Ensemble Learning

Lecture 9

Today: Learning Objectives

1. Ensemble Learning
2. Bagging
3. Random Forest
4. Boosting
5. Stacking

1. Ensemble Learning

NETFLIX

Netflix Prize

Home | Rules | Leaderboard | Register | Update | Submit | Download

“Our winning model is an ensemble of 107 models. Our experience is that most efforts should be concentrated in deriving substantially different approaches, rather than refining a simple technique.”

Ensemble Learning

Wisdom of the Crowd: consider a collective answer from 1000 random people vs. 1 expert. **Which one would you prefer?**

Aggregate the prediction of a group of predictors (classifiers or regressors) → can be better prediction than the best individual predictor → Ensemble Learning

Popular ensemble learning methods:

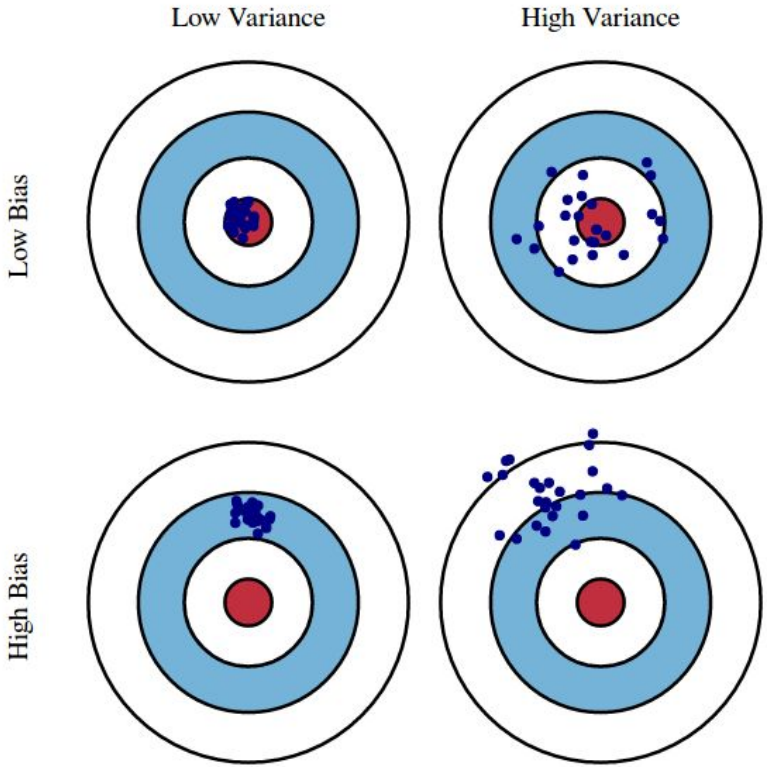
- Bagging
- Random Forest
- Boosting
- Stacking

Review: Bias and Variance

Model's Error = **Bias** +
Variance +
Irreducible Error

Bias: the error that is introduced by approximating a real-life problem, which may be extremely complicated, by a much simpler model.

Variance: the amount by which model's prediction would change if it's trained using a different training data set.



Averaging Reduces Variance

If a variable **x** is measured N models $(X_1, X_2, \ldots, X_n)$ and the value is estimated as $(X_1 + X_2 + \ldots + X_n)/N$

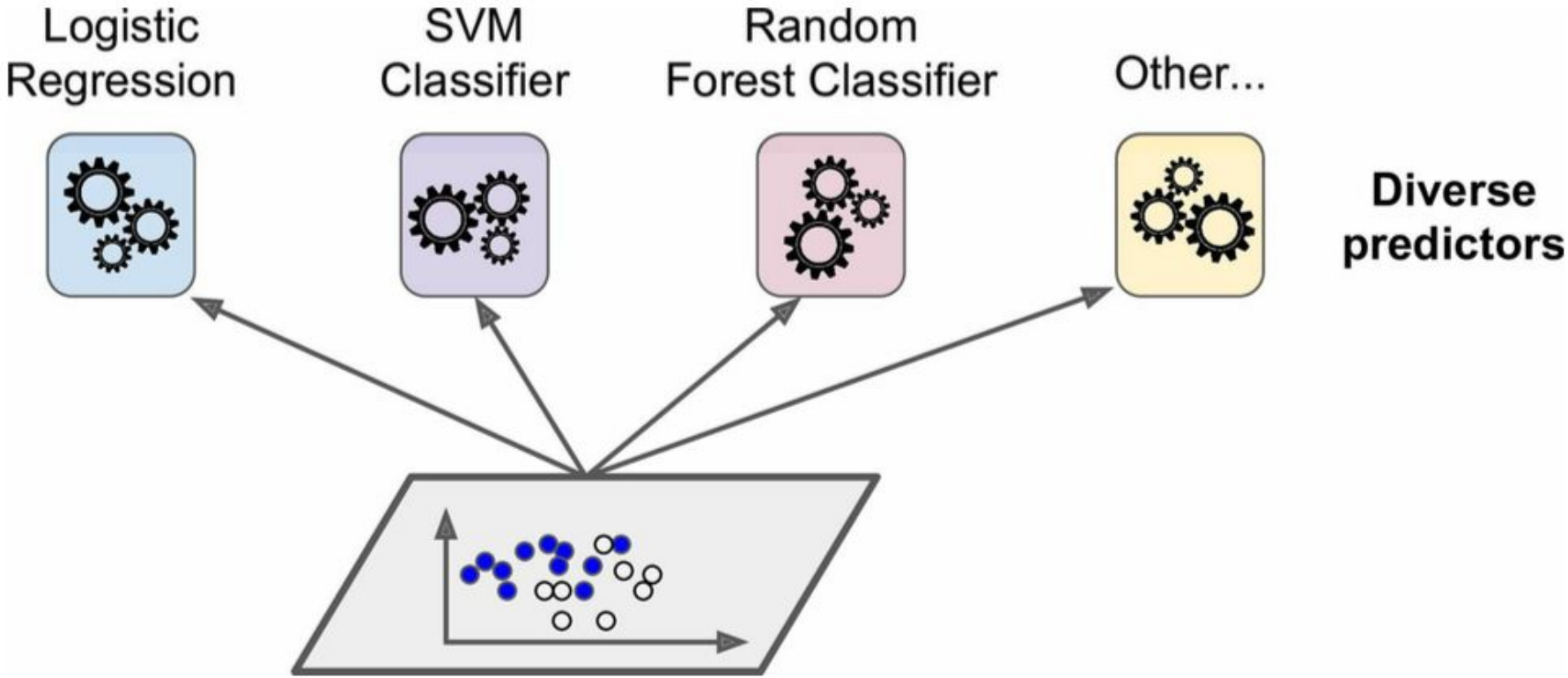
- Mean of the models is still the same
- However, variance is smaller:

$$\mathrm{Var}\left(\frac{1}{N}\sum_i X_i\right) = \frac{1}{N}\mathrm{Var}(X_i)$$

