Chương 7: Mô hình máy học kết hợp (ensemble learning)

(Tài liệu nội bộ)

Tháng 6 năm 2020

Nội dung trình bày

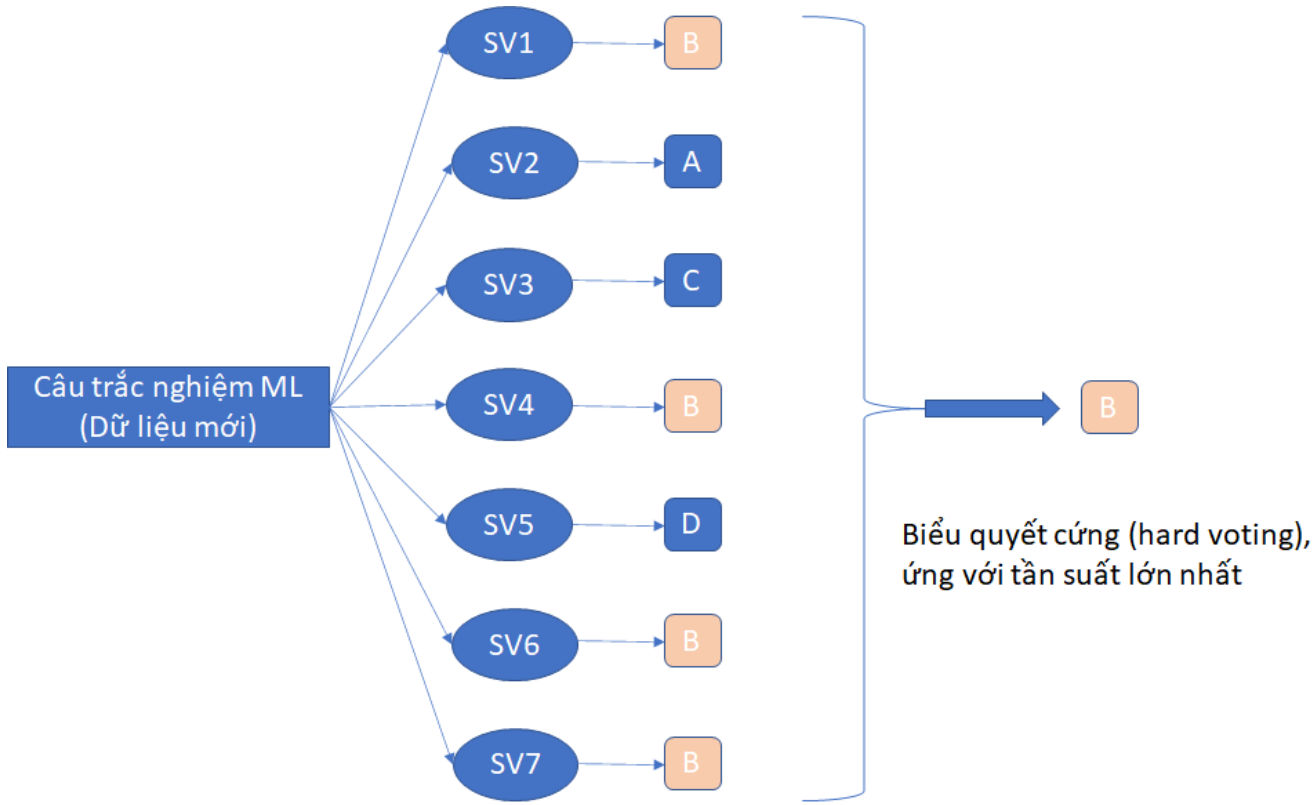
1. Mở đầu
2. Phương pháp Bagging và Pasting
3. Nhóm ngẫu nhiên (random patches) và không gian con ngẫu nhiên (random subspaces)
4. Rừng ngẫu nhiên (Random forests)
5. Boosting (kỹ thuật học hỗ trợ)
6. Stacking (Xếp chồng)

Nội dung trình bày

1. Mở đầu

Trí tuệ đám đông

- Có hẳn một cuốn sách nói về điều này. Cuốn sách có tên: Trí Tuệ Đám Đông: vì Sao Đa Số Thông Minh Hơn Thiểu Số của tác giả James Surowiecki-2004
- Đại ý của trí tuệ đám đông là: Nếu bạn hỏi một câu hỏi phức tạp (khó) cho cả ngàn người được chọn ngẫu nhiên rồi tổng hợp câu trả lời của họ thì trong đa số trường hợp bạn sẽ nhận được câu trả lời tốt hơn câu trả lời của một chuyên gia.



- Trong máy học, nếu ta kết hợp các mô hình học máy (ta sẽ gọi là mô hình cơ sở) với nhau một cách hợp lý, tức là các mô hình có thể khắc phục các điểm yếu của nhau, hoặc bổ sung cho nhau, thì nhiều khả năng ta sẽ đạt được độ chính xác cao hơn so với việc áp dụng các mô hình đơn lẻ
- Kỹ thuật kết hợp các mô hình cơ sở được gọi là mô hình máy học kết hợp (ensemble learning), và thuật toán của mô hình này được gọi là phương pháp kết hợp (Ensemble method)
- Phương pháp máy học kết hợp sẽ hiệu quả nhất khi các mô hình cơ sở dự đoán độc lập nhau.
  Để có các mô hình cơ sở đa dạng, ta có thể huấn luyện các mô hình cơ sở bằng các thuật toán khác nhau. Điều này làm tăng khả năng sửa các lỗi khác nhau, qua đó làm tăng độ chính xác của mô hình kết hợp.
- Trong thực tế, các giải pháp chiến thắng trong các cuộc thi máy học thường liên quan đến các phương pháp học kết hợp - nổi tiếng nhất là cuộc thi giải thưởng Netflix (Netflix Prize competition)

Cải thiện độ chính xác của mô hình cơ sở

The following code creates and trains a voting classifier in Scikit-Learn, composed of three diverse classifiers (the training set is the moons dataset, introduced in Chapter 5):

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='hard')
voting_clf.fit(X_train, y_train)
```

Let's look at each classifier's accuracy on the test set:

