



# Machine Learning

(Học máy – IT3190E)

**Khoat Than**

School of Information and Communication Technology  
Hanoi University of Science and Technology

2022

# Contents

---

- Introduction to Machine Learning
- **Supervised learning**
  - **Ensemble learning**
- Unsupervised learning
- Reinforcement learning
- Practical advice

# Theoretical study

---

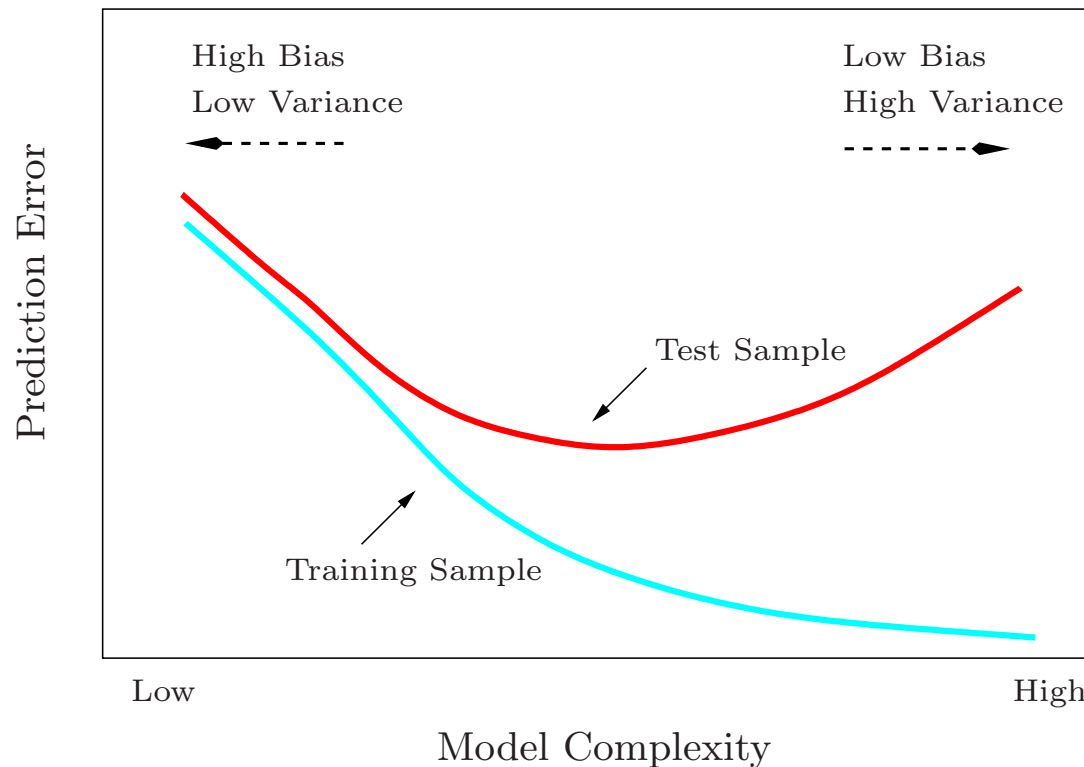
1. Bias-variance tradeoff
2. Strategies for combining classifiers
  - a) Majority voting (averaging)
  - b) Weighted average
  - c) Gating
  - d) Stacking
3. Ensemble learning methods based on data sampling
  - a) Bagging
  - b) Random subspaces
4. Example method based on data sampling: Random Forests
5. Example method based on classifier stacking: Boosting

# 1. Bias - Variance tradeoff

# BIAS - VARIANCE TRADEOFF

5

- High bias typically occurs when underfitting the data
- High variance typically occurs when overfitting the data



- Beyond the problems linked to underfitting/overfitting, some predictive models are inherently **unstable/sensitive**
  - ❖ Small differences in the learning dataset might lead to very different predictions (regression/classification), e.g.:
    - Sensitivity to outliers
    - Sensitivity to noises
  - ❖ Resulting in a large variance in the prediction

- Decision trees are unstable

- ❖ Example from <https://towardsdatascience.com/the-indecisive-decision-tree-story-of-an-emotional-algorithm-1-2-8611eea7e397>

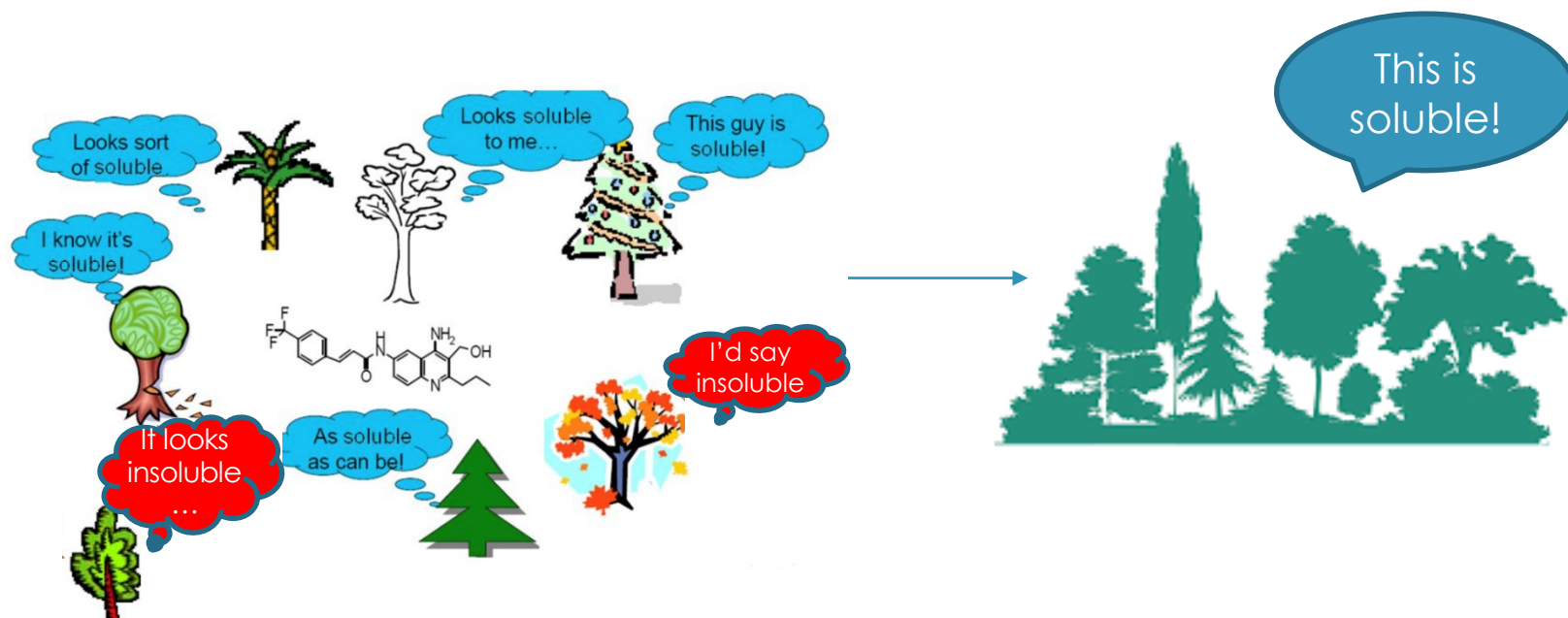
I trained 10 decision trees having exact same hyper-parameters with 10 different seed values, which ensures that each tree trains with a slightly different sample. Let's look at the top three most important variables, which represent structure of the tree, obtained by the trees for both the data sets.