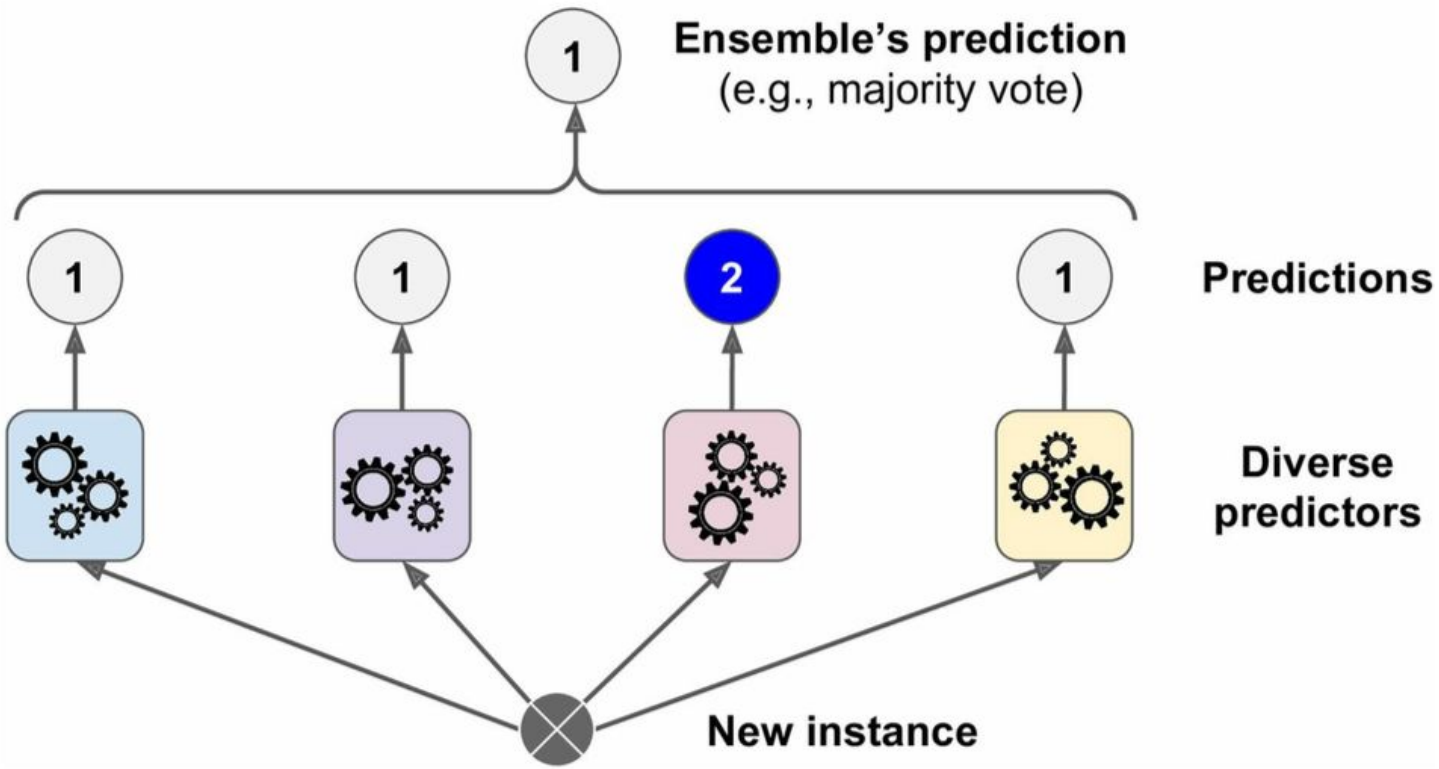
So averaging reduces the model's variance. One problem:

Where do multiple models come from (we only have one training set)?

Voting Classifiers



Hard voting classifier



Code Demo

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

log_clf = LogisticRegression(random_state=42)
rnd_clf = RandomForestClassifier(random_state=42)
svm_clf = SVC(random_state=42)

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

Will this voting classifier outperforms the individual classifiers?

Soft Voting Classifiers

Estimate class probabilities → might improve accuracy (slightly?)

```
log_clf = LogisticRegression(random_state=42)
rnd_clf = RandomForestClassifier(random_state=42)
svm_clf = SVC(probability=True, random_state=42)

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

2. Bagging

Bootstrap Estimation

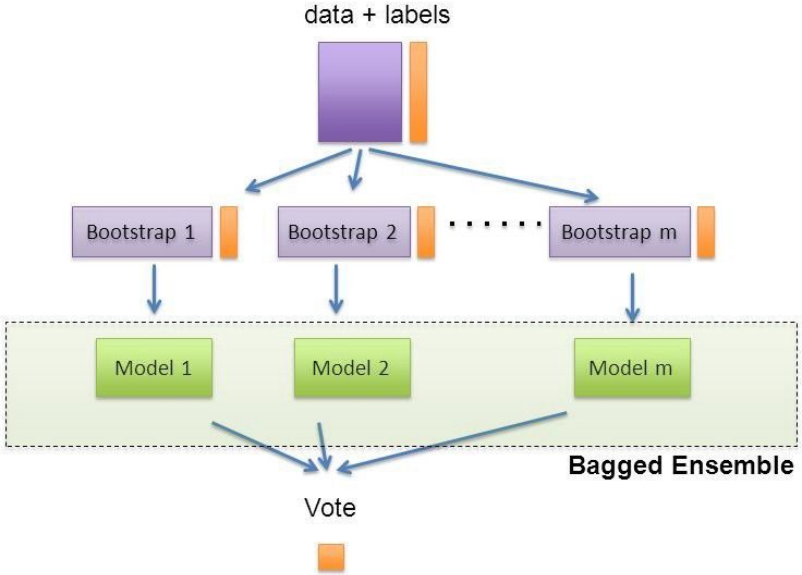
One way to get diverse set of classifiers is to use **different training algorithms**. Another way is to use the same algorithm, but train on **different random subsets.**

The idea:

- Repeatedly draw $\mathbf{m}$ bootstrap samples from the original training set X
- For each set of samples, train a model and then predict the labels
- Combine results of all bootstrap sets using majority voting

→ Bagging: **B**ootstrap **Agg**regation

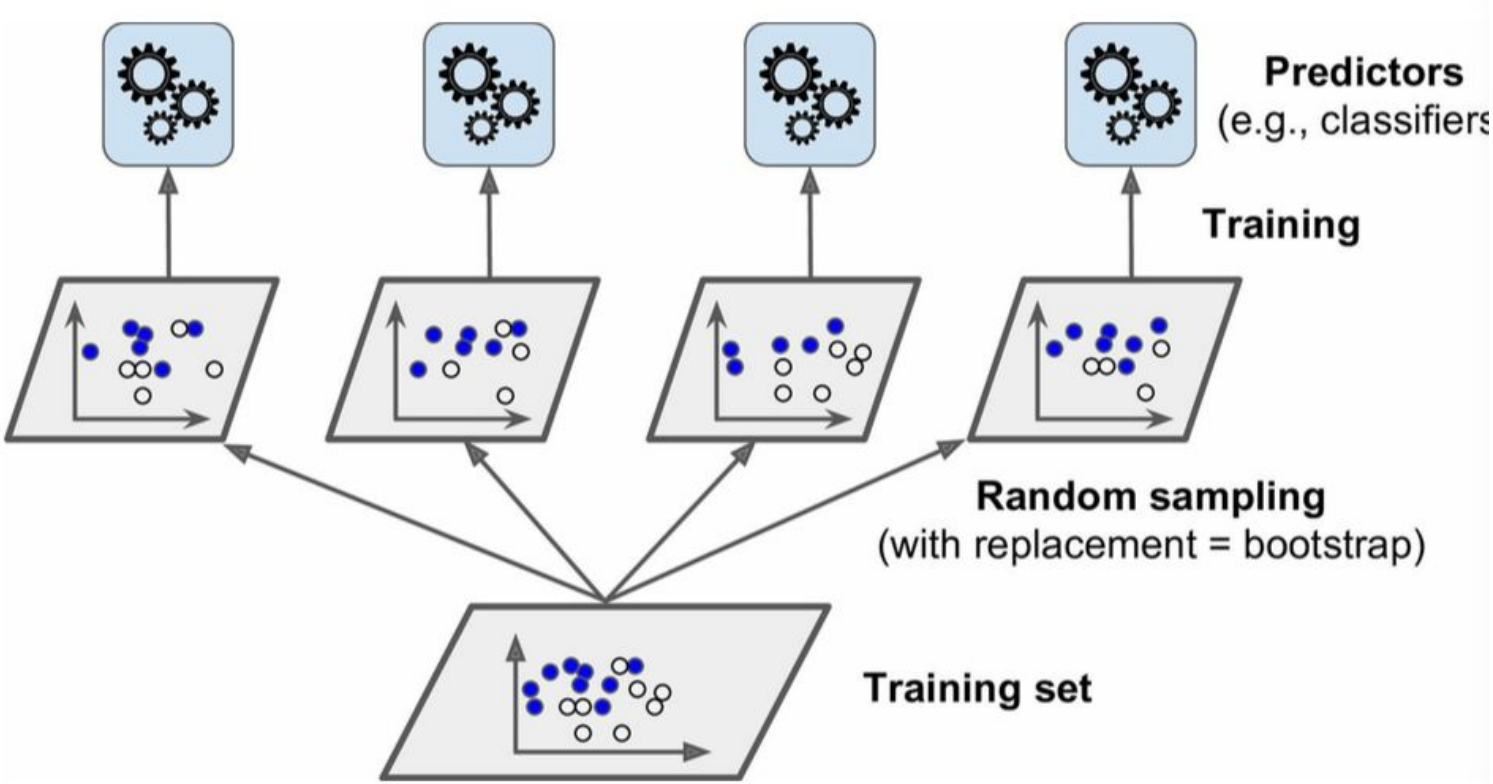
Bagging vs. Pasting



- Sampling is performed **with** replacement → **bagging**
- Sampling is performed **without** replacement → **pasting**

Which one is generally better?

Training in Parallel with Bagging (Scale up)



Code

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1, random_state=42)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

Decision Boundary Comparison

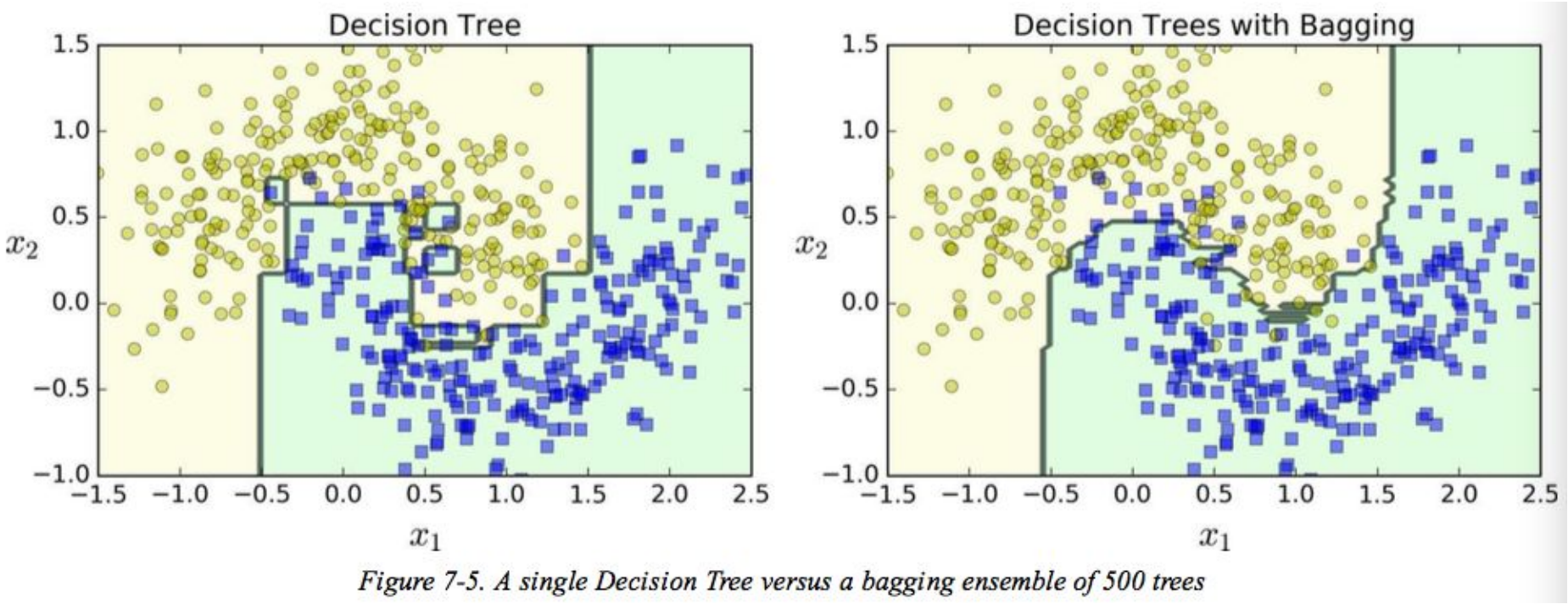


Figure 7-5. A single Decision Tree versus a bagging ensemble of 500 trees

Out-of-bag Evaluation

Because it's random sampling, some examples may not be sampled at all → **out-of-bag** (oob) examples

→ They can be used for testing instead of a separate test set.