```
>>> from sklearn.metrics import accuracy_score
>>> for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
...     clf.fit(X_train, y_train)
...     y_pred = clf.predict(X_test)
...     print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
...
LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.888
VotingClassifier 0.904
```

Hình 2

Nội dung trình bày

❷ Phương pháp Bagging và Pasting

- Một kỹ thuật học kết hợp sử dụng một hoặc vài thuật toán để huấn luyện mô hình máy học nhưng trên các tập dữ liệu con được chọn ngẫu nhiên từ tập huấn luyện.
- Nếu tập con dùng huấn luyện được chọn có hoàn lại (tức là chọn phần tử vào tập con có sự trùng lại) thì phương pháp được gọi là **bagging** (viết tắt của bootstrap aggregating)
- Nếu tập con được chọn không hoàn lại thì phương pháp được gọi là **pasting**
- Khi có dữ liệu mới, mô hình thường dự đoán bằng giá trị **mode** (giá trị ứng với tần suất lớn nhất), ta gọi cách chọn này là biểu quyết cứng (hard voting/ majority vote),
- Trong trường hợp các mô hình cơ sở dự đoán xác suất của tất cả lớp nhãn thì ta sẽ tính trung bình cộng của xác suất của từng lớp nhãn rồi lấy xác suất có giá trị lớn nhất. Cách này được gọi là biểu quyết mềm (soft-voting). Phương pháp biểu quyết ở trên được chứng minh bằng **luật số lớn** trong lý thuyết xác suất.

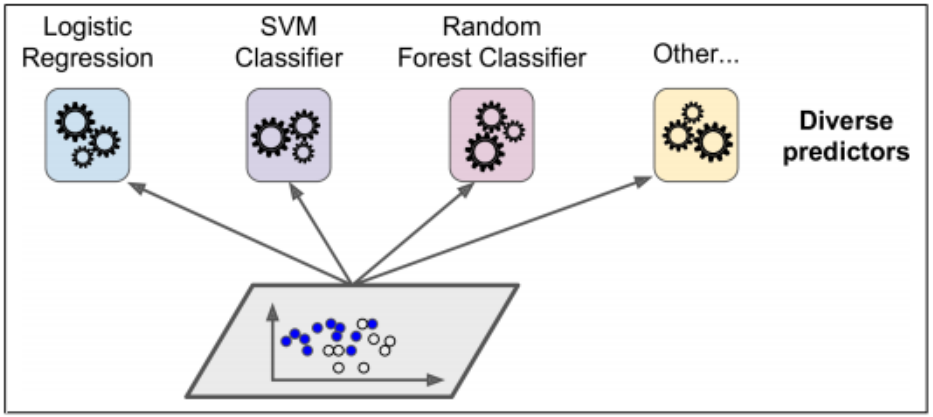


Figure 7-1. Training diverse classifiers

Hình 3

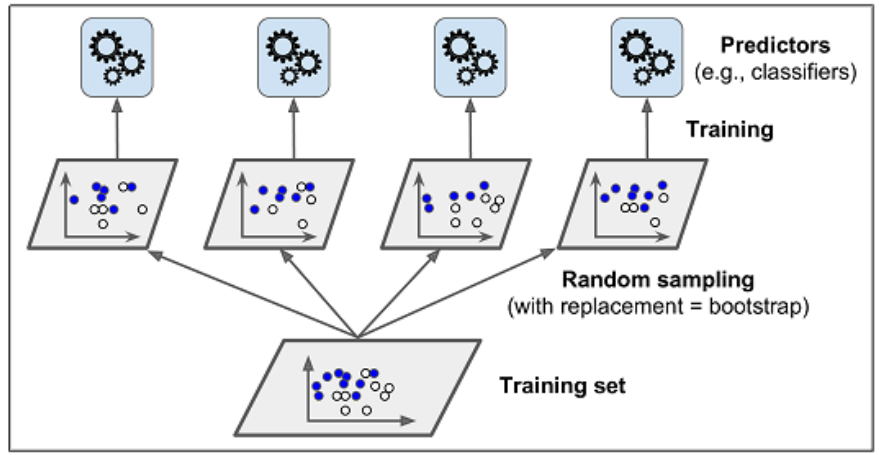


Figure 7-4. Pasting/bagging training set sampling and training

Hình 4

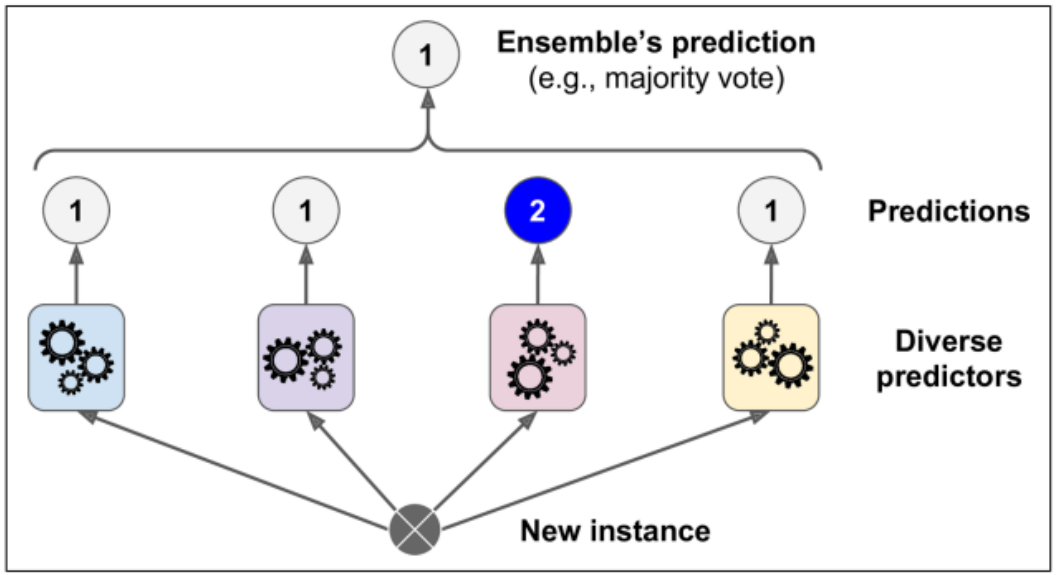
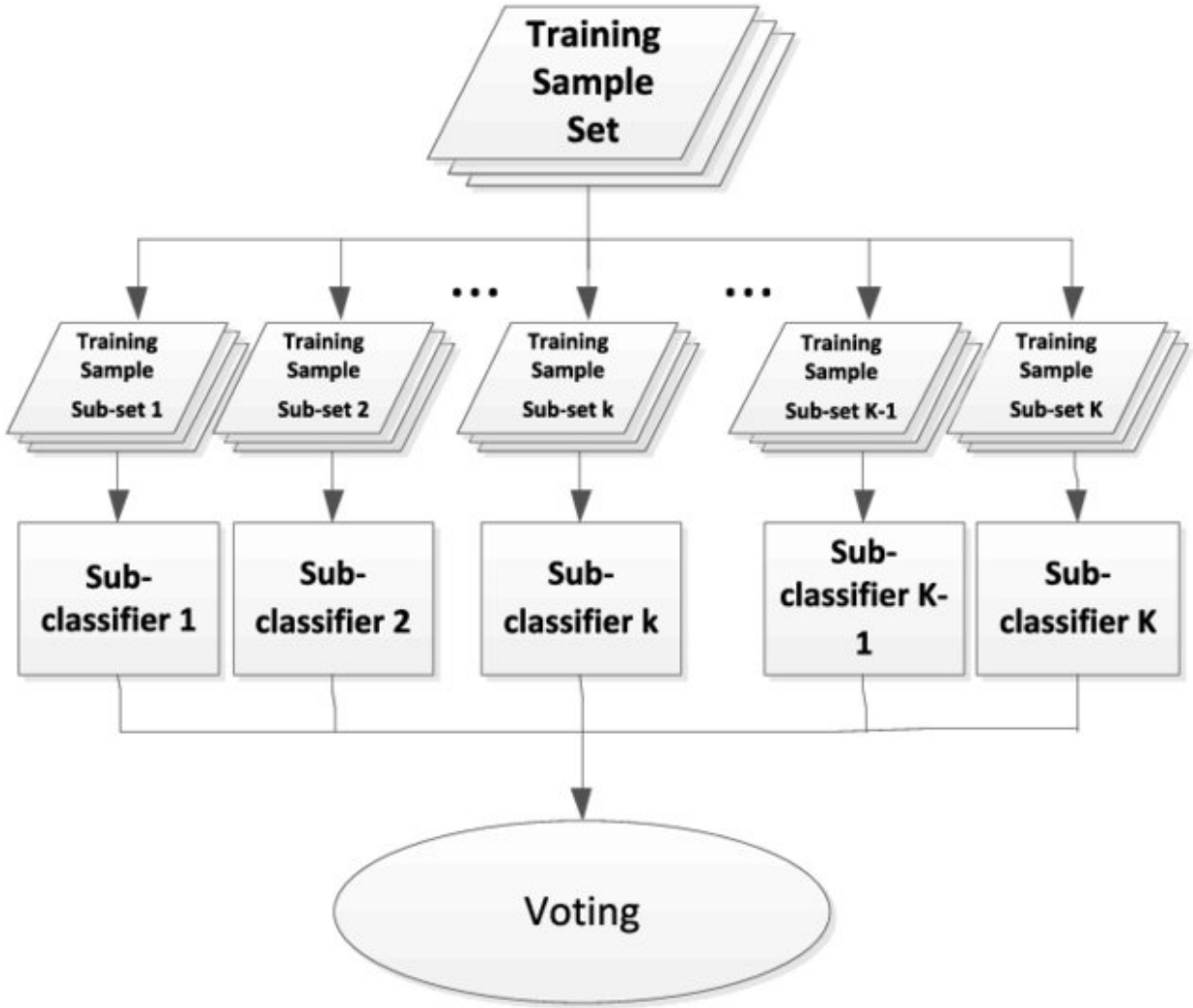


Figure 7-2. Hard voting classifier predictions

Hình 5

Mô hình thuật toán Bagging/pasting



Hình 6

Khi nào dùng mô hình kết hợp

- Tăng độ chính xác của các mô hình có độ chính xác chưa cao (tạm gọi là mô hình yếu)
- Giảm hiện tượng quá khớp (overfitting) trong những mô hình phức tạp

Sinh viên thực hành đoạn code trang 194,196

Ghi chú:

- lệnh BaggingClassifier tự động thực hiện biểu quyết mềm nếu bộ phân lớp cơ sở cho dự đoán xác suất, như trường hợp dùng cây quyết định.
- Nếu đặt tham số *bootstrap=True (default)(lặp lại)* thì ta có phương pháp Bagging, nếu *bootstrap=False* thì ta có phương pháp Pasting.
- Tham khảo ý nghĩa các tham số của BaggingClassifier tại https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

Out of Bag evaluation

- Trong phương pháp bagging, một điểm dữ liệu có thể được chọn làm mẫu huấn luyện nhiều lần, trong khi có thể có các điểm dữ liệu khác lại không được chọn lần nào. Người ta chứng minh được rằng nếu m là kích thước tập huấn luyện thì khi m tăng đủ lớn, số điểm dữ liệu được chọn khoảng chừng $1 - 1/e = 63\%$. 37% còn lại được gọi là các điểm dữ liệu OOB (out -of- bag), là các điểm chưa được chọn lần nào.
- Trong quá trình huấn luyện ta có thể sử dụng tập OOB này để đánh giá mô hình mà không cần tách dữ liệu thành tập thẩm định (validation set) riêng biệt.
- Trong Scikit-Learn, ta đánh giá tập OOB bằng cách đặt *oob_score=True*
- Thực hành đoạn code trang 197, 198.
  Lệnh *bag_clf.oob_score_* cho dự đoán độ chính xác của mô hình khoảng 90.1%, khá gần khi kiểm chứng mô hình bằng tập kiểm tra (91.2%)

Nội dung trình bày

❸ Nhóm ngẫu nhiên (random patches) và không gian con ngẫu nhiên (random subspaces)