	seed	train_acc	valid_acc	top_feature	second_feature	third_feature
0	0	0.903439	0.785185	Title	Fare	Age
1	288	0.888889	0.748148	Sex	Age	Fare
2	576	0.879630	0.740741	Title	Fare	Age
3	864	0.892857	0.792593	Sex	Age	Fare
4	1152	0.883598	0.748148	Sex	Age	Fare
5	1440	0.886243	0.725926	Title	Age	Sex
6	1728	0.895503	0.703704	Fare	Sex	Age
7	2016	0.894180	0.770370	Title	Fare	Age
8	2304	0.902116	0.792593	Sex	Pclass	Fare
9	2592	0.871693	0.777778	Title	Age	Fare
10	2880	0.898148	0.785185	Title	Age	Pclass

# SOLUTION: ENSEMBLE LEARNING

8

- Idea of ensemble learning methods:
  - ❖ “United we stand, divided we fall”
- Objective of ensemble learning methods
  - ❖ Reducing variance/instability of individual models by combining multiple models





- Ensemble learning can be applied for both unsupervised and supervised learning
  - ❖ Supervised learning: regression or classification
    - *E.g.* random forests
  - ❖ Unsupervised learning: clustering ensembles
    - *E.g.* consensus clustering
- Ensemble learning can be seen as a special kind of **meta-learning**
- In this lecture, we will focus mostly on classification

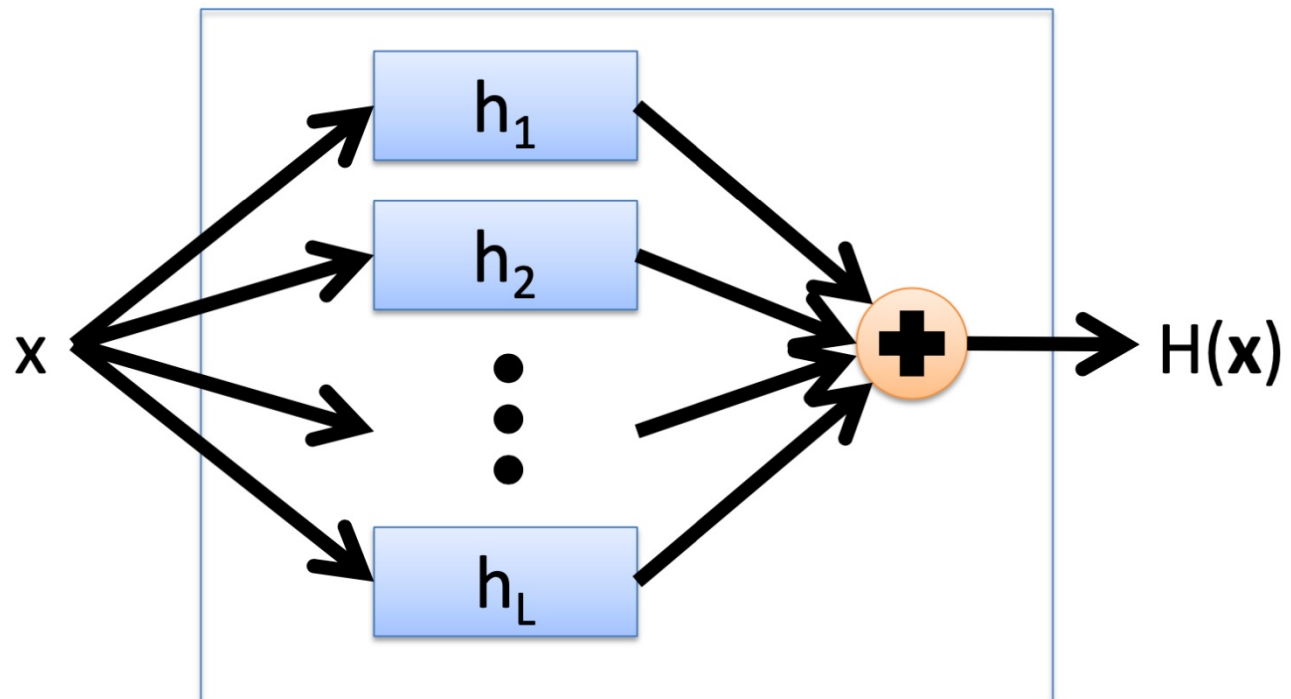
## 2. Strategies for combining classifiers

- Consider a set of classifiers  $h_1, \dots, h_L$  being **diverse**
  - ❖ Maybe from the same model, but making different mistakes (e.g. trees in the random forest)
  - ❖ Maybe based on different models (e.g. SVM + naive Bayes)
- **Idea:** construct a classifier  $H(\mathbf{x})$  that combines the individual predictions of  $h_1, \dots, h_L$ 
  - ❖  $h_1, \dots, h_L$  might return different predictions, or
  - ❖  $h_1, \dots, h_L$  might focus on different regions of the representation space
- $H(\mathbf{x})$  is sometimes called a **meta-classifier**