```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=500,
    bootstrap=True, n_jobs=-1, oob_score=True, random_state=40)
bag_clf.fit(X_train, y_train)
bag_clf.oob_score_
```

```
0.9013333333333333
```

Random Patches and Random Subspaces

We can sampling the features as well (`max_features`)

Each model will be trained on a random subset of the input features

Useful when dealing with high-D inputs (ie. images)

- **Random Patches:** sampling both samples and features
- **Random Subspaces:** keeping all training samples while sampling the features

Sampling features results in more predictor diversity, and thus trades more bias for lower variance.

3.

Random Forest

An ensemble of Decision Trees, generally trained with **bagging**

Introduce **extra randomness** when growing trees: instead of searching for the best feature with splitting node, it searches for the best feature among a random set of features (subspaces) → a greater tree **diversity**

It also trades a **higher bias** for a **lower variance**

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1, random_state=42)
rnd_clf.fit(X_train, y_train)
```

Extra Trees

When growing a tree in a Random Forest, only a **random set of features** is considered for splitting at each node.

Make tree more random by using a **random threshold** for each feature rather than search for the best possible threshold like regular decision tree

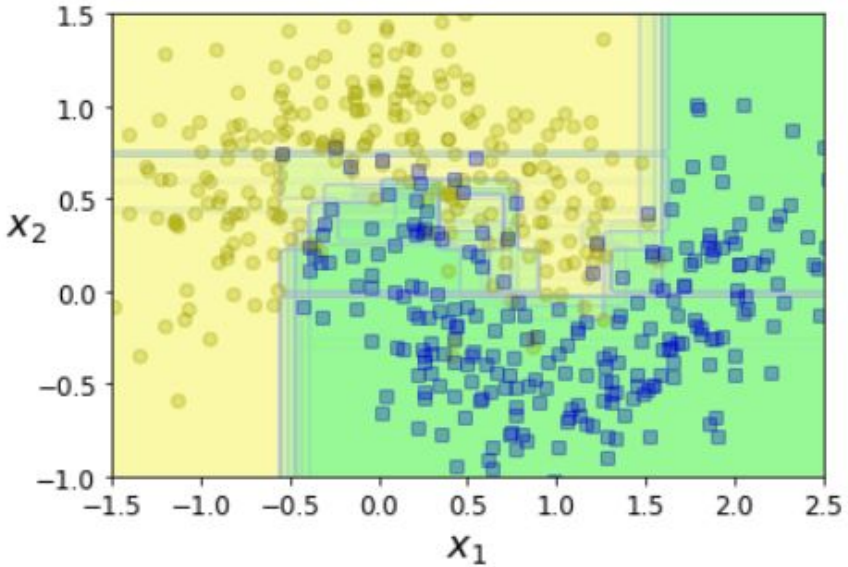
Extremely Randomized Trees (Extra-Trees): faster to train, and trades even higher bias or a lower variance.

Code

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, n_jobs=-1, random_state=42)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```