Đây là hai phương pháp tương tự như phương pháp bagging, nhưng có sự khác biệt trong chọn ngẫu nhiên phần tử vào tập huấn luyện con, đó là có kết hợp chọn ngẫu nhiên thuộc tính vào tập huấn luyện. Phương pháp này thường được sử dụng khi ta muốn giới hạn số chiều của không gian thuộc tính trong quá trình huấn luyện.

- Phương pháp nhóm ngẫu nhiên chọn phần tử để huấn luyện dựa trên không những tập dữ liệu huấn luyện mà còn dựa trên tập thuộc tính.
  VD: Với tập huấn luyện có N phần tử với số thuộc tính là M, ta chọn ngẫu nhiên n<N phần tử và m<M thuộc tính để huấn luyện mô hình kết hợp.
- Phương pháp không gian con ngẫu nhiên chọn phần tử để huấn luyện chỉ dựa trên chọn ngẫu nhiên các thuộc tính (giữ lại toàn bộ tập dữ liệu huấn luyện)
- Kiểm soát việc chọn phần tử dựa vào thuộc tính trong BaggingClassifier bằng hai siêu tham số
  max_features=số thuộc tính được chọn từ tập dữ liệu để huấn luyện mô hình cơ sở (mặc định là 1.0- tức là 100% thuộc tính)
  bootstrap_features: Mặc định là ”False= lấy thuộc tính có lặp lại, true= không lặp lại.

GIẢI THUẬT CHO PHƯƠNG PHÁP KHÔNG GIAN CON NGẪU NHIÊN

Giả sử X tập huấn luyện với m thuộc tính. r là số thuộc tính được rút ra từ m thuộc tính để huấn luyện mô hình cơ sở. B là số mô hình cơ sở. k là số nhãn.
Thuật giải:

1 Lặp với $b = 1, ..., B$:

a Chọn không gian con ngẫu nhiên r chiều $\bar{X}^b$ từ không gian thuộc tính X-m chiều.

b Huấn luyện mô hình $M^b(x)$ với tập huấn luyện $\bar{X}^b$

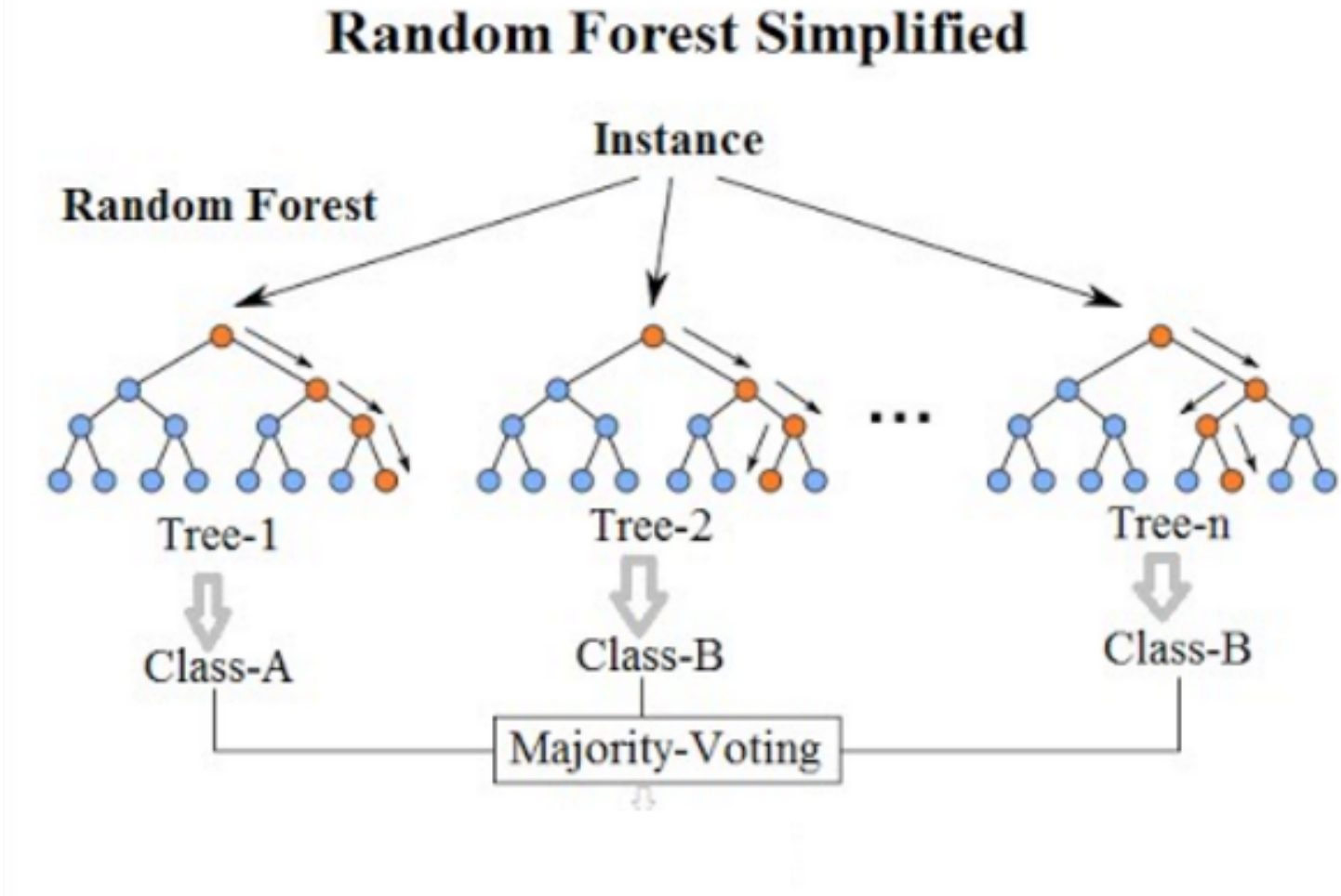
2 Kết hợp các mô hình $M^b(x)$ để ra được mô hình cuối cùng với phương thức quyết định dựa trên biểu quyết (voting)

Thuật giải cho phương pháp nhóm ngẫu nhiên được thực hiện tương tự, ta thêm phần chọn số phần tử của mẫu vào tập huấn luyện ở bước (1.a)

Nội dung trình bày

❹ Rừng ngẫu nhiên (Random forests)

Bài toán mở đầu: Giả sử bạn muốn đi tham quan du lịch miền Trung và có sự cân nhắc cho việc tham quan thành phố nào như: Đà Nẵng, Hội An, Huế, Đồng Hới. Để trả lời câu hỏi này bạn sẽ cần tham khảo rất nhiều ý kiến từ bạn bè, blog du lịch, tour lữ hành … Mỗi một ý kiến tương ứng với một Cây quyết định trả lời các câu hỏi như: thành phố này đẹp không, có được tham quan các di tích lịch sử quốc gia khi đến thăm không, số tiền bỏ ra là bao nhiêu, phương tiện di chuyển?, điều kiện khách sạn?, thời gian để tham quan thành phố?…
Kết quả tham khảo được sẽ là ”một rừng” các câu trả lời.
Như vậy, mong muốn và điều kiện của bạn sẽ là điểm dữ liệu mới và dựa trên kết quả tham khảo để quyết định xem nên đi tham quan thành phố nào.

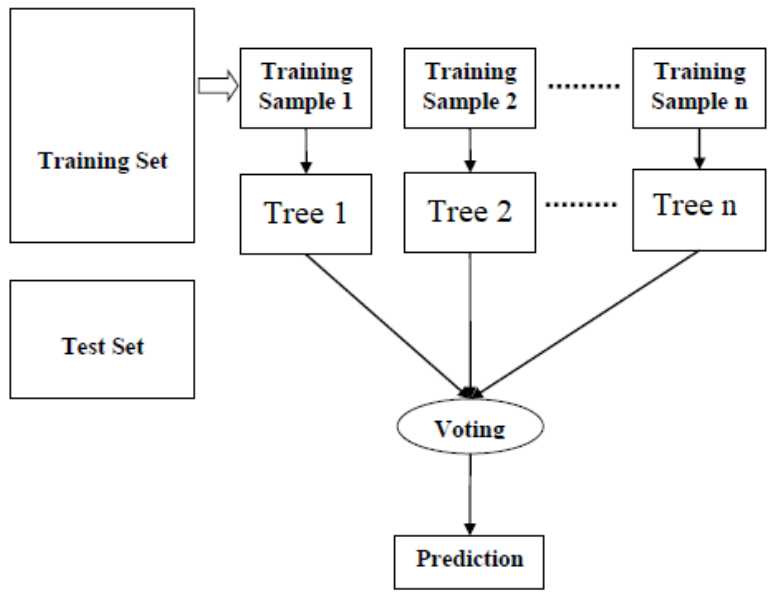

Random Forest Simplified
Instance
Random Forest
Tree-1
Tree-2
Tree-n
Class-A
Class-B
Class-B
Majority-Voting

Hình 7

- Rừng ngẫu nhiên là một thuật toán có vai trò quan trọng trong những thuật toán học có giám sát
- Rừng ngẫu nhiên là một phương pháp học kết hợp các cây quyết định được huấn luyện theo kỹ thuật bagging (hoặc pasting)
- Thuật toán rừng ngẫu nhiên có thể sử dụng cho cả hai bài toán: phân loại (Classification) và hồi quy (Regression)
- Rừng ngẫu nhiên làm việc được với dữ liệu thiếu giá trị
- Rừng càng có nhiều cây thì ta càng có cơ hội tránh được hiện tượng quá khớp
- Rừng ngẫu nhiên hoạt động bằng cách cho điểm dữ liệu mới được đánh giá/phân loại qua nhiều cây quyết định, và lấy ra kết quả được đánh giá tốt nhất trong số kết quả trả về

Thuật toán rừng ngẫu nhiên

1. Xây dựng rừng ngẫu nhiên. Lặp từ $k = 1, \ldots, n$