# MAJORITY VOTING (AVERAGING)

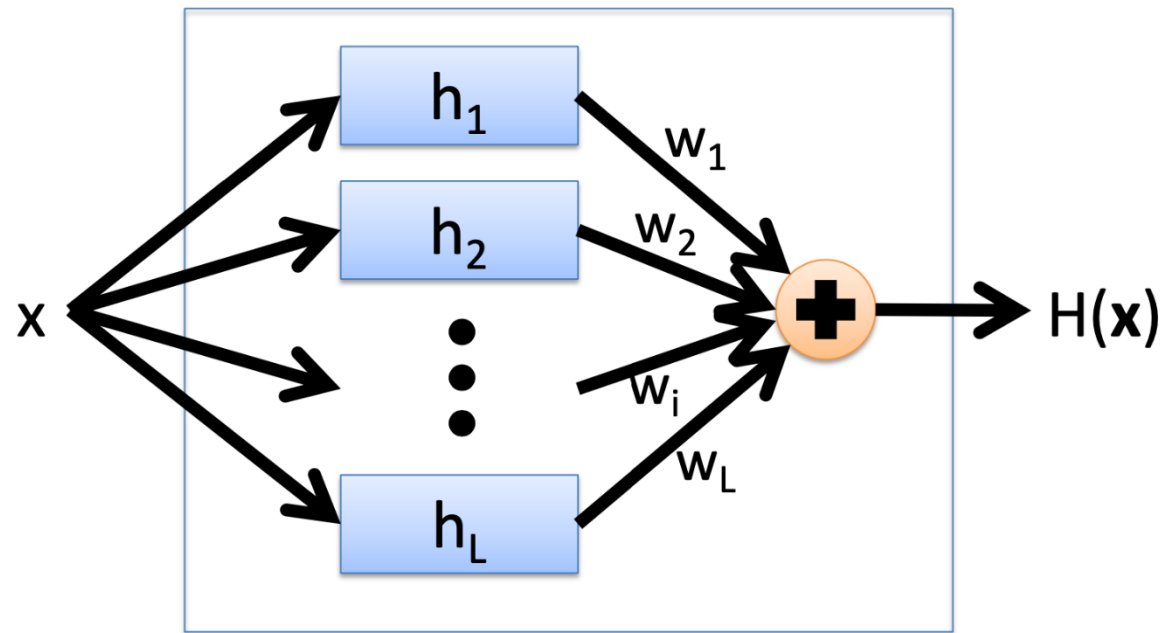
12

- $H(\mathbf{x})$  is simply the majority vote
  - ❖ Or, the average output for regression
  - ❖ This is the most usual strategy for random forests



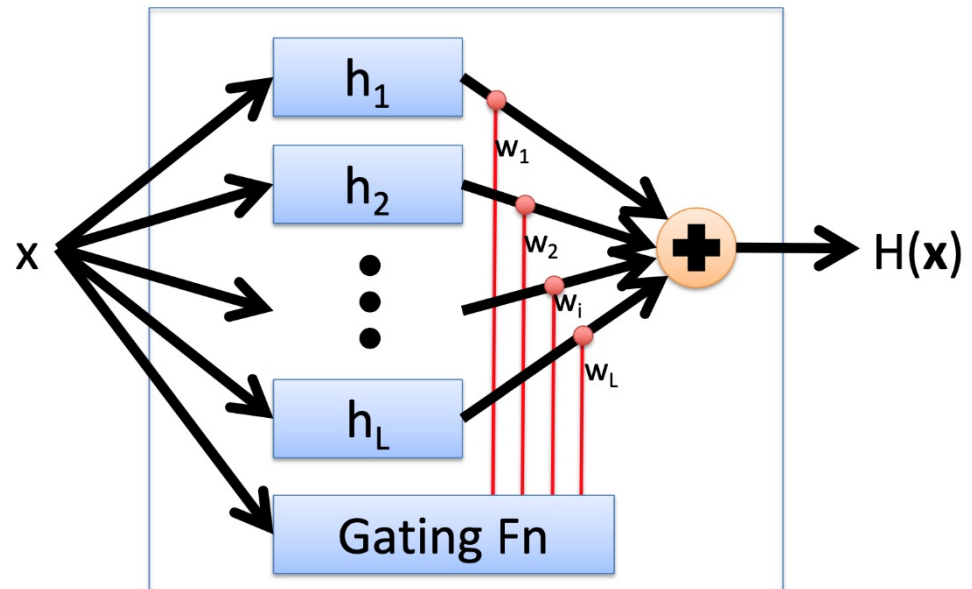
[[https://courses.cs.washington.edu/courses/cse446/20wi/Lecture16/16\\_EnsembleMethods.pdf](https://courses.cs.washington.edu/courses/cse446/20wi/Lecture16/16_EnsembleMethods.pdf)]

- $H(\mathbf{x})$  is simply a weighted majority vote
  - ❖ Or, the weighted average output for regression
  - ❖ The weights  $w_1, \dots, w_L$  are learnt from a validation dataset

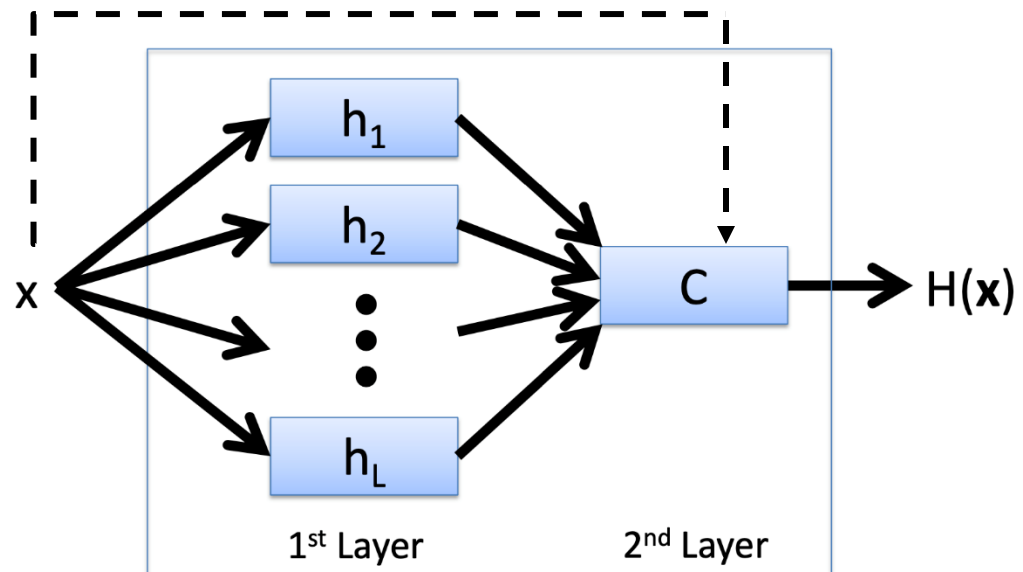


[[https://courses.cs.washington.edu/courses/cse446/20wi/Lecture16/16\\_EnsembleMethods.pdf](https://courses.cs.washington.edu/courses/cse446/20wi/Lecture16/16_EnsembleMethods.pdf)]