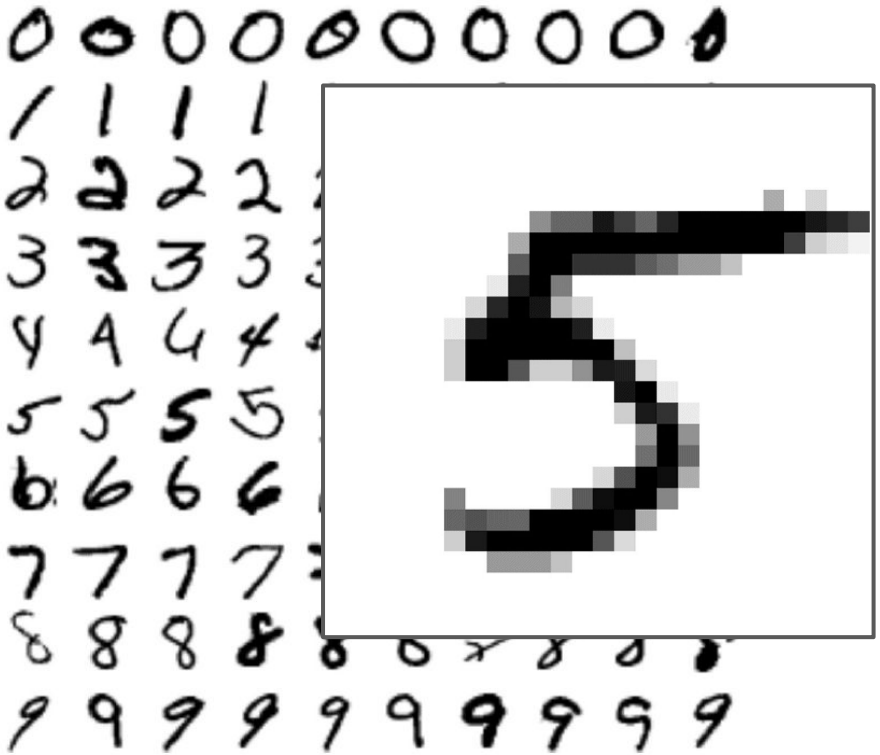
Feature Importance

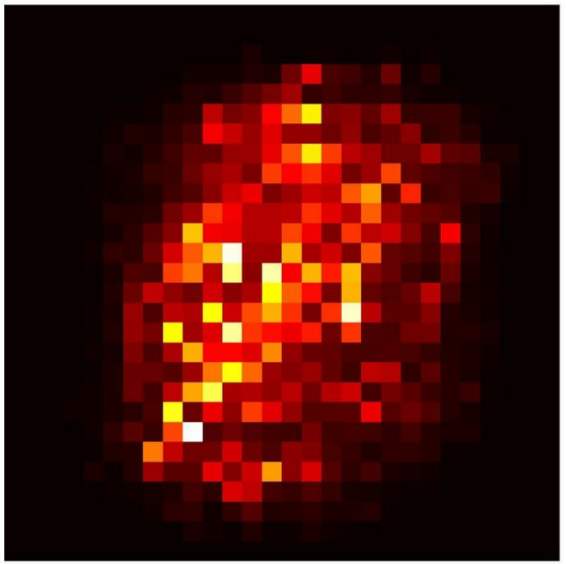
In Decision Tree, the important features are likely to appear closer to the root → it is possible to estimate the importance of the features by computing **the average depth** at which they appear across all trees in the forest (*feature_importances*)

```
from sklearn.datasets import load_iris
iris = load_iris()
rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1, random_state=42)
rnd_clf.fit(iris["data"], iris["target"])
for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
    print(name, score)
```

```
sepal length (cm) 0.112492250999
sepal width (cm) 0.0231192882825
petal length (cm) 0.441030464364
petal width (cm) 0.423357996355
```

Feature Importance in MNIST Data



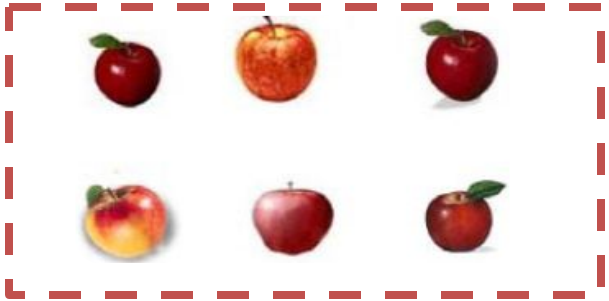


4. Boosting

Recognizing apples

- Collect a set of real apples and plastic apples
- Observe some rules to tell them apart based on their characteristics