   - Chọn ngẫn nhiên tập con X^k từ tập huấn luyện
   - Xây dựng cây quyết định T^k trên tập huấn luyện X^k
2. Đưa vào điểm dữ liệu mới. Tính toán kết quả của các cây.
3. Biểu quyết và Dự đoán kết quả



Hình 8

SV thực hành đoạn code trang 197, 198 và giải thích ý nghĩa các tham số

- n_estimators
- max_leaf_nodes
- n_jobs
- splitter=”random”
- max_samples
- bootstrap=True

Extra-trees

- Ta có thể tạo các cây ngẫu nhiên hơn nữa bằng cách dùng các điểm tách (threshold) ngẫu nhiên cho mỗi thuộc tính hơn là tìm điểm tách tốt nhất (như khi tạo cây quyết định). Rừng ngẫu nhiên được tạo theo kỹ thuật này được gọi là *Extra-Trees (Extremely Randomized Trees)*
- Kỹ thuật này chấp nhận mức độ *lỗi chênh lệch* (bias) cao để có được mức độ *lỗi quá nhạy* (variance) thấp .
- Extra-Trees chạy nhanh hơn giải thuật rừng ngẫu nhiên (vì việc tìm điểm tách tốt nhất cho mỗi thuộc tính là việc tốn khá nhiều thời gian)
- So sánh ưu điểm của bộ phân loại dựa trên rừng ngẫu nhiên và dựa trên Extra-Trees là việc khó, thường ta phải thực hiện cả hai và so sánh chúng dựa vào đánh giá chéo (cross-validation)

Sự được mất giữa độ lệch và độ nhạy của mô hình (nhắc lại)

(Chương 4) Lỗi của một mô hình máy học có thể do 3 nguyên nhân sau:

- **Lỗi chênh lệch - độ lệch (Bias)**: Lỗi do giả định sai về dữ liệu, chẳng hạn chúng ta cho rằng dữ liệu phân bố theo hàm tuyến tính nhưng thực tế dữ liệu phân bố theo hàm bậc hai. Mô hình có lỗi chênh lệch cao sẽ có khả năng cao gây ra hiện tượng *chưa khớp dữ liệu*.
- **Lỗi quá nhạy - độ nhạy (Variance)**: Lỗi do mô hình quá nhạy với những hiện tượng nhỏ, đặc biệt trong dữ liệu huấn luyện. Những mô hình có độ tự do cao (như hàm đa thức bậc cao) có thể có lỗi quá nhạy cao dẫn đến *quá khớp dữ liệu*.
- **Lỗi bất khả quy (Irreducible error)**: Lỗi do dữ liệu huấn luyện chứa nhiều nhiễu. Cách duy nhất để giảm lỗi dạng này là làm sạch dữ liệu (Xem chương 2).

Khi ta tăng độ phức tạp của mô hình thì thường đồng nghĩa với tăng độ nhạy và giảm độ lệch, và ngược lại.

Tầm quan trọng của thuộc tính

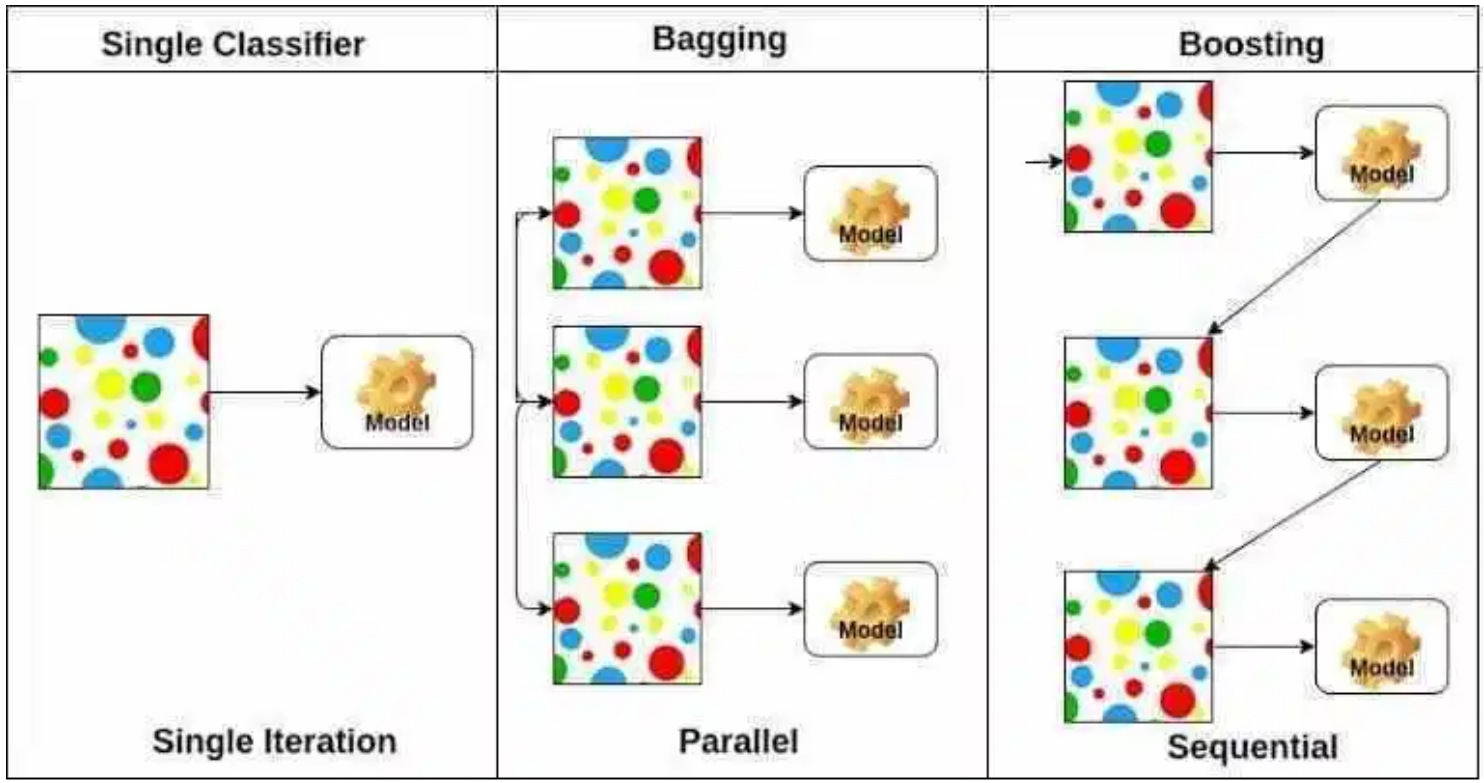
- Thuật toán Rừng ngẫu nhiên có một điểm tuyệt vời, đó là nó giúp ta dễ dàng đo lường tầm quan trọng (tương đối) của từng thuộc tính. Ta thực hiện điều này bằng cách theo dõi số lần các nút của cây sử dụng thuộc tính làm giảm tính không thuần khiết (impurity) (tính theo trung bình có trọng số) Scikit-Learn đo tầm quan trọng của thuộc tính qua
- *feature_importances* -là tham số đo tầm quan trọng của một thuộc tính
- Kết quả ở trang 201: sepal length (cm) 0.112; sepal width (cm) 0.023; petal length (cm) 0.441; petal width (cm) 0.423 cho ta biết thuộc tính quan trọng nhất là ”petal length” (44.1%)

THỰC HÀNH ĐOẠN CODE TRANG 201

Nội dung trình bày

5. Boosting (kỹ thuật học hỗ trợ)

- Boosting là một kỹ thuật học kết hợp có mục tiêu tạo ra một bộ phân lớp mạnh từ một số các bộ phân lớp yếu hơn (nhưng độ chính xác cũng phải trên 50%).
- Ý tưởng của kỹ thuật boosting là: xây dựng một mô hình (gọi là mô hình cơ sở) từ dữ liệu huấn luyện, sau đó ta tiếp tục huấn luyện mô hình cơ sở này và cố gắng sửa các lỗi từ mô hình đầu tiên, các mô hình cơ sở tiếp tục được thêm vào cho đến khi tập huấn luyện được dự đoán hoàn hảo hoặc số lượng mô hình cơ sở đạt ngưỡng cực đại.



Hình 9

AdaBoost- Adaptive Boosting (Kỹ thuật học hỗ trợ tương thích)