- This is a special case of weighted average where the weights  $w_1, \dots, w_L$  depend on the input(s)
  - ❖ The gating function selects/weights the best models for each problem
  - ❖ The gating function is learnt from a validation dataset



- Consists in stacking layers of classifiers
  - ❖ The predictions of the 1<sup>st</sup> layer are used as an input for the 2<sup>nd</sup> layer
  - ❖ The second layer is trained on a validation dataset
  - ❖ One might stack more than 2 layers of classifiers



- Consider a set of classifiers  $h_1, \dots, h_L$  being **diverse**
  - ❖ Maybe based on different models (e.g. SVM + naive Bayes)
    - In this case, the classifiers can be learnt from the same learning dataset (diversity is achieved by the diversity of the models)
  - ❖ Maybe from the same model, but making different mistakes (e.g. trees in the random forest): models can be
    - Learnt from different samplings of the training dataset
      - **Random** strategies
        - e.g. bagging, random subspaces
    - Stacked
      - **Adaptive** strategies
        - e.g. boosting

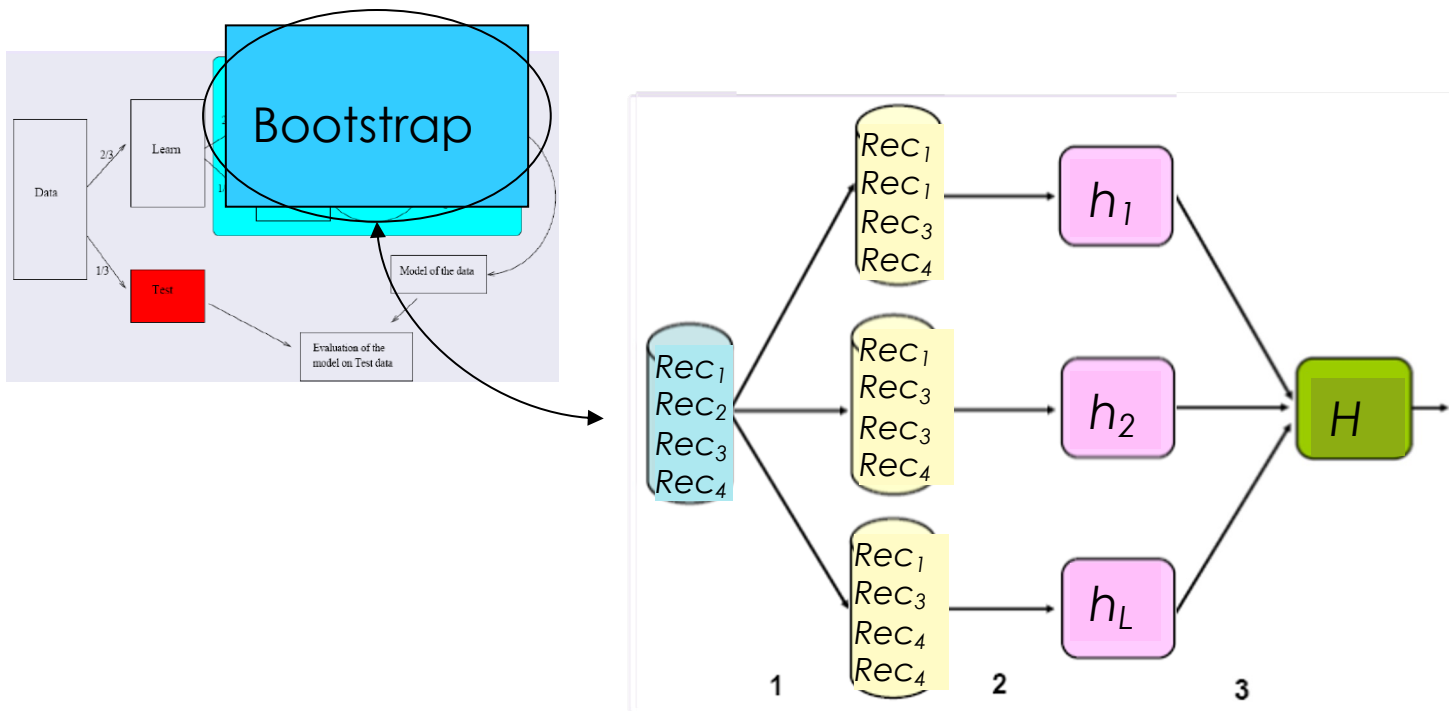


### 3. Ensemble learning methods based on data sampling

- A bootstrap sample is built from an initial data set
  - ❖ By sampling  $n$  instances with replacement
  - ❖ Excludes ~35% of the training instances
- A bootstrap sample contains as many objects as the original sample, but...
  - ❖ Some objects are replicated several times
  - ❖ Others are missing
- The selection of the  $n$  objects to be replicated is ALMOST random
  - ❖ For example, bootstrap samples might be further **stratified** (mostly for classification)
    - Keeps the class distribution of the initial data set

## ■ Bagging [Breiman, 1996]

1. Creating  $L$  bootstrap samples
2. Learning  $L$  classifiers  $h_i$
3. Combining the  $L$  classifiers  $h_1, \dots, h_L \rightarrow H$  (final classifier)