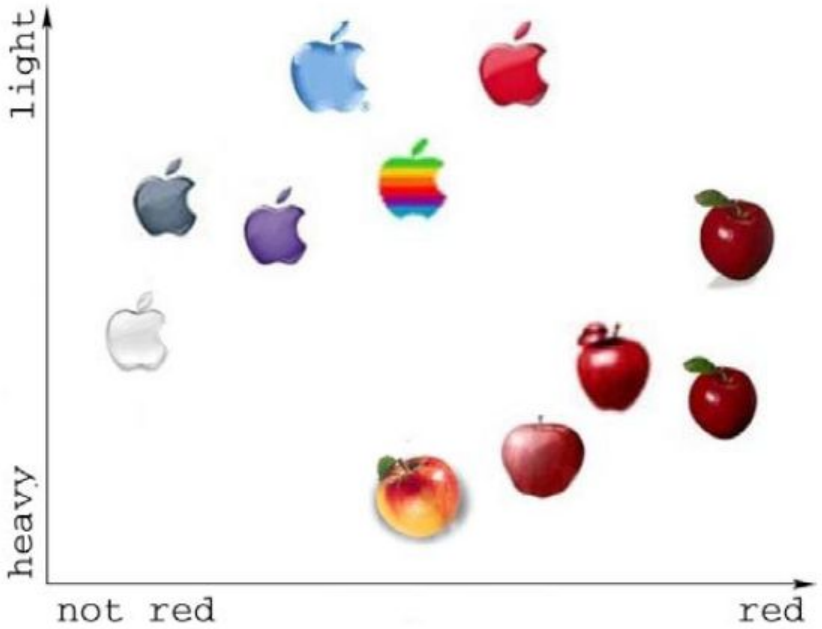
Real Apples

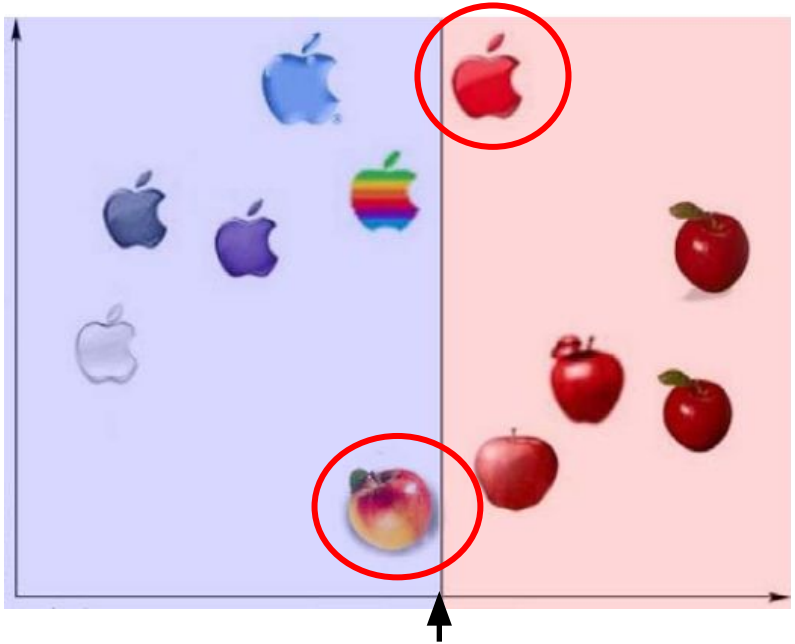


Plastic Apples

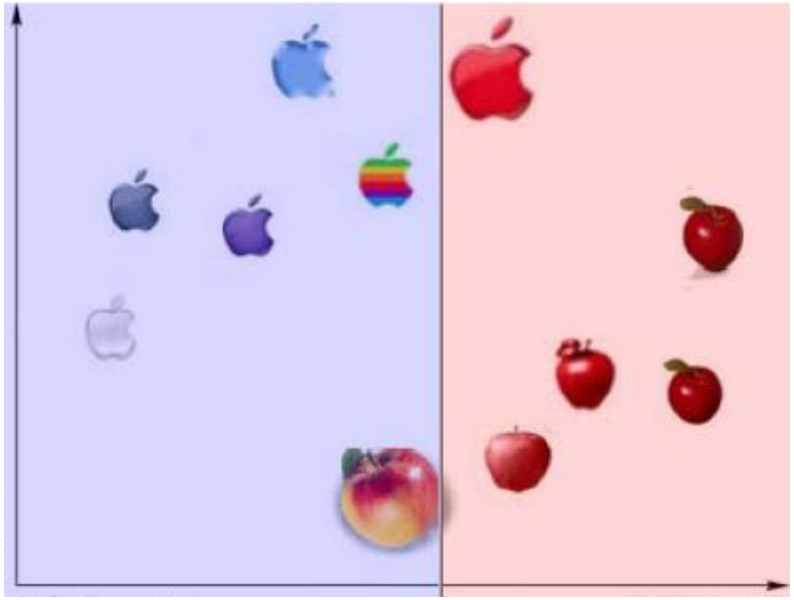
Boosting Strategies

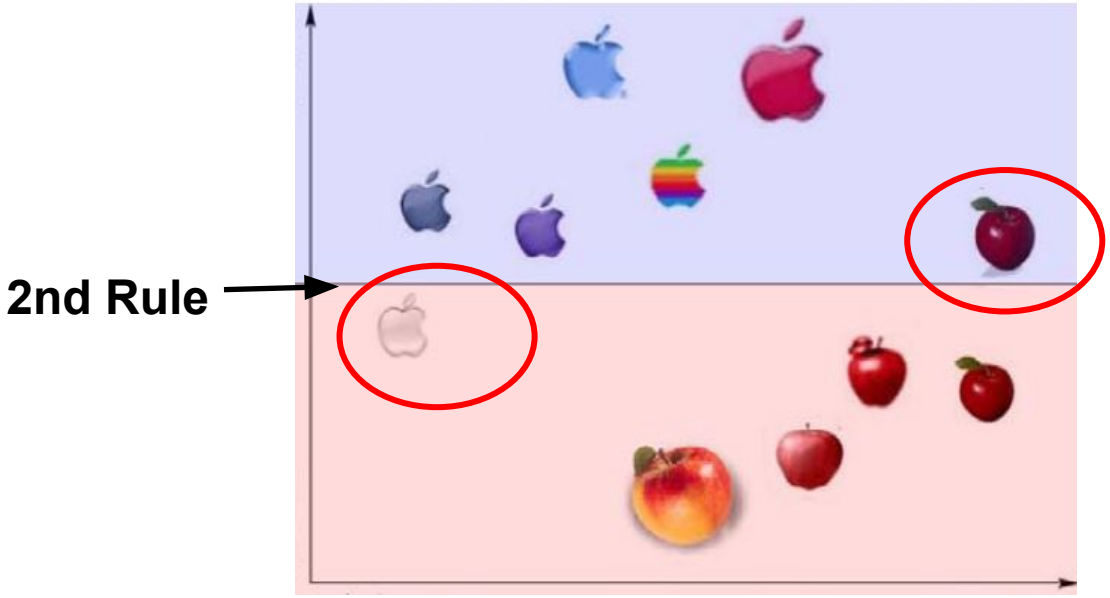
1. Have many rules (base classifiers) to **vote** on the decision
2. Sequentially train rule that **corrects** mistakes of previous rule → focus on **hard** examples
3. Give higher **weight** to better rules