AdaBoost (Adaptive Boost) là một kỹ thuật học giúp đẩy nhanh việc tạo ra một mô hình mạnh bằng cách học tuần tự qua các mô hình cơ sở. Mô hình huấn luyện tiếp theo khắc phục các lỗi của mô hình trước bằng cách chú ý đến các điểm dữ liệu mà mô hình trước dự đoán sai qua các trọng số, đó là: *tăng trọng số cho các dữ liệu được dự đoán sai và giảm trọng số cho dữ liệu được dự đoán đúng*. Kỹ thuật huấn luyện này giúp các điểm dữ liệu khó (là các điểm mà hay bị dự đoán sai) sẽ dần được dự đoán đúng.

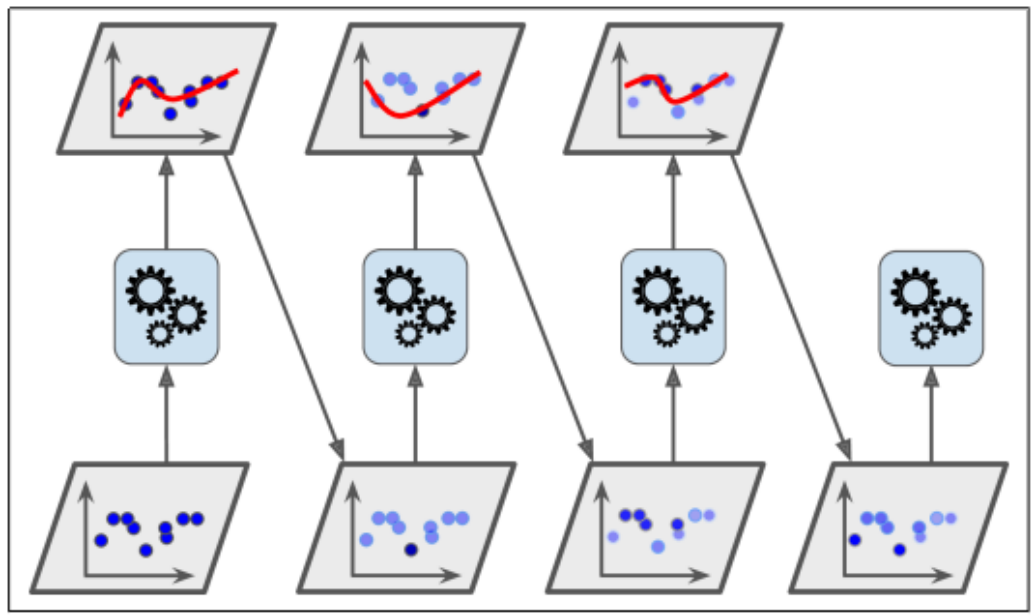
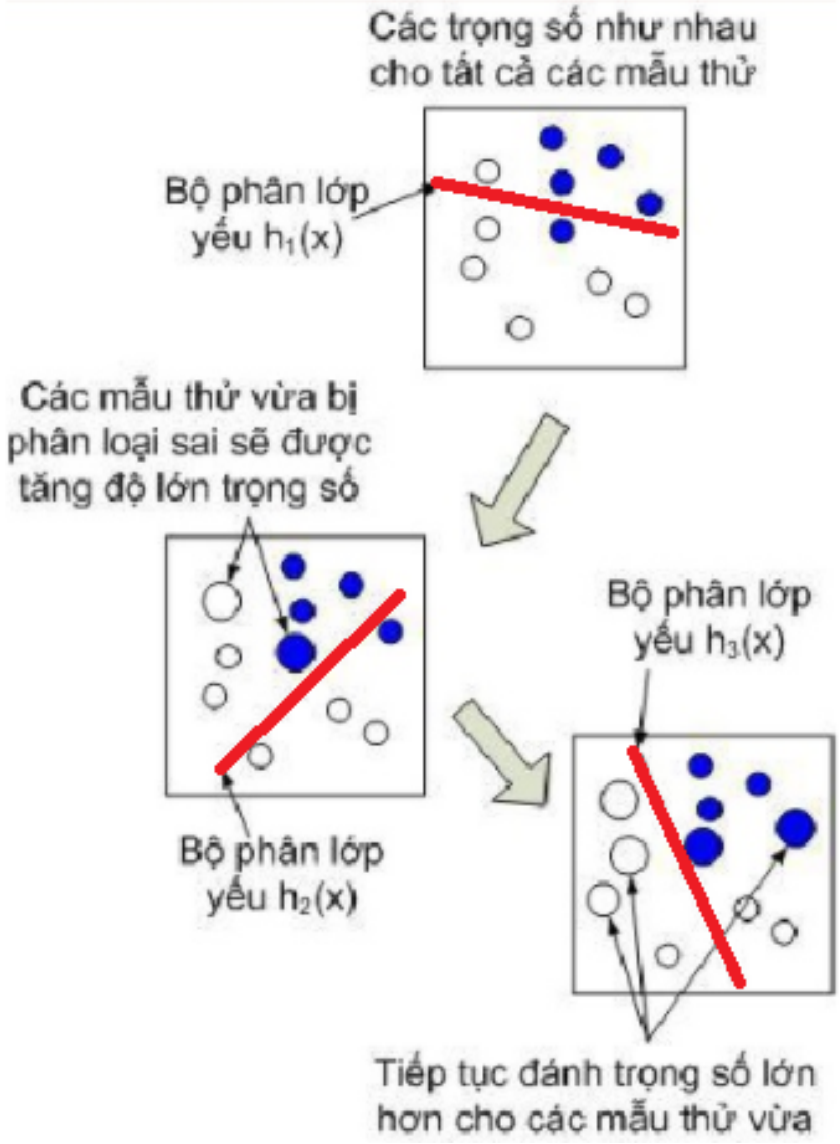


Figure 7-7. AdaBoost sequential training with instance weight updates

Hình 10: AdaBoost với bài toán hồi quy



Hình 11: AdaBoost với bài toán phân lớp

THUẬT GIẢI AdaBoost

1 Mỗi trọng số của điểm dữ liệu được khởi tạo $w^{(i)} = \frac{1}{m}$, m là kích thước tập huấn luyện. Mô hình đầu tiên được huấn luyện và có tỉ lệ dự đoán sai có trọng số là r_1 được tính trên tập huấn luyện. Công thức tính tỉ lệ dự đoán sai r_j ở mô hình cơ sở thứ j là

$$r_j = \frac{\sum_{i=1\ldots m, \hat{y}_j^{(i)} \neq y^{(i)}} \mathrm{w}^{(i)}}{\sum_{i=1}^{m} \mathrm{w}^{(i)}} \tag{1}$$

với $\hat{y}_j^{(i)}$ là giá trị dự đoán của mô hình thứ j cho điểm dữ liệu thứ i

Nhận xét: r_j chính là xác suất mô hình j dự đoán sai

2 Trọng số α_j của mô hình thứ j được gán sao cho mô hình càng chính xác thì trọng số càng cao, nếu kết quả dự đoán chỉ là ngẫu nhiên thì trọng số này sẽ gần 0:

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}, \eta \text{ là tốc độ học} \tag{2}$$

THUẬT GIẢI AdaBoost (tt)

3 Trọng số của các điểm dữ liệu thứ i ($i = 1, \ldots, m$) được cập nhật trong mô hình thứ j theo công thức

$$\mathrm{w}^{(i)} = \begin{cases} \mathrm{w}^{(i)} \ , & \text{Nếu } \hat{y}_j^{(i)} = y^{(i)} \\ \mathrm{w}^{(i)} \exp\left(\alpha_j\right) \ , & \text{Nếu } \hat{y}_j^{(i)} \neq y^{(i)} \end{cases} \tag{3}$$

Các trọng số $w^{(i)}$ tiếp tục được chuẩn hóa (tổng bằng 1) bằng cách đem chia cho $\sum_{i=1}^{m} \mathrm{w}^{(i)}$

4 Ta lặp lại bước 2,3 để huấn luyện mô hình tiếp theo dựa trên các trọng số của dữ liệu đã được cập nhật.

5 Thuật toán dừng lại khi ta đạt được số mô hình cơ sở xác định trước, hoặc khi mô hình đủ mạnh như ta mong muốn.

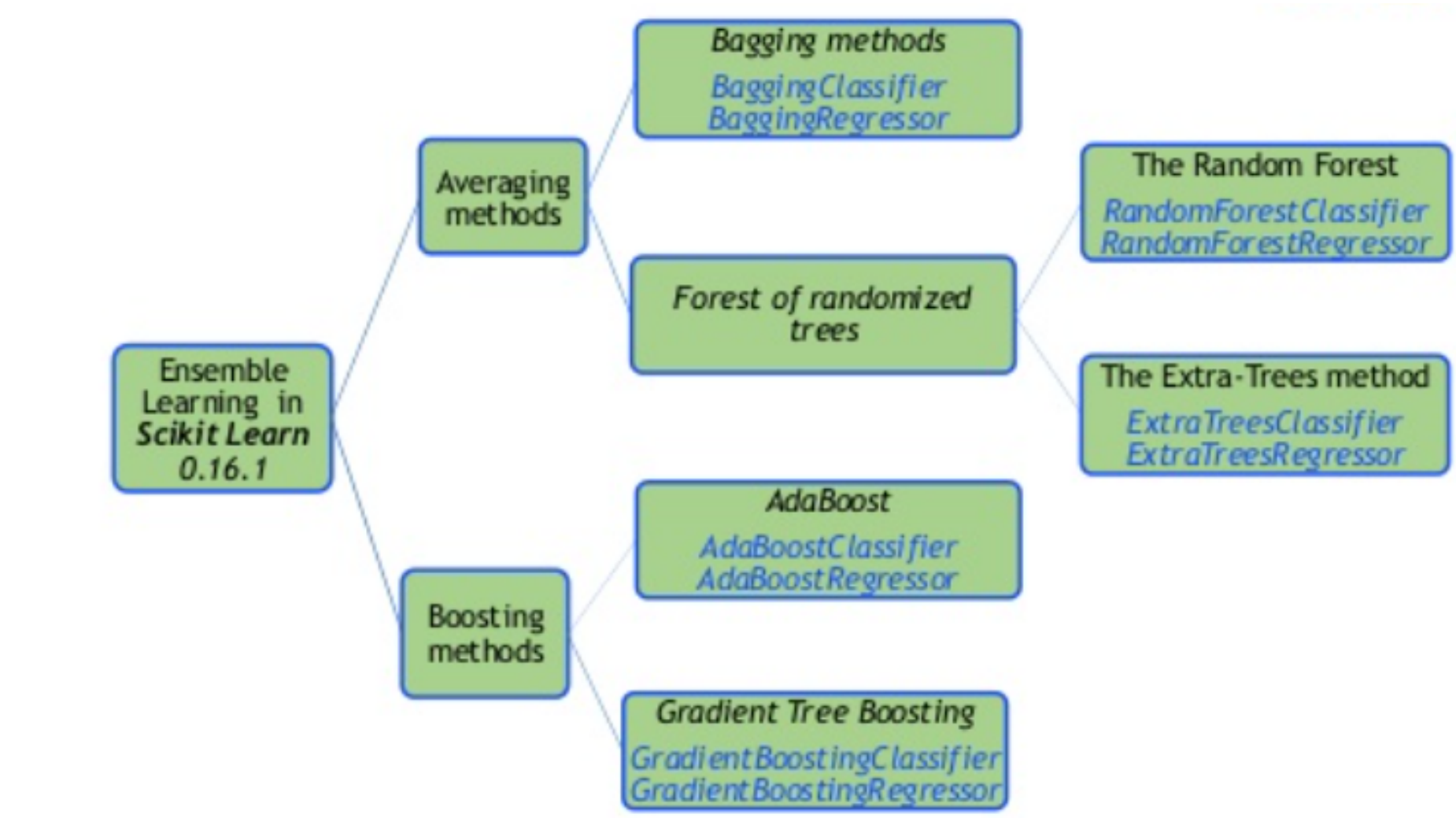
CÔNG THỨC DỰ ĐOÁN CỦA THUẬT TOÁN AdaBoost

Khi xây dựng xong mô hình, thuật toán AdaBoost thực hiện biểu quyết có trọng số theo công thức

$$\hat{y}(x) = arg\ max_k \sum_{j=1..N, \hat{y}_j(x)=k} \alpha_j \quad , N = \text{ số mô hình} \tag{4}$$

LÀM GÌ KHI AdaBoost QUÁ KHỚP

Khi mô hình quá khớp, ta có thể điều chỉnh giảm số mô hình cơ sở trong quá trình huấn luyện, hoặc tăng thêm ràng buộc cho mô hình cơ sở (Chẳng hạn, nếu mô hình cây quyết định thì có thể khống chế chiều cao, số nút,...; nếu là mô hình hồi quy thì có thể khống chế bậc, thuộc tính,...)



Hình 12: Scikit-Learn với các thuật toán học kết hợp

Gradient Boosting (Tăng cường độ dốc??)

- Gradient Boosting (GB) là một thuật toán phổ biến khác trong lớp thuật toán Boosting, được phát triển bởi GS Jerome H. Friedman năm 1999.
- Gradient Boosting tạo mô hình học kết hợp bằng cách thêm tuần tự các mô hình cơ sở sao cho mô hình sau sửa lỗi cho mô hình liền trước. Tuy nhiên, khác với AdaBoost, Gradient Boosting khớp mô hình cơ sở mới dựa trên sai số dư (residual error) của mô hình cơ sở trước đó