- Objective of bagging:
  - ❖ Reduce the effects of outliers / instances that are « too influential »
- The performance of the individual classifiers  $h_i$  is evaluated using out-of-bootstrap data
- For step 3 (classifier combination), one can use:
  - ❖ Averaging
  - ❖ Weighted average
  - ❖ Gating

- Advantages

- ❖ Can reduce the detrimental effects of outliers
- ❖ Can enhance performances

- Disadvantages

- ❖ Higher computational complexity
- ❖ Higher space complexity

- Objectives of random subspaces [Ho, 1998]:
  - ❖ Reduce the effects of ill-prepared attributes on the final model
  - ❖ Reduce the effects of attributes (features) correlation on the final model
  - ❖ Circumvent the problems that arise when the number of samples in the training dataset is insufficient compared to the number of attributes
- The  $L$  classifiers  $h_1, \dots, h_L$  are built from all examples
- Each classifier  $h_i$  is built from  $q$  attributes
  - ❖ Randomly selected from the whole set of attributes
  - ❖ In general,  $q \ll p$  (where  $p$  is the initial number of attributes)

- For step 3 (classifier combination), one can use:
  - ❖ Averaging
  - ❖ Weighted average
  - ❖ Gating

## ■ Advantages

- ❖ Can effectively reduce the effects of ill-prepared attributes on the final model
- ❖ Can effectively reduce the effects of attributes (features) correlation on the final model
- ❖ Can effectively circumvent the problems that arise when the number of samples in the training dataset is insufficient compared to the number of attributes

## ■ Disadvantages

- ❖ Higher computational complexity
- ❖ Higher space complexity
- ❖ Loss of readability / interpretability, for some models
  - e.g. for random forests

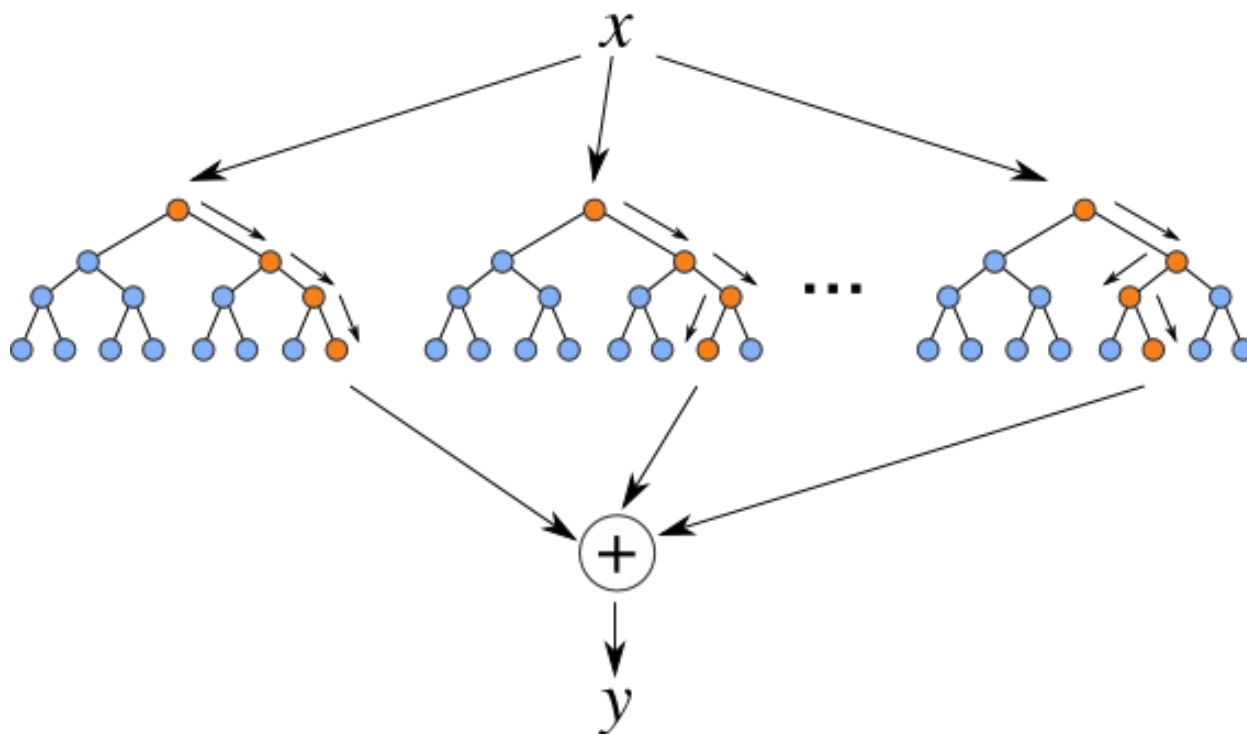


## 4. Methods based on data sampling

### Random Forests

- Random forests might be used for classification or regression
- Somehow combines bagging and random subspaces, as
  - ❖ Each tree in the forest is built:
    - Using bootstrap replicas
    - Restrict the node decisions to a small subset of features picked randomly for each node (node-level random subspaces)

- In its original version [Ho, 1995]
  - ❖ The trees in the forest are not pruned
  - ❖ The outputs of the trees are averaged
  - ❖ Might be used for classification or regression



- Documentation of RandomForestClassifier in the scikit-learn Python library:
  - ❖ <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
  - ❖ N.B. Different strategies for pre-pruning the trees are possible
- A comprehensive tutorial about Random Forests in Python:
  - ❖ <https://towardsdatascience.com/random-forest-in-python-24d0893d51c0>

# RANDOM FORESTS

---

## ■ Advantages

- ❖ Better stability (effectively reduces variance)
- ❖ On average,  $H$  (random forest) achieves better performances than the  $h_i$  (individual trees)

## ■ Disadvantages

- ❖ More computationally expensive
  - A lot more for the learning stage
  - A bit more for the classification stage
- ❖ Loose the readability / interpretability of the individual trees  $h_i$

# 5. Methods based on classifier stacking

Boosting

- Consider a set of classifiers  $h_1, \dots, h_L$  being **diverse**
  - ❖ Maybe based on different models (e.g. SVM + naive Bayes)
    - In this case, the classifiers can be learnt from the same learning dataset (diversity is achieved by the diversity of the models)
  - ❖ Maybe from the same model, but making different mistakes (e.g. trees in the random forest): models can be
    - Learnt from different samplings of the learning dataset
      - **Random** strategies
        - e.g. bagging, random subspaces
    - **Stacked**
      - **Adaptive** strategies
        - e.g. boosting

- Ensemble Learning models based on data sampling can effectively reduce the variance of the model
- But, they might still suffer from a large bias
  - ❖ At least, in some regions of the data space (*i.e.* for some examples)
- Boosting aims at reducing the bias




## ■ AI: « How can I help you? »

[Gorin et al.]