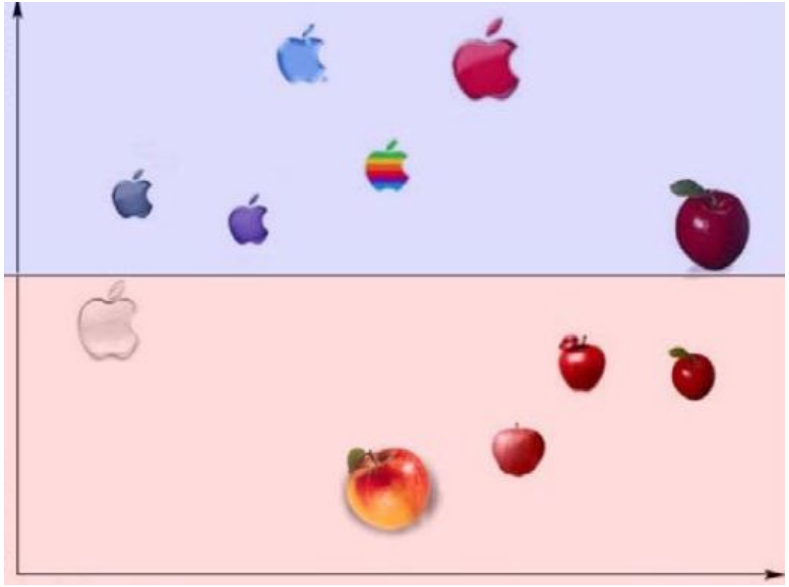

light
heavy
not red
red



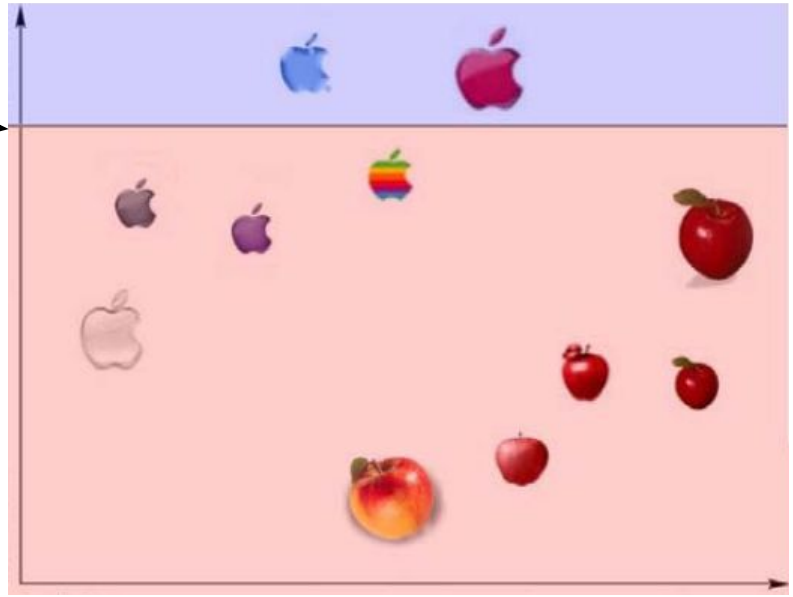
1st Simple Rule

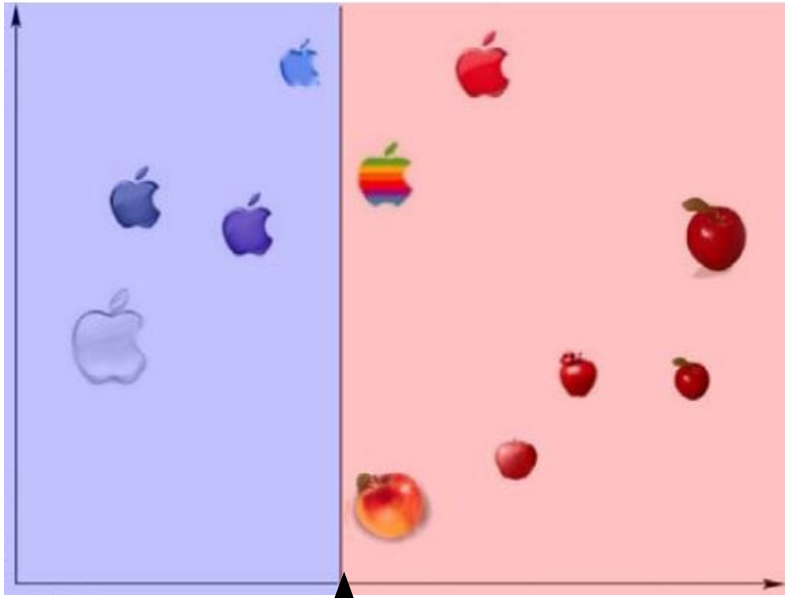


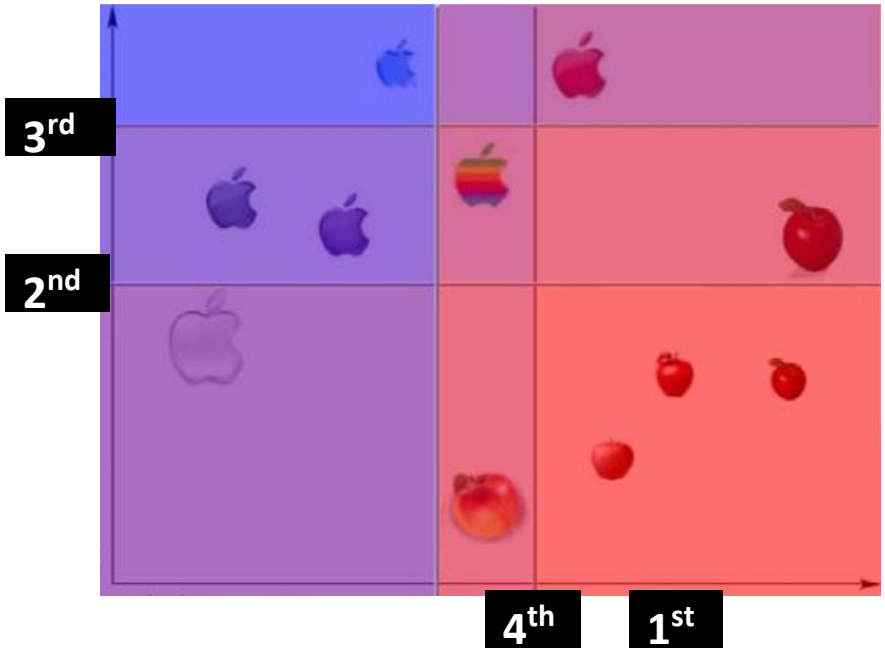

2nd Rule

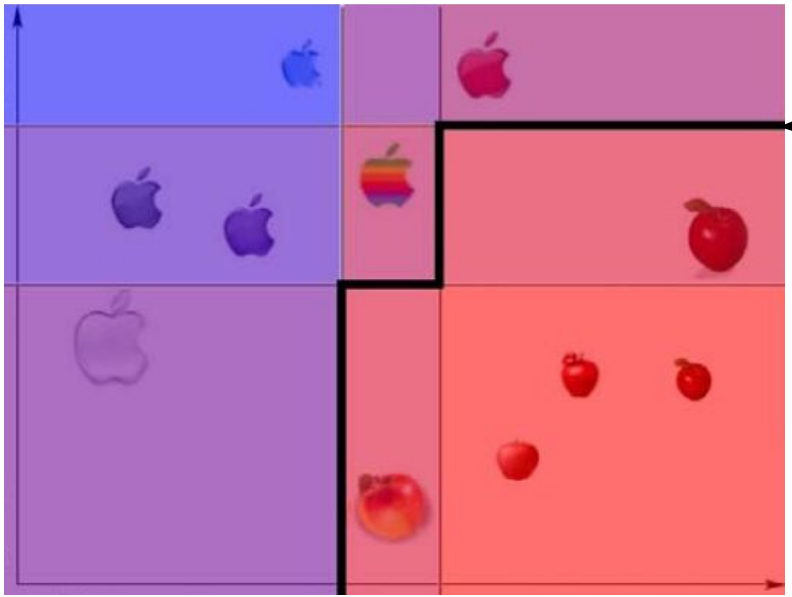


3rd Rule →

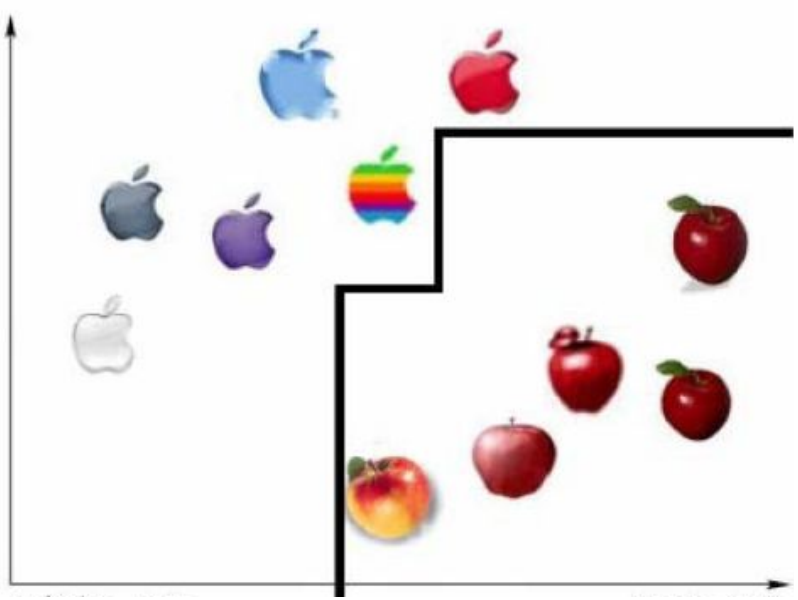



4th Rule


3rd
2nd
4th
1st



A More Complex Rule



Adaboost Algorithm (Proposed by Robert Schapire)

Training Data: $\mathbf{D} = \{(\mathbf{x}_i, y_i) | \mathbf{x}_i \in \mathcal{R}^n, y_i \in \{-1, 1\}, 1 \leq i \leq m\}$

Set uniform example weight $w_i, 1 \leq i \leq m$

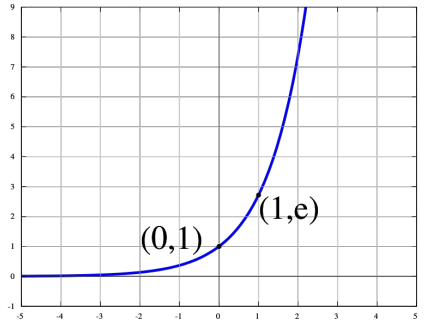
For `t = 1` to `T` iterations:

Select a base classifier: $h_t(\mathbf{x}_i) = \arg\min(\epsilon_t)$

$$\epsilon_t = \sum_{i=1}^{m} w_i [y_i \neq h_t(\mathbf{x}_i)]$$

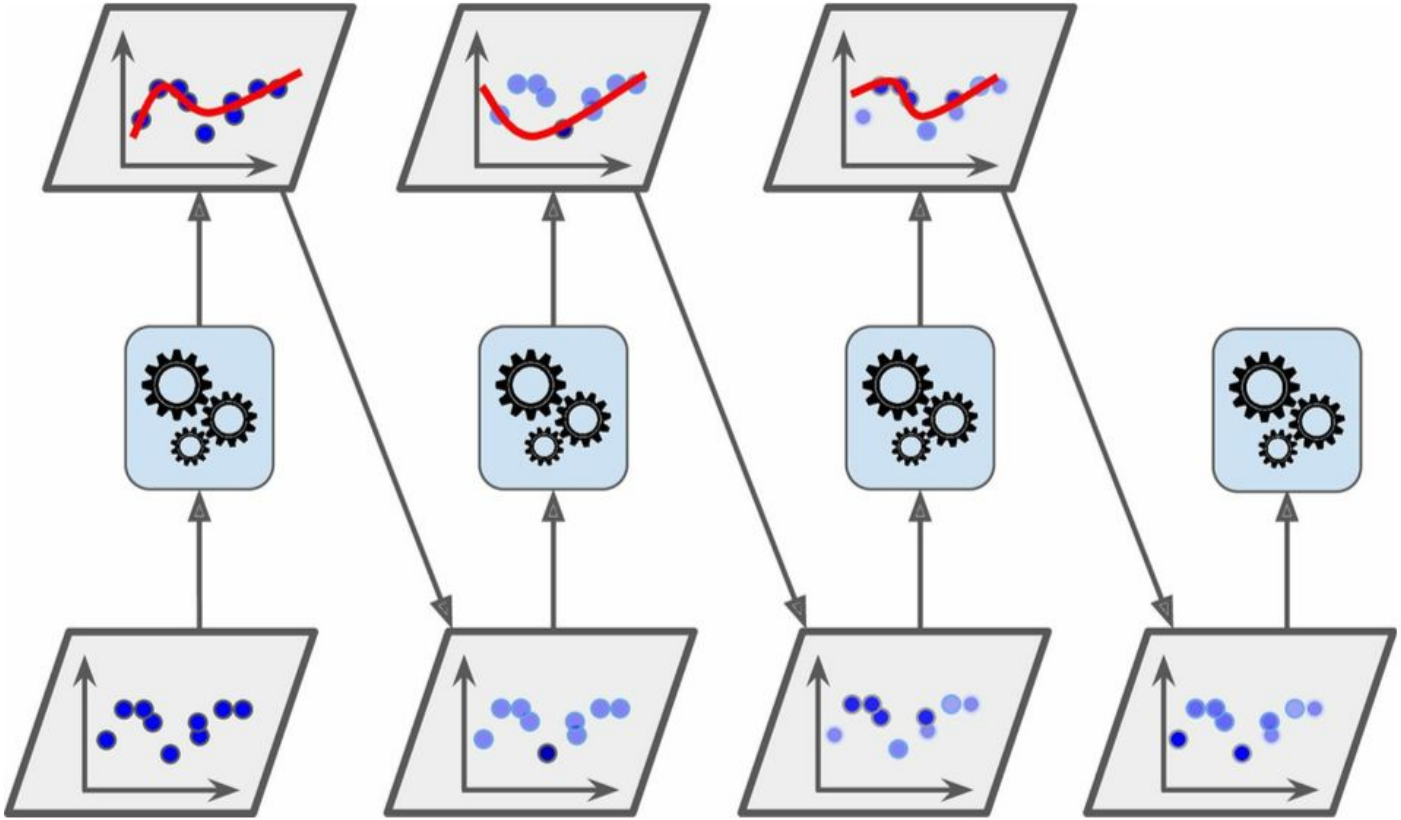
Set classifier weight: $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$

