

# PATTERN RECOGNITION AND MACHINE LEARNING SYSTEMS DAY 5

*Dr Zhu Fangming*  
*Institute of Systems Science*  
*National University of Singapore*  
[\*fangming@nus.edu.sg\*](mailto:fangming@nus.edu.sg)

***Not be reproduced in any form or by any means, without the written permission of ISS, NUS, other than for the purpose for which it has been supplied.***

# DAY 5 AGENDA

5.1 Ensemble and Hybrid Machine Learning Techniques

5.2 Ensemble Workshop

# 5.1

## Ensemble and Hybrid Machine Learning Techniques

# Topics

- Intro to ensemble models
- Random Forest
- Boosting methods
- Hybrid machine learning models