- **goal:** automatically categorize type of call requested by phone customer (`Collect`, `CallingCard`, `PersonToPerson`, etc.)
  - yes I'd like to place a collect call long distance please (`Collect`)
  - operator I need to make a call but I need to bill it to my office (`ThirdNumber`)
  - yes I'd like to place a call on my master card please (`CallingCard`)
  - I just called a number in sioux city and I musta rang the wrong number because I got the wrong party and I would like to have that taken off of my bill (`BillingCredit`)
- **observation:**
  - **easy** to find "rules of thumb" that are "often" correct
    - e.g.: "IF 'card' occurs in utterance  
THEN predict 'CallingCard' "
  - **hard** to find **single** highly accurate prediction rule

Simple decision rules (often from tacit knowledge)

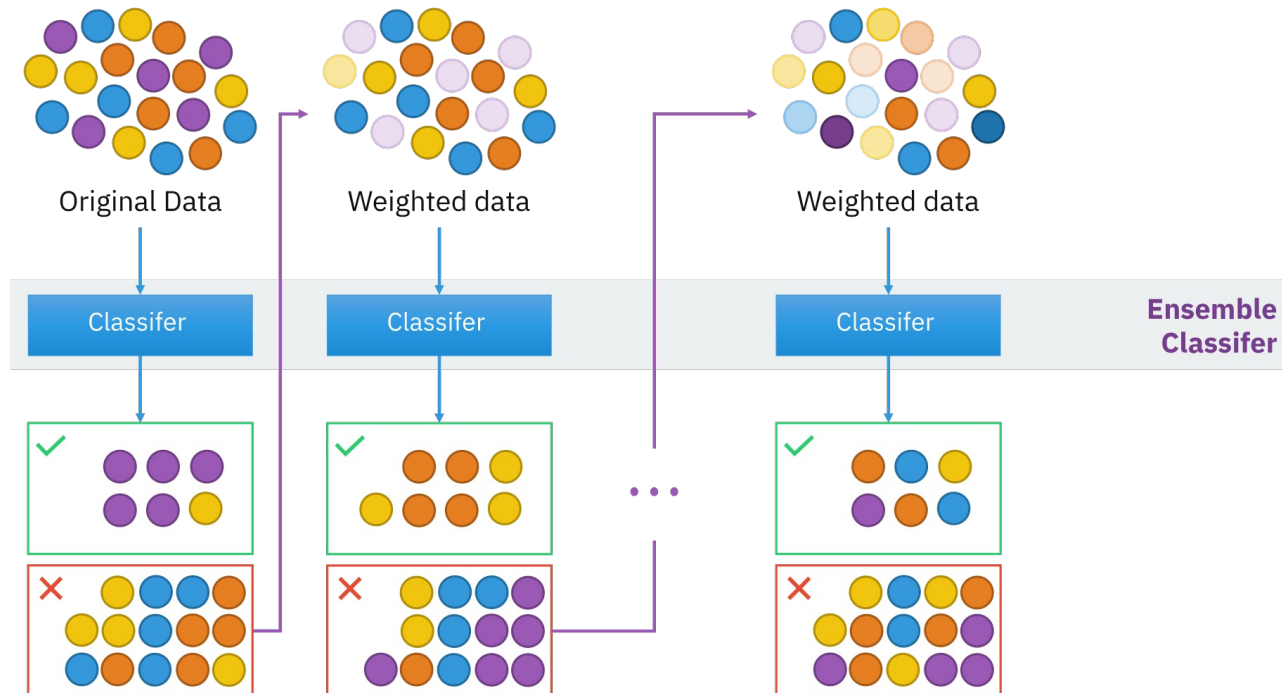


- Idea: sequentially derives weak classifiers (can be rules of thumb)
  - ❖ Apply procedure to subset of examples
  - ❖ Obtain the first weak classifier
  - ❖ Apply to 2nd subset of examples
  - ❖ Obtain 2nd weak classifier
  - ❖ Repeat  $L$  times
- How to choose examples on each round?
  - ❖ Concentrate on “hardest” examples
    - *I.e.* the most often misclassified by previous classifiers

- Technically:

- ❖ Assume given “weak” classifiers  $h_t$  that perform slightly better than random (i.e. accuracy  $\geq 55\%$  for 2-class problems)
  - ❖ We will stack them in a cascade, such that  $h_{t+1}$  focuses on samples that are misclassified by  $h_t$  (difficult examples)
- Given a sufficient number of training samples, a boosting algorithm can provably construct a meta-classifier  $H$  with much better accuracy (up to 99%)

- Principle: building a cascade of « weak » classifiers  $h_t \dots$
- ... Each individual classifier  $h_t$  aims at focusing on the « hard samples » that were incorrectly classified by its predecessor in the cascade  $h_{t-1}$



- In practice, there are two main strategies:
  - ❖ **Arcing:** augmenting the probability of selecting these « hard samples » in the training dataset of classifier  $h_t$
  - ❖ **AdaBoost:** augmenting the weight of these « hard samples » when learning the classifier  $h_t$
- In general, the outputs of the classifiers are combined using (weighted) averaging

- Example of practical application of boosting
  - ❖ **Face detection** from numeric images [Viola&Jones, 2001]
    - Find faces in photograph or movie using Adaboost algorithm
    - Weak classifiers: detect light/dark rectangles in image
    - Many clever tricks to it make extremely fast and accurate
    - Up until today, still a state-of-the-art method for face detection



## ■ Advantages

- ❖ Reduces the effect of « difficult records » (at the frontier between classes)
- ❖ Often gives very good results in practice
  - If there is a large number of records in the training dataset, boosting often gives better results than bagging

## ■ Disadvantages

- ❖ More computationally expensive than the individual classifiers
- ❖ If there are not enough training samples:
  - Either the learning set is perfectly classified by the first classifier, and the boosting algorithm is useless
  - Either the algorithm focuses on a small number of samples, potentially unrepresentative of the classes, and the "boosted" classifier may perform worse than the original classifier...