Update example weight: $w_i = w_i e^{-\alpha_t y_i h_t(\mathbf{x}_i)}$



Final Classifier: $\hat{f} = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}_i)\right)$

Boosting (visual perspective)

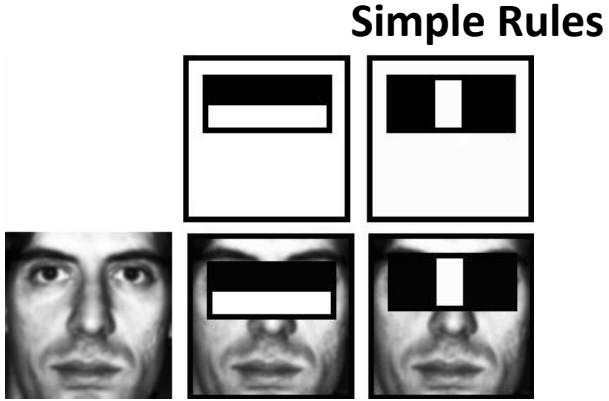


Why use boosting?

- Fast and simple to code
- No hyper-parameter to tune (except for $\mathbf{T}$)
- No prior knowledge needed about base classifier
- Flexible to combine with any learning algorithm

An application in Face Detection

- Viola-Jones Face Detector
- Uses Adaboost algorithm to combine lots of simple rules for face detection



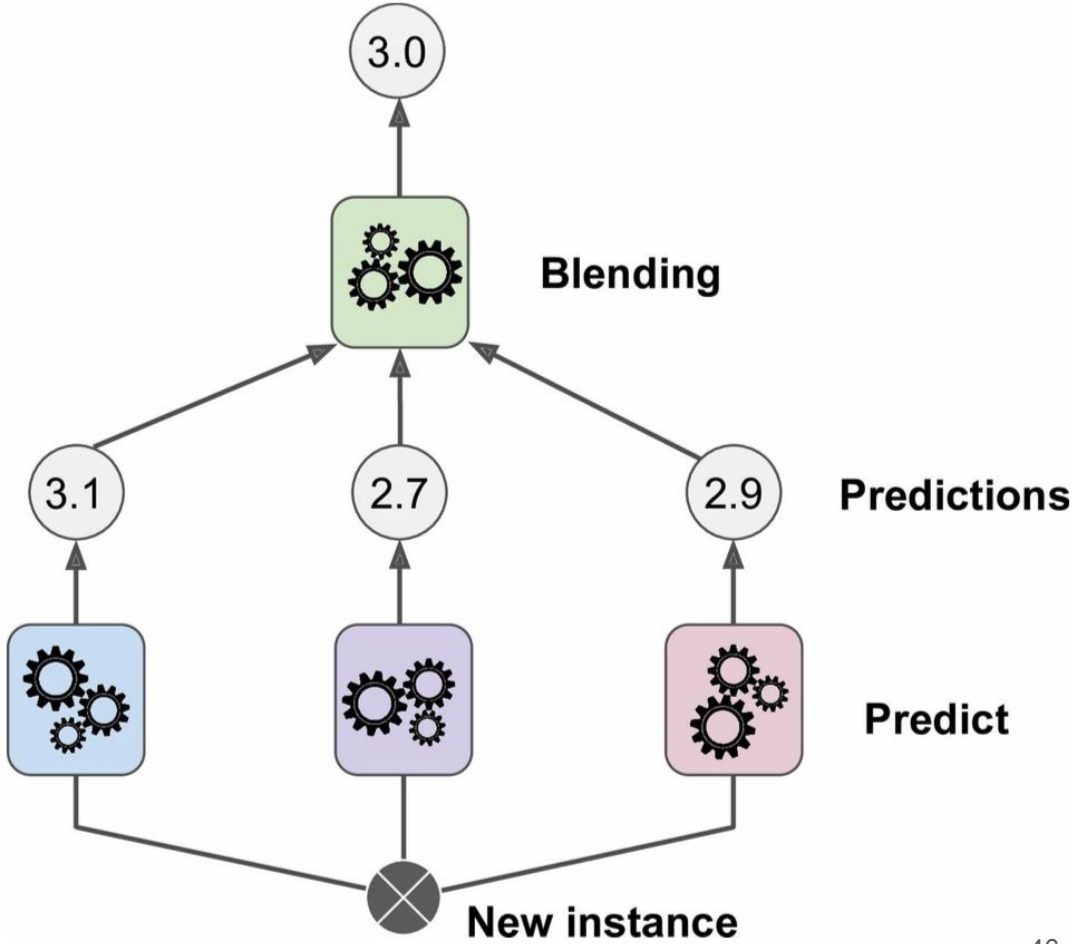
Boosting vs. Bagging

- Similar to bagging, boosting combines a weighted sum of many classifiers, thus **it reduces variance**.
- One key difference: unlike bagging, boosting fit the tree to the entire training set, and adaptively weight the examples. Boosting tries to do better at each iteration, thus **it reduces bias**.
- In general: **Boosting > Bagging > Decision Tree**