```python
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2)
tree_reg1.fit(X, y)

y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2)
tree_reg2.fit(X, y2)

y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2)
tree_reg3.fit(X, y3)

y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```

Hình 13: Gradient Boosted Regression Trees (GBRT)

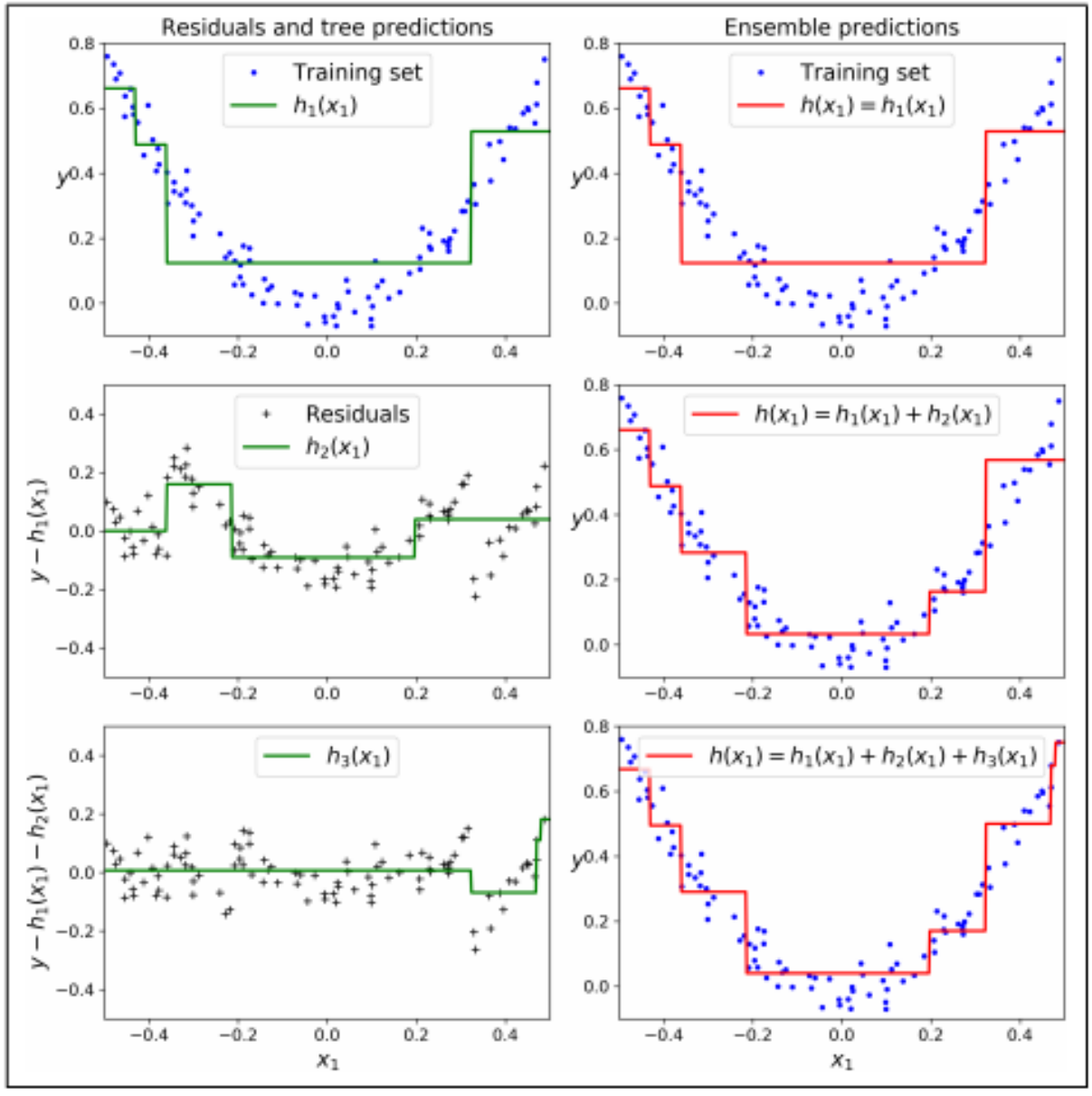


Figure 7-9. Gradient Boosting

Hình 14: Đánh giá sai sô dư và kết quả đường hồi quy theo GBRT

BÀI TOÁN: XD MÔ HÌNH MÁY HỌC ƯỚC LƯỢNG HÀM MỤC TIÊU TỪ DỮ LIỆU HUẤN LUYỆN

- Mô hình máy học được xây dựng từ dữ liệu huấn luyện $(x_i, y_i), i = 1, ..., N$
- Nhiệm vụ của mô hình là ước lượng một hàm mục tiêu $F(x)$ sao cho các điểm $F(x_i)$ khớp nhất với các điểm y_i
- Độ khớp được đo bởi hàm mất mát khả vi L mà ta sẽ ký hiệu là $L(y, F(x))$, hay $L(F(x))$, hay $L(F(x_1), ..., F(x_N))$, nghĩa là L là hàm theo các biến $F(x_i)$
- Tùy bài toán mà chọn hàm mất mát.
  - Đầu vào: Tập dữ liệu huấn luyện
  - Đầu ra: Hàm mục tiêu $F(x)$ với $L(y, F(x))$ bé nhất.

THUẬT TOÁN GRADIENT BOOSTING: Ý TƯỞNG

GB giả sử hàm mục tiêu là một tổ hợp tuyến tính của K hàm cơ sở $\varphi_i(x)$:

$$F(x) = \theta_0 + \theta_1\varphi_1(x) + ... + \theta_K\varphi_K(x) \tag{5}$$

Hay

$$F(x) = \sum_{k=0}^{K} F_k(x), F_k(x) = \theta_k\varphi_k(x) \tag{6}$$

Ý tưởng của GB:

- Khởi tạo $F(x) = F_0(x) = \theta_0$ (θ_0 *được xác định thế nào?*)
- Lần lượt thêm vào hàm $F(x)$ thành phần $[\theta_k \,.\, \varphi_k(x)]$ sao cho hàm mất mát $L(F(x))$ có giá trị nhỏ dần, $k = 1, ..., K$:

$$L(F_{k-1}(x_1) + \theta_k\varphi_k(x_1), ...) < L(F_{k-1}(x_1), ...) \tag{7}$$

- Khi đó $F(x) = F_K(x)$

THUẬT TOÁN GRADIENT BOOSTING: Ý TƯỞNG (tt)

Ước lượng θ_k và $\varphi_k(x)$ như thế nào?

- Nếu ta tìm được các giá trị thực $\theta_k r_i, i = 1, ..., N$ sao cho

$$L(F_{k-1}(x_1) + \theta_k r_1, F_{k-1}(x_2) + \theta_k r_2, ...) < L(F_{k-1}(x_1), F_{k-1}(x_2), ...) \quad (8)$$

  thì ta chỉ cần ước lượng $\varphi(x)$ bằng cách huấn luyện mô hình cơ sở M_k với tập dữ liệu $\{(x_i, r_i), i = 1, ..., N\}$, nghĩa là xem r_i là biến đáp ứng, thay vì y_i.
- Xét hàm mất mát $L(F_{k-1}(x))$, ta phải chọn điểm tiếp theo sao cho giá trị hàm L tại đó là nhỏ hơn.
  Từ (8) $\Longrightarrow$ Hướng chọn điểm chính là véc tơ $[r_i]$, và "khoảng cách" chọn điểm tiếp theo được xác định bởi số thực θ_k
- Do tính chất của gradient, ta suy ra hướng chọn điểm tốt nhất chính là ngược hướng gradient của hàm $L(F_{k-1}(x))$, nghĩa là

$$[r_i] = -\nabla L(F_{k-1}(x)) \quad (9)$$

- Cuối cùng, hệ số θ_k được ước lượng theo phương pháp tối ưu theo đường (line search method)

$$\theta_k = arg \min_{\theta} L(F_{k-1}(x) + \theta \varphi_k(x)) \quad (10)$$

Phương pháp tối ưu theo đường (line search method)

Xét hàm mục tiêu n biến $f(x)$. Thuật toán tìm điểm cực tiểu địa phương x^* của f bằng phương pháp lặp theo chiến thuật tìm theo đường như sau: Tại mỗi bước, từ điểm x_k hiện tại, phương pháp line search tính một hướng tìm kiếm (search direction) p_k sao cho hàm mục tiêu giảm dần, rồi quyết định sẽ tiến bao xa theo hướng đó. Hướng p_k có thể được xác định bằng cách chọn hướng gradient descent.