# 5. Example of EL method based on classifier stacking AdaBoost



- 2-class problem formulation:
- Given a training set  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$ 
  - ❖  $\mathbf{x}_i$  being the vector of explanatory variables values for record  $i$
  - ❖  $y_i \in \{-1, +1\}$  being the true label of instance  $x_i \in X$  (2-class problem)
- For  $t = 1, \dots, L$ :
  - ❖ construct distribution  $D_t$  on  $\{1, \dots, n\}$
  - ❖ find weak classifier (“rule of thumb”)  $h_t : X \rightarrow \{-1, +1\}$ 
    - with error  $\varepsilon_t$  on  $D_t$ :
  - ❖  $\varepsilon_t = P_{i \sim D_t}(h_t(x_i) \neq y_i)$
- output final/combined classifier  $H$

- Construct distribution  $D_t$  on  $\{1, \dots, n\}$

given  $D_t$  and  $h_t$ :

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases} \\ &= \frac{D_t(i)}{Z_t} \exp(-\alpha_t y_i h_t(x_i)) \end{aligned}$$

Weight of sample  $i$  in the learning dataset of classifier  $h_{t+1}$ :

- weights of correct predictions are multiplied by  $e^{-\alpha_t} \leq 1$
- weights of incorrect predictions are multiplied by  $e^{\alpha_t} \geq 1$

where  $Z_t$  = normalization factor

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right) > 0$$

Weight of classifier  $h_t$  in the final decision (denoted  $w_t$  in the slide on average weighing)

- $\epsilon_t = \sum_{i: y_i \neq h_t(x_i)} D_t(i)$

Such that:

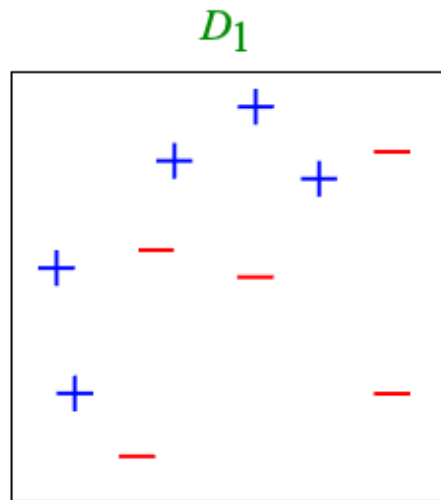
- $\alpha_t$  grows as  $\epsilon_t$  gets smaller
- if  $\epsilon_t \leq 0.5$ , then  $\alpha_t \geq 0$  (trivial, otherwise flip  $h_t$ 's prediction)

- Final classifier:

$$H(x) = \text{sign} \left( \sum_t \alpha_t h_t(x) \right)$$

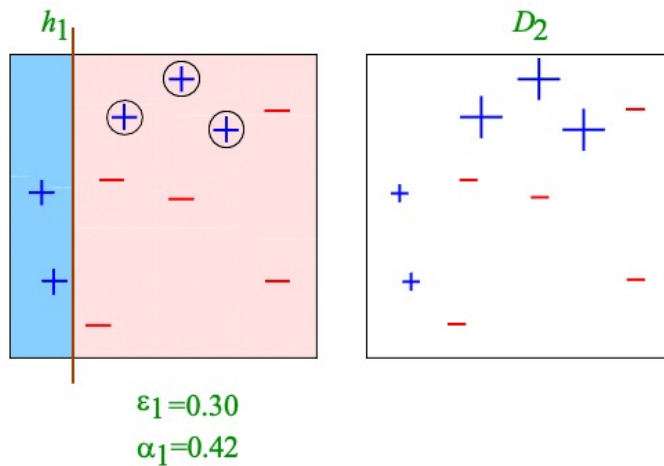
Error of  $h_t$  is the sum of the weights of all examples misclassified by  $h_t$

- Toy example (from Rob Schapire):
  - ❖ Weak classifiers  $h_t$  = vertical or horizontal half-planes that best discriminate the two classes
  - ❖ Initial learning dataset:



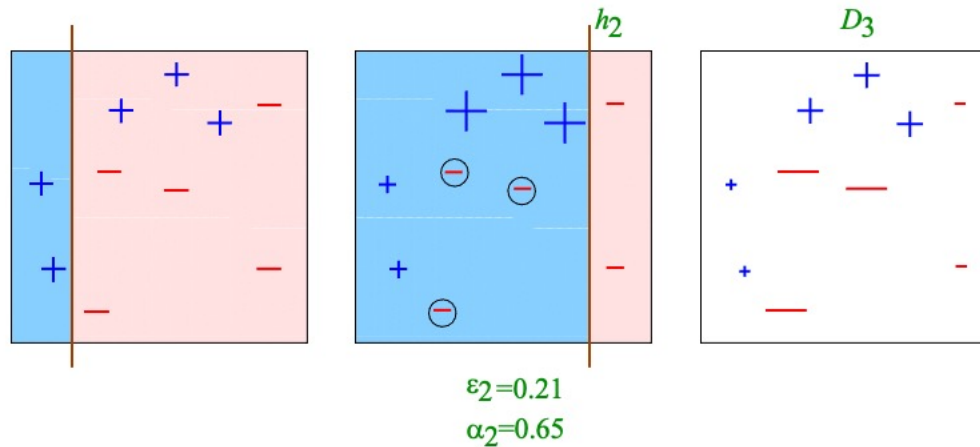
- Toy example (from Rob Schapire):

- ❖ Round 1:



- Toy example (from Rob Schapire):

- ❖ Round 2:

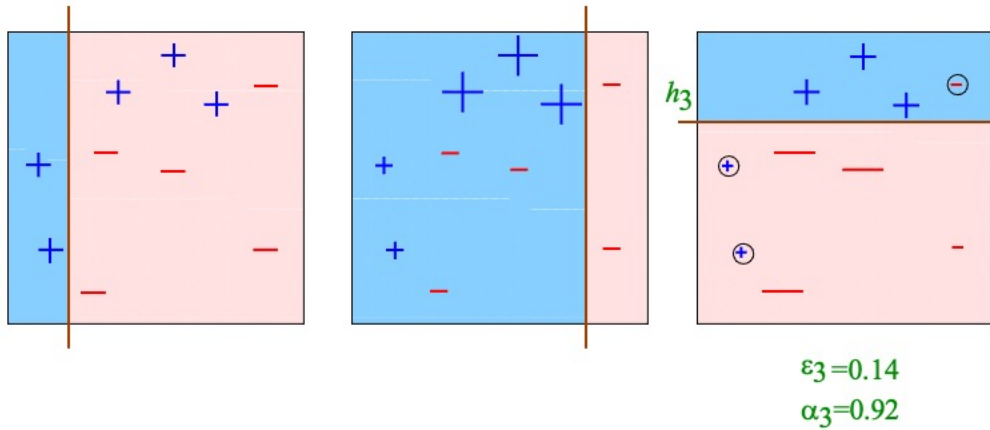


- ❖ The size of the point  $i$  in the above illustration is an indicator of  $D_t(i)$

# ADABOOST ALGORITHM

46

- Toy example (from Rob Schapire):
- Round 3:



# ADABOOST ALGORITHM

47

- Toy example (from Rob Schapire):
- Final classifier:

$$H = \text{sign} \left( 0.42 \begin{array}{|c|c|} \hline \text{blue} & \text{red} \\ \hline \end{array} + 0.65 \begin{array}{|c|c|} \hline \text{blue} & \text{red} \\ \hline \end{array} + 0.92 \begin{array}{|c|c|} \hline \text{blue} & \text{red} \\ \hline \end{array} \right)$$
  
$$= \begin{array}{|c|c|c|} \hline \text{blue} & \text{blue} & \text{red} \\ \hline \end{array}$$

# ADABOOST ALGORITHM

- Problem: how to weight the training samples during learning?
- For algorithms like logistic regression, one can simply incorporate weights  $D(i)$  into the cost function

$$J_{\text{reg}}(\boldsymbol{\theta}) = - \sum_{i=1}^n D(i) [y_i \log h_{\boldsymbol{\theta}}(\mathbf{x}_i) + (1 - y_i) \log (1 - h_{\boldsymbol{\theta}}(\mathbf{x}_i))] + \lambda \|\boldsymbol{\theta}_{[1:d]}\|_2^2$$

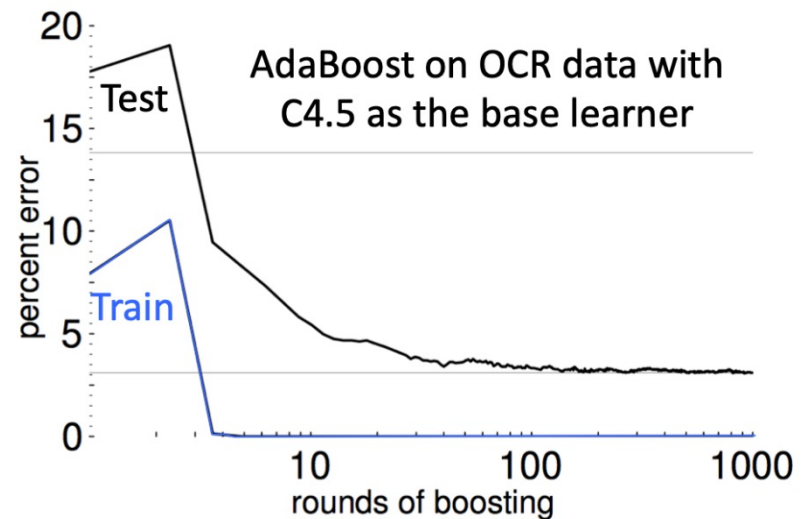
- ❖ Essentially, weights the cost of misclassification differently for each instance  $i$
- For algorithms that don't directly support instance weights (e.g. ID3 decision trees), use weighted bootstrap sampling
  - ❖ Form the training set by resampling instances with replacement according to  $D$



- It can be proven that the error on the training dataset (apparent error) goes to 0 in  $O(\log n)$  iterations
- AdaBoost works best with “weak” learners
  - ❖ Should not be complex
  - ❖ Typically highly biased classifiers
  - ❖ Works even when weak learners are just slightly better than random
    - Examples of such weak learners:
      - Decision stumps (1 level decision trees)
      - Depth-limited decision trees
      - Linear classifiers

- Successive weak learners focus on the hardest parts of the representation space
- In other words, if a point is repeatedly misclassified...
- ... Each time, its weight is increased...
- ... Eventually, it will be emphasized enough to generate a hypothesis (weak classifier) that correctly predicts it
  - ❖ But, instances with highest weight are often outliers
  - ❖ So, like all boosting-based methods, it only works well when there is a large number of records in the training dataset

- In theory, AdaBoost should overfit as the number of weak classifiers  $L$  grows large
  - ❖ *i.e.*, when the final classifier keeps growing more complex
- But in practice, AdaBoost often does not overfit
  - ❖ Provided there are enough training samples
  - ❖ Why?
    - it continues to drive down the test error even AFTER the training error reaches zero



[Figure from Schapire: "Explaining AdaBoost"]

# ADABOOST: DISCUSSION

---

## ■ Strengths:

- ❖ Fast
- ❖ Simple to program
- ❖ No parameter to tune (beside  $L$ )
- ❖ No assumption about the weak learners

## ■ Cases where boosting can fail:

- ❖ Insufficient / non-representative learning data
- ❖ Overly complex weak learners
- ❖ Lot of noise
- ❖ Large number of outliers

- A comprehensive tutorial about Viola & Jones' algorithm:
  - ❖ <https://realpython.com/traditional-face-detection-python/>
- Generalization of AdaBoost: **Gradient Boosting**
  - ❖ <https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/>
  - ❖ <https://machinelearningmastery.com/gradient-boosting-machine-ensemble-in-python/>

# References

---

- Breiman, Leo. Bagging predictors. *Machine Learning*, 24(2), pp.123-140, 1996
- Breiman, Leo. *Random forests*. *Machine Learning*, 45(1), 5-32, 2001.
- Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, Dinani Amorim. *Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?* *Journal of Machine Learning Research*, 15(Oct):3133–3181, 2014.
- Trevor Hastie, Robert Tibshirani, Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2017.
- Ho, T.K. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8), pp.832-844, 1998.