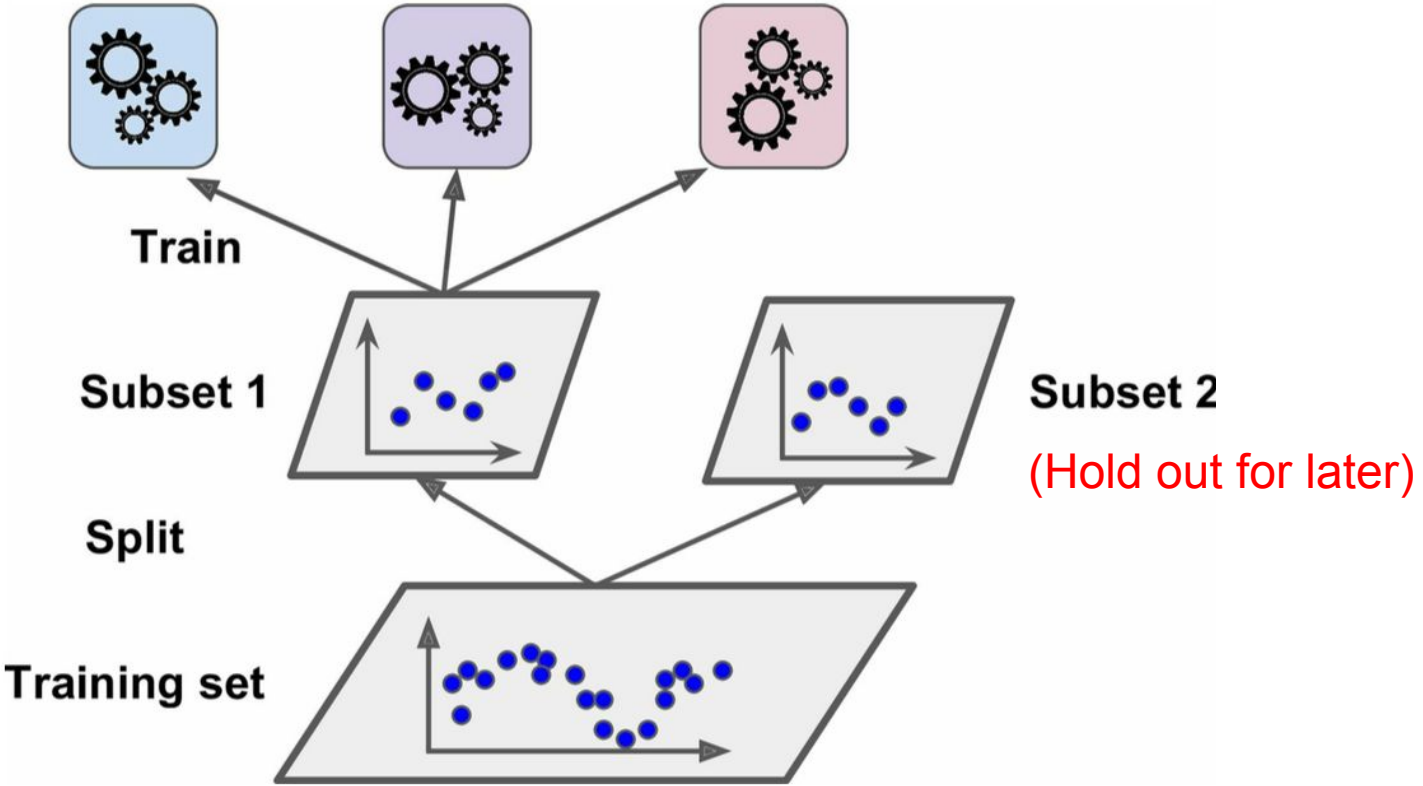
5. Stacking

Stacking

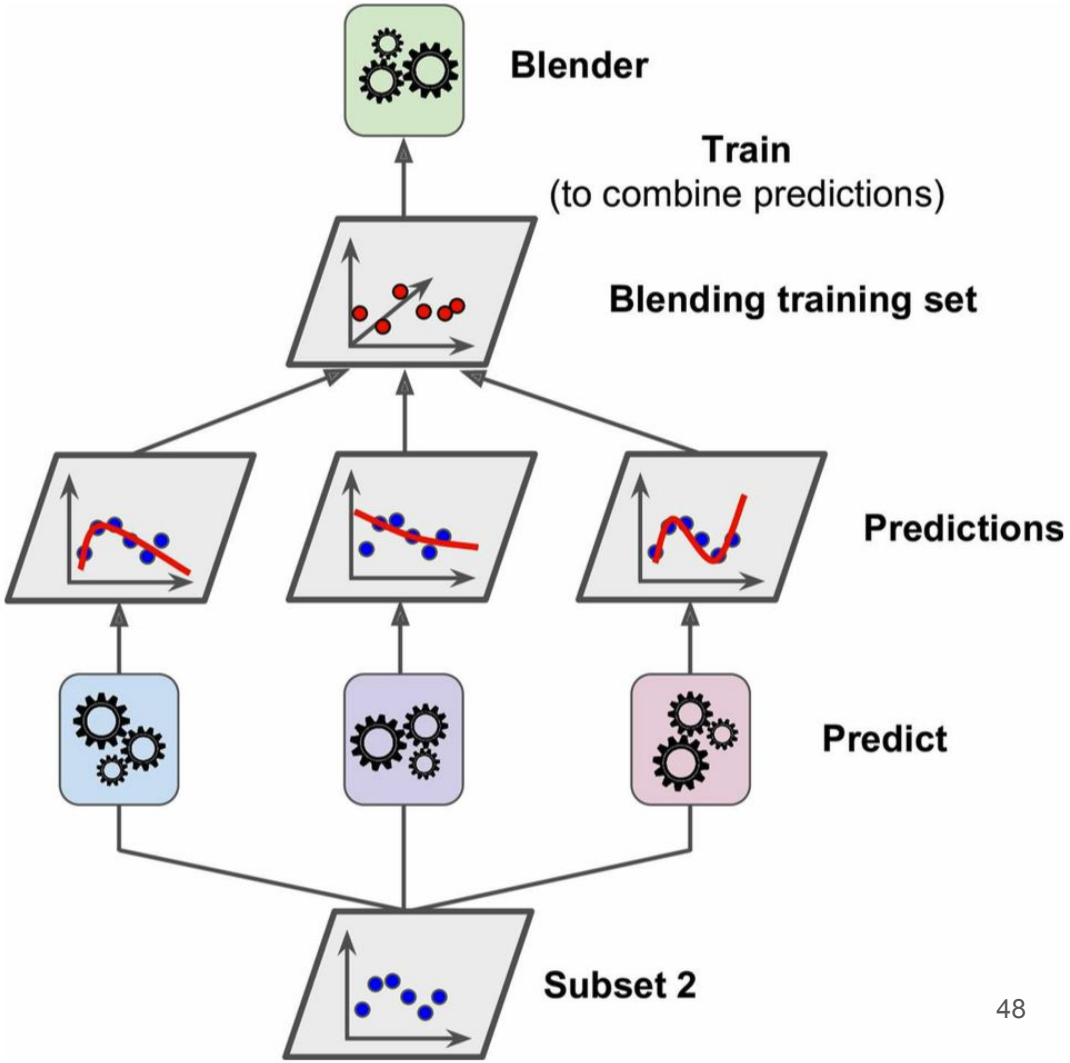
Instead of using a trivial function (such as hard voting) to aggregate the predictions of the ensemble, **we train a model to perform the aggregation**.



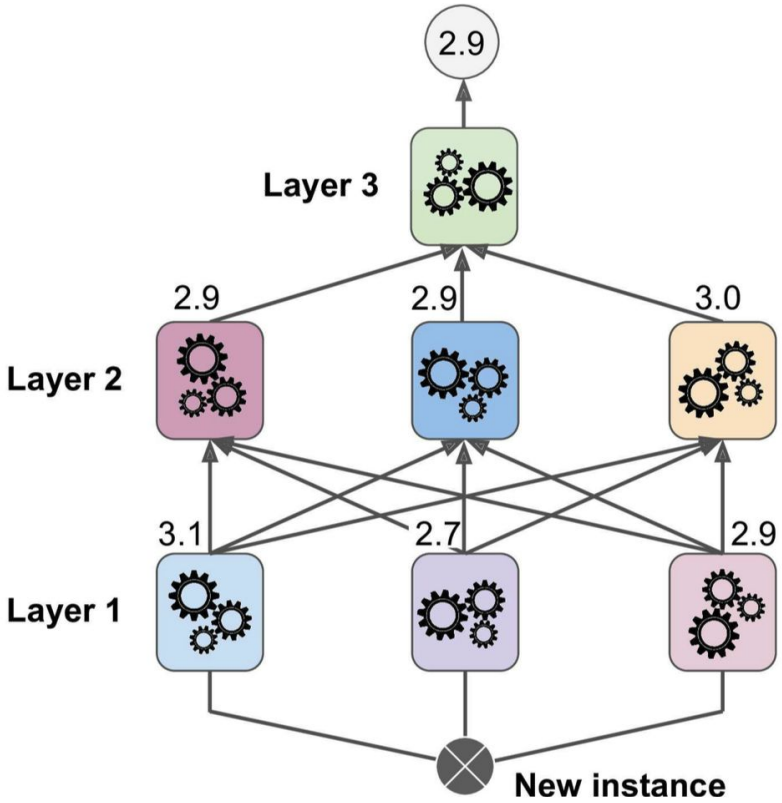
Training the predictors



Training the blender



Prediction in a multi-layer stacking ensemble



This resembles a neural network…
(more next week!)

Summary: Learning Objectives

- ✓ Ensemble Learning
- ✓ Bagging and Pasting
- ✓ Random Forest
- ✓ Boosting
- ✓ Stacking

Bonus Slides

When Bagging works

Main property of Bagging:

- Decreases variance of the base model **without** changing the bias
- Why does it work? By averaging!

Bagging typically helps when applied with an overfitting **base** model

In practice:

- Since models are correlated, so variance reduction is smaller than `1/N`
- Variance of models trained on few training instances usually larger.

Why do Ensemble Learning Work?

Based on one of 2 basic observations:

- **Variance reduction:** if the training sets are completely independent, it will always help to average an ensemble because this will reduce variance without affecting bias (e.g. bagging)
- **Bias reduction:** for simple models, average of models can reduce bias substantially by increasing capacity, and control variance by fitting one component at a time (e.g. boosting)

The Law of Large Numbers

10 Series of biased coin tosses. The ratio of head approaches 51%.