- Khởi tạo giá trị ban đầu x_k và sai số cho phép ε
- lặp quá trình sau cho đến khi $||\nabla f(x_k)|| < \varepsilon$
  - Xác định hướng giảm p_k
  - Chọn α_k (được gọi là độ dài bước) làm cực tiểu hàm $h(\alpha) = f(x_k + \alpha * p_k), \alpha \in R+$
  - $x_k = x_k + \alpha_k p_k$

THUẬT TOÁN GRADIENT BOOSTING

Đầu vào: Dữ liệu huấn luyện X; Hàm mất mát L; Mô hình học máy cơ sở M_θ; Số vòng lặp K; Tốc độ học η.
Đầu ra: Hàm mục tiêu $F(x)$

THUẬT TOÁN GRADIENT BOOSTING

1 Khởi tạo $\hat{F}^{(0)}(x) = \hat{F}_0(x) = \hat{\theta}_0 = arg\ \min_\theta \sum_{i=1}^n L(y_i, \theta)$

2 **for** $k = 1, ..., K$ **do**

a Tính gradient ngược hướng

$$\hat{g}_k(x_i) = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=\hat{F}^{(k-1)}(x)}, i = 1, ..., n \tag{11}$$

b Ước lượng hàm cơ sở thứ k

$$\hat{\varphi}_k = arg \min_{\varphi,\beta} \sum_{i=1}^n [(-\hat{g}_k(x_i)) - \beta\varphi(x_i)]^2 \tag{12}$$

c Tính kích thước bước nhảy

$$\hat{\rho}_k = arg \min_\rho \sum_{i=1}^n L(y_i, \hat{F}^{(k-1)}(x_i) + \rho\hat{\varphi}_k(x_i)) \tag{13}$$

d Cập nhật hàm mục tiêu

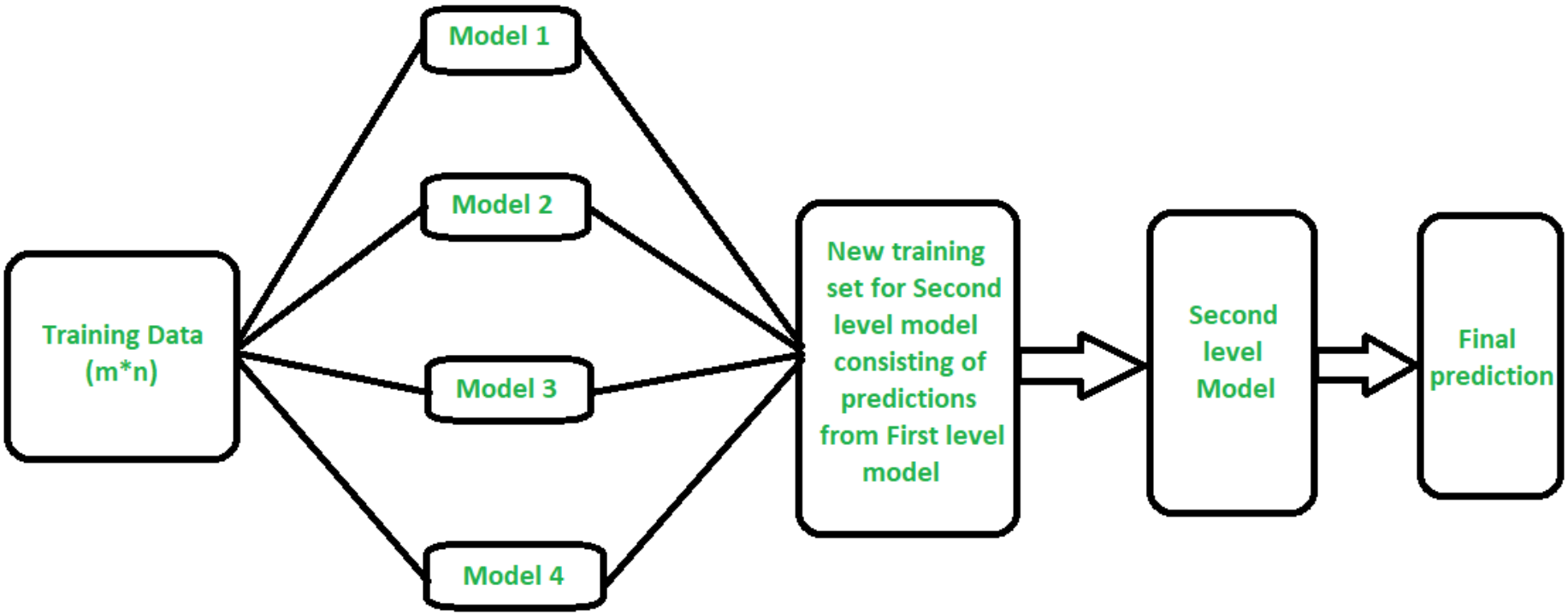
$$\hat{F}^k(x) = \hat{F}^{(k-1)}(x) + \hat{F}_k(x) \text{ với } \hat{F}_k(x) = \eta\hat{\rho}_k\hat{\varphi}_k(x) \tag{14}$$

3 $\hat{F}(x) = \hat{F}_K(x) = \sum_{k=0}^K \hat{F}_k(x)$

Nội dung trình bày

6. Stacking (Xếp chồng)

- Stacking = stacked generalization - D. Wolpert (1992)
- Ý tưởng của stacking: Huấn luyện một mô hình để thực hiện dự đoán trong mô hình học kết hợp, thay vì dùng các hàm đơn giản, chẳng hạn như trong biểu quyết cứng.



Hình 15

THUẬT TOÁN STACKING

Đầu vào: Tập huấn luyện $D = \{x_i, y_i\}, i = 1, ..., N$, số mô hình cơ sở K
Đầu ra: Mô hình học tích hợp H
Thuật toán:

- B1: Huấn luyện mô hình cơ sở h_k trên tập D, $k = 1, ..., K$
- B2: Xây dựng tập huấn luyện mới trên kết quả của các mô hình cơ sở:

$$D_h = \{x'_i, y_i\}, x'_i = \{h_1(x_i), ..., h_K(x_i)\}$$

- B3: Huấn luyện mô hình H dựa trên D_h

Mô hình phối trộn (Blending)

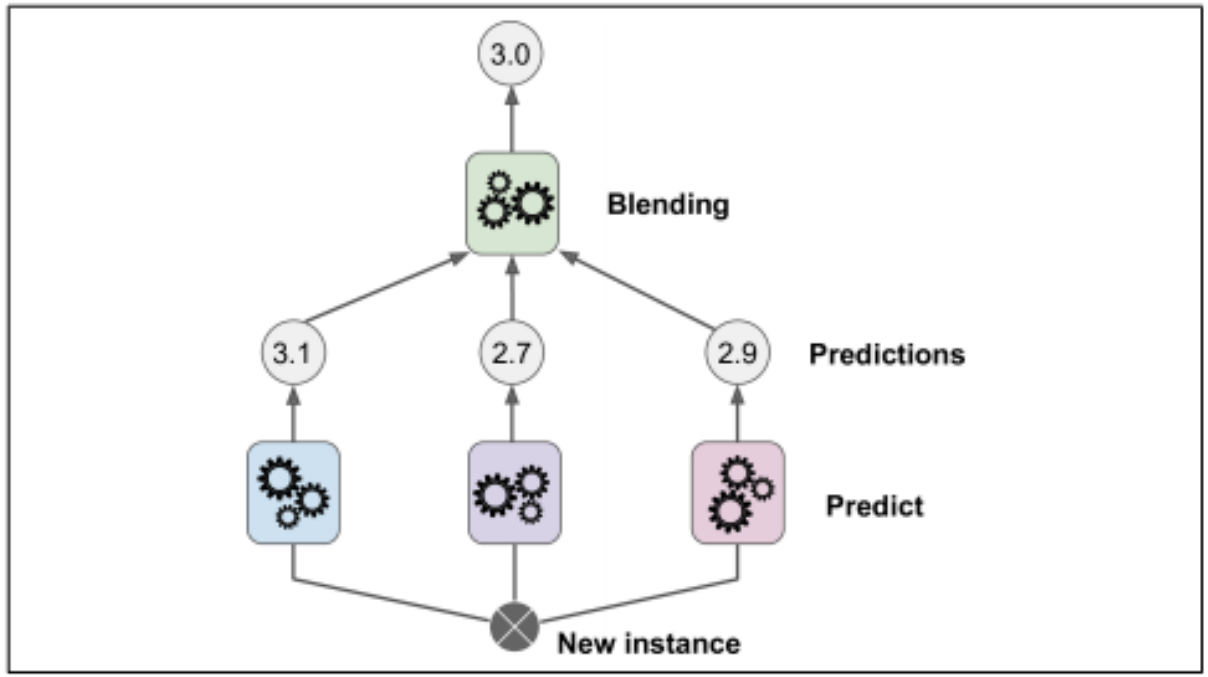
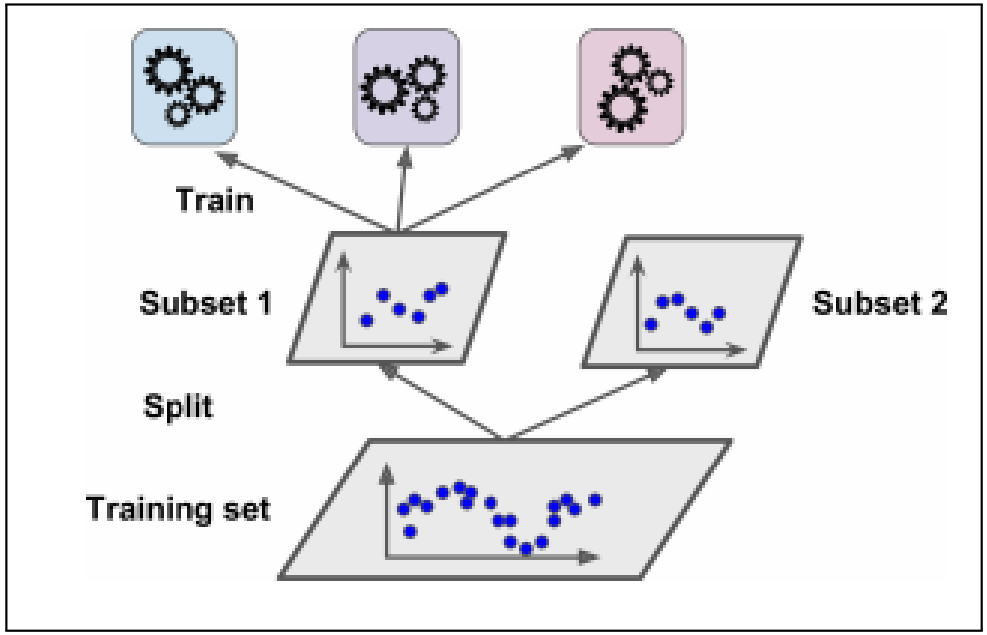


