tối đa bao nhiêu model?

A 3 model

B 6 model

C 7 model

D 8 model

Ghép đúng lớp và ID

Một bảng xác suất của model A có thứ tự lớp $[0, 1]$, model B có $[1, 0]$. Trước khi mean, ta phải đổi cột B cho khớp. Ví dụ sau không train thêm; nó thể hiện một lỗi cấu trúc mà shape không phát hiện.

Python

```
a_probability = np.array([[0.8, 0.2]]) # lớp [0, 1]
b_probability = np.array([[0.3, 0.7]]) # lớp [1, 0]
b_classes = [1, 0]
columns = [b_classes.index(c) for c in [0, 1]]
wrong = (a_probability + b_probability) / 2
correct = (a_probability + b_probability[:, columns]) / 2
print("wrong", wrong.round(2).tolist())
```

```
print("aligned", correct.round(2).tolist())
print("row sum", correct.sum(axis=1).tolist())
```

KẾT QUẢ

```
wrong [[0.55, 0.45]]
aligned [[0.75, 0.25]]
row sum [1.0]
```

Tổng hàng của bản sai vẫn là 1, nên kiểm tổng xác suất chưa đủ. Tương tự, các ID phải có cùng tập và cùng thứ tự sau ghép; kiểm không trùng, không thiếu rồi reindex theo thứ tự chuẩn.

Với pandas, cộng hai Series tự căn theo nhãn index, còn `.to_numpy()` hoặc `.values` bỏ nhãn và cộng theo vị trí. Khi nộp nhiều model, lưu config trọng số cùng artifact; đừng để trọng số chỉ nằm trong một cell RAM chưa được chạy lại.

HỎI NHANH

Hai bảng xác suất có cùng shape, và tổng mỗi hàng đều bằng 1. Đã lấy trung bình ngay được chưa?

- A Được, vì shape của hai bảng đã khớp
- B Được, nếu hai model dùng cùng một thư viện
- C Được, nếu hai model có cùng accuracy
- D Chưa, phải kiểm ID, thứ tự mẫu và thứ tự lớp

Khi đi thi

- Trước khi thêm model vào ensemble, xem nó sửa được mẫu nào mà bản hiện có sai.
- Chọn cách gộp theo metric: trung bình xác suất cho log loss, trung bình hạng cho AUC, bỏ phiếu cho nhãn.

- Ghép theo ID và thứ tự lớp bằng code; giữ bản model đơn tốt nhất làm dự phòng.

Tóm tắt

- Ensemble có lợi khi các model sai ở những chỗ khác nhau; tương quan sai số chặn mức lợi.
- Chọn cách gộp theo metric: bỏ phiếu cho nhãn, trung bình xác suất cho log loss, trung bình hạng cho AUC.
- Chọn trọng số trên một tập, kiểm phương án đã chốt trên tập khác.
- Ghép đúng ID và đúng thứ tự lớp; đo cả thời gian chạy.
- Nhiều seed giảm dao động, không sửa được lỗi chung như rò rỉ hay đường tắt; stacking cần dự đoán out-of-fold.

Kiểm tra cuối bài

CÂU 1

Một cây quyết định không cắt tỉa đạt 100% trên train nhưng chỉ 78% trên validation, và con số validation đổi hơn 5 điểm khi đổi bootstrap. Lấy trung bình 50 cây như vậy, mỗi cây học trên một bootstrap riêng, chủ yếu thay đổi điều gì?

- A Giảm variance, còn bias của từng cây gần như giữ nguyên
- B Giảm bias, còn variance của từng cây gần như giữ nguyên
- C Giảm cả bias lẫn variance, vì mỗi cây thấy dữ liệu khác nhau
- D Không đổi gì, vì mọi cây học từ cùng một nguồn dữ liệu

PHÒNG THEO [NOAI 2025 Singapore · Phần 1 · Câu 2](#)

CÂU 2

Logistic regression trên dữ liệu hai bán nguyệt cho 0.844 trên train và 0.846 trên validation, tức cả hai cùng thấp. Lấy trung bình 30 bản học trên các bootstrap khác nhau vẫn cho 0.842 trên validation. Nên giảm lỗi theo hướng nào?

- A Tăng bagging lên 300 bản để giảm dao động thêm nữa
- B Tăng L2 để các bản bootstrap bớt khác nhau
- C Dừng boosting với cây nông, hoặc thêm đặc trưng phi tuyến
- D Loại những bản bootstrap có accuracy thấp nhất

PHÒNG THEO [NOAI 2026 Singapore](#) · Vòng sơ loại · Câu 166

CÂU 3

Mười lăm bộ phân loại nhị phân độc lập, mỗi bộ đúng 45% trên một bài hai lớp cân bằng, được gộp bằng bỏ phiếu đa số. Một bạn nói “càng nhiều model càng tốt”. Accuracy sau bỏ phiếu xấp xỉ bao nhiêu?

- A 0.65
- B 0.55
- C 0.45
- D 0.35

PHÒNG THEO [CMU 10-601](#) · Exam 3 Practice · Q6.1.4

CÂU 4

Có 400 mẫu có nhãn. Một đội làm blending: 320 mẫu dùng để fit sáu base model, 80 mẫu còn lại dùng để fit một meta-model nhận sáu xác suất, tức bảy hệ số. Rủi ro chính là gì?

- A Meta-model chỉ có sáu feature nên không học được gì
- B 80 mẫu cho bảy hệ số là ít, nên trọng số học theo nhiễu
- C Base model chưa thấy 80 mẫu ấy, nên chúng làm rò nhãn
- D Blending luôn bị rò rỉ, chỉ OOF mới hợp lệ

CÂU 5

Hai model chấm bốn mẫu a, b, c, d , điểm càng cao càng nghiêng về lớp dương. Model A cho 0.51, 0.50, 0.52, 0.53. Model B cho 0.10, 0.90, 0.20, 0.30. Lấy trung bình điểm thô thì mẫu nào cao nhất, và lấy trung bình hạng thì mẫu nào cao nhất?

- A Điểm thô thì b , hạng thì d
- B Điểm thô thì d , hạng thì b
- C Điểm thô thì b , hạng cũng b
- D Điểm thô thì d , hạng cũng d

CÂU 6

Ba model trả phân phối trên ba lớp (a, b, c). M1 cho (0.40, 0.35, 0.25), M2 cho (0.10, 0.50, 0.40), M3 cho (0.45, 0.30, 0.25). Hard voting và soft voting, tức trung bình đều, chọn lớp nào?

- A Hard voting chọn b , soft voting chọn b
- B Hard voting chọn b , soft voting chọn a
- C Hard voting chọn a , soft voting chọn b
- D Hard voting chọn a , soft voting chọn a

CÂU 7

Xét đoạn code sau.

```
a = pd.Series([0.9, 0.2, 0.6], index=["id3", "id1", "id2"])
b = pd.Series([0.1, 0.8, 0.5], index=["id1", "id2", "id3"])
x = (a + b) / 2
y = (a.to_numpy() + b.to_numpy()) / 2
```

Gộp đều hai bản thì xác suất đúng của id3 là bao nhiêu, và đọc nó ở đâu?

- A 0.50, đọc ở $y[0]$
- B 0.70, đọc ở $x["id3"]$
- C 0.55, đọc ở $y[2]$
- D 0.15, đọc ở $x["id1"]$

CÂU 8

Một đội cache embedding của một encoder ảnh đóng băng, tạo từ ảnh đã chuẩn hóa về $[0, 1]$, rồi train ba head trên cache ấy. Head thứ ba được thiết kế cho embedding sinh từ ảnh chuẩn hóa bằng mean và std. Dùng chung cache cho head này thì sao?

- A Hợp lệ, vì encoder đóng băng nên embedding không đổi
- B Hợp lệ, nếu thứ tự ảnh trong cache khớp với head
- C Chỉ không hợp lệ khi head thứ ba có kiến trúc khác
- D Không hợp lệ, vì đầu vào khác thì embedding cũng khác

CÂU 9

Ba model fine-tune từ cùng một backbone đều học được rằng viên đỏ gắn với lớp 1 ở camera cũ. Validation chia ngẫu nhiên đạt 0.95, còn ảnh từ camera mới chỉ được 0.62. Chạy thêm 10 seed cho mỗi model rồi lấy trung bình, thì kết quả trên camera mới sẽ ra sao?

- A Vẫn quanh 0.62, bất dao động hơn, còn đường tắt chung vẫn còn đó
- B Tăng vừa phải, vì mười seed có mười đường tắt khác nhau
- C Tăng lên gần 0.95, vì nhiều seed bù được mọi lỗi
- D Không đổi chút nào, vì seed không ảnh hưởng tới fine-tuning

CÂU 10

Hai model bất đồng về nhãn trên 30% validation. Chỉ từ tỉ lệ này, ta kết luận được gì?

- A Sai số của hai model độc lập hoàn toàn với nhau
- B Ensemble của hai model tăng accuracy ít nhất 30%
- C Hai model khác nhau nhiều, nhưng chưa rõ ensemble có giúp không
- D Model thứ hai sai trên mọi mẫu mà hai model bất đồng

Đọc thêm

- [IOAI 2026 Syllabus](#). Model ensembles thuộc phần thực hành; đọc cùng phần evaluation và cross-validation.
- [scikit-learn 1.7 – VotingClassifier](#). Đọc hard/soft voting, weights và thứ tự lớp.
- [scikit-learn 1.7 – Ensembles](#), phần voting và stacking. Đọc cơ chế dùng dự đoán cross-validation cho meta-model.
- James và cộng sự – [An Introduction to Statistical Learning](#), chương 8, bagging và random forests. Đọc trực giác giảm dao động và đa dạng hóa cây.
- [scikit-learn – Probability calibration](#). Đọc vì sao một con số 0.9 cần được kiểm cách hiệu chỉnh trước khi gộp theo độ tự tin.
- [SciPy – scipy.stats.rankdata](#). Đọc cách xử lý điểm bằng nhau (method) trước khi trung bình hạng.