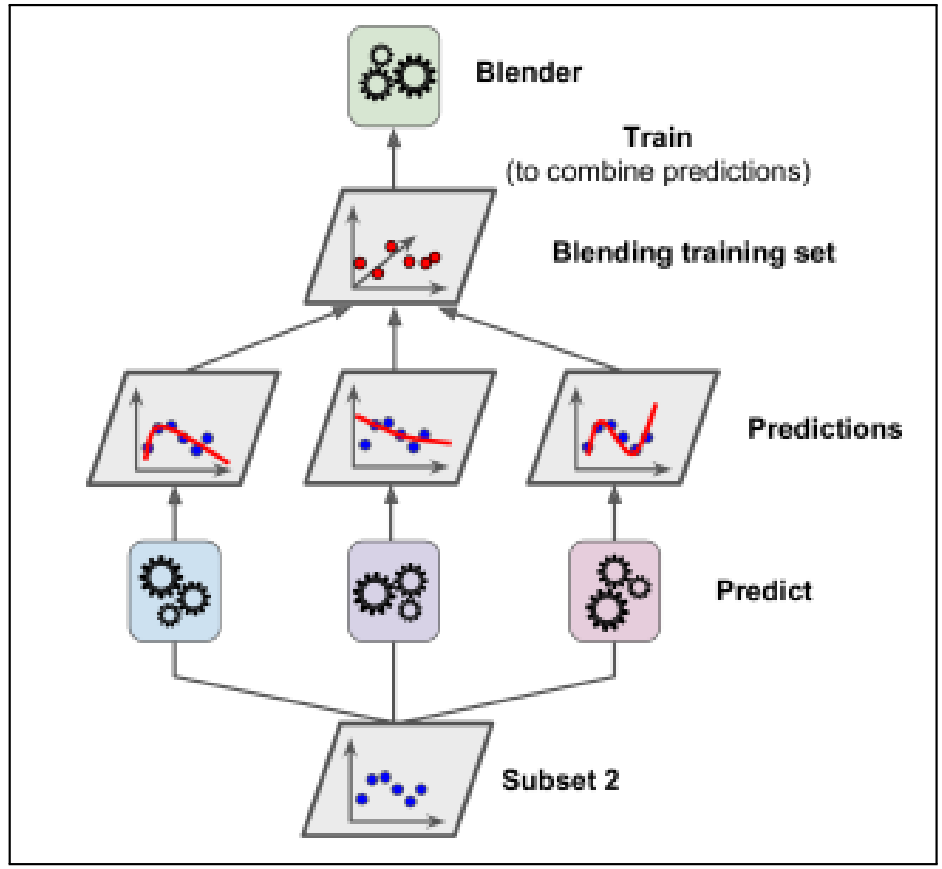
Figure 7-12. Aggregating predictions using a blending predictor

Hình 16

- Mô hình phối trộn là một dạng mô hình xếp chồng
- Thuật toán cho mô hình 2 lớp:
  - Tập dữ liệu được chia 2: $S1$ và $S2$
  - Huấn luyện K mô hình cơ sở trên $S1$
  - Xây dựng tập huấn luyện S_2' cho lớp 2 là các dự đoán của tập $S2$ qua K mô hình cơ sở.
  - Huấn luyện lớp 2 trên tập S_2'
- Xây dựng mô hình phối trộn nhiều lớp hơn được thực hiện tương tự mô hình 2 lớp

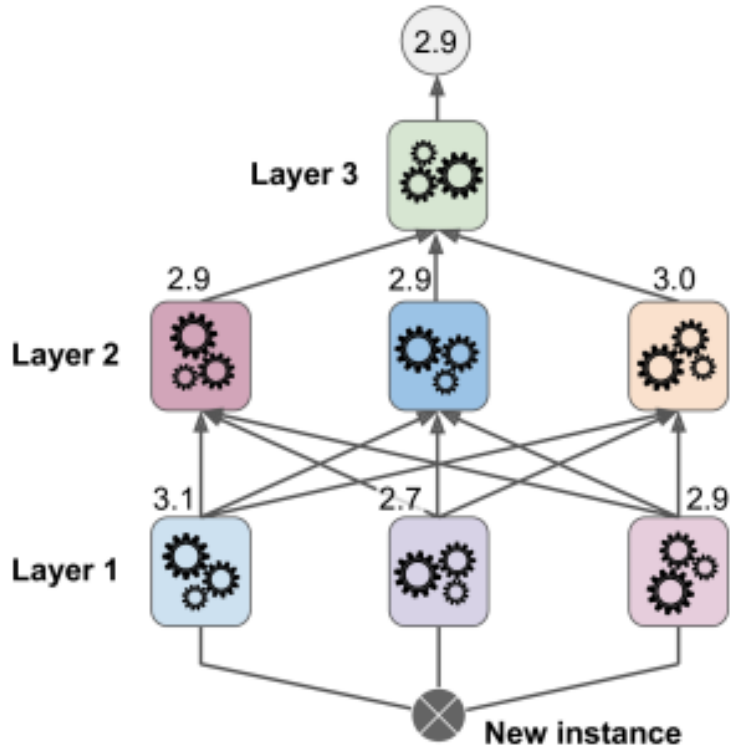


Hình 17: Huấn luyện lớp thứ 1



Hình 18: Huấn luyện lớp thứ 2 (blender)

Mô hình phối trộn 3 LỚP



Hình 19: Mô hình dự báo phối trộn 3 lớp

Bài tập

1. Biểu quyết cứng và biểu quyết mềm khác nhau thế nào?
2. Giả sử bạn đã huấn luyện được 3 mô hình trên cùng tập dữ liệu với độ chính xác 90%. Có cách nào để kết hợp các mô hình này để tăng độ chính xác lên không? Giải thích.
3. Nêu sự khác biệt cơ bản giữa các phương pháp
   a. bagging và pasting
   b. random patches và random subspaces
   c. bagging và boosting
4. Tập OOB là gì? Vai trò của tập OOB trong đánh giá mô hình?
5. Trình bày thuật toán rừng ngẫu nhiên
6. Khi Adaboost quá khớp thì cần hiệu chỉnh các siêu tham số nào?
7. Trình bày ý tưởng của thuật toán Gradient Boosting
8. Trình bày mô hình kỹ thuật xếp chồng (stacking)
9. Trình bày thuật toán xây dựng mô hình phối trộn (blending) hai lớp.

Bài tập

10 Hãy sử dụng tập dữ liệu MNIST và chia thành 3 tập: Huấn luyện (training), kiểm tra (validation) và đánh giá (test)

a Huấn luyện 3 mô hình phân lớp: Rừng ngẫu nhiên (random forest), Rừng ngẫu nhiên hoàn toàn (extra-trees) và SVM

b Hãy xây dựng mô hình kết hợp từ 3 mô hình ở câu *a*.

c So sánh độ chính xác các mô hình ở câu *a* và câu *b*

d Hãy xây dựng mô hình phối trộn (blending): Chạy từng mô hình ở câu *a* với tập kiểm tra để tạo tập huấn luyện mới S_2'. Chọn mô hình phân lớp và huấn luyện mô hình này với tập S_2'. Sử dụng tập đánh giá để đánh giá mô hình phối trộn

e So sánh mô hình ở câu *d* với cách biểu quyết cứng khi chạy 3 mô hình riêng lẻ theo phương pháp bagging ở câu *a*

Tài liệu tham khảo

- Sách tham khảo "Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, 2nd Edition" của tác giả "Aurélien Géron".
- https://ongxuanhong.wordpress.com/2017/12/21/xgboost-thuat-toan-gianh-chien-thang-tai-nhieu-cuoc-thi-kaggle/
- https://www.brandsvietnam.com/17493-Thu-doan-cua-Netflix
- https://www.researchgate.net/publication/332196117_SO_SANH_MO_HINI
- Marina Skurichina and Robert P. W. Duin, *Bagging, Boosting and the Random Subspace Method for Linear Classifiers*, Pattern Analysis & Applications (2002)5:121–135 http://rduin.nl/papers/paa_02_bagging.pdf
- Tree Boosting With XGBoost – Why Does XGBoost Win "Every" Machine Learning Competition? https://syncedreview.com/2017/10/22/tree-boosting-with-xgboost-why-does-xgboost-win-every-machine-learning-competition/
- Juan ChengGen LiGen LiXianhua Chen, Research on Travel Time Prediction Model of Freeway based on Gradient Boosting Decision Tree, IEEE Access PP(99):1-1, 2018
- https://fr.slideshare.net/katatunix/gradient-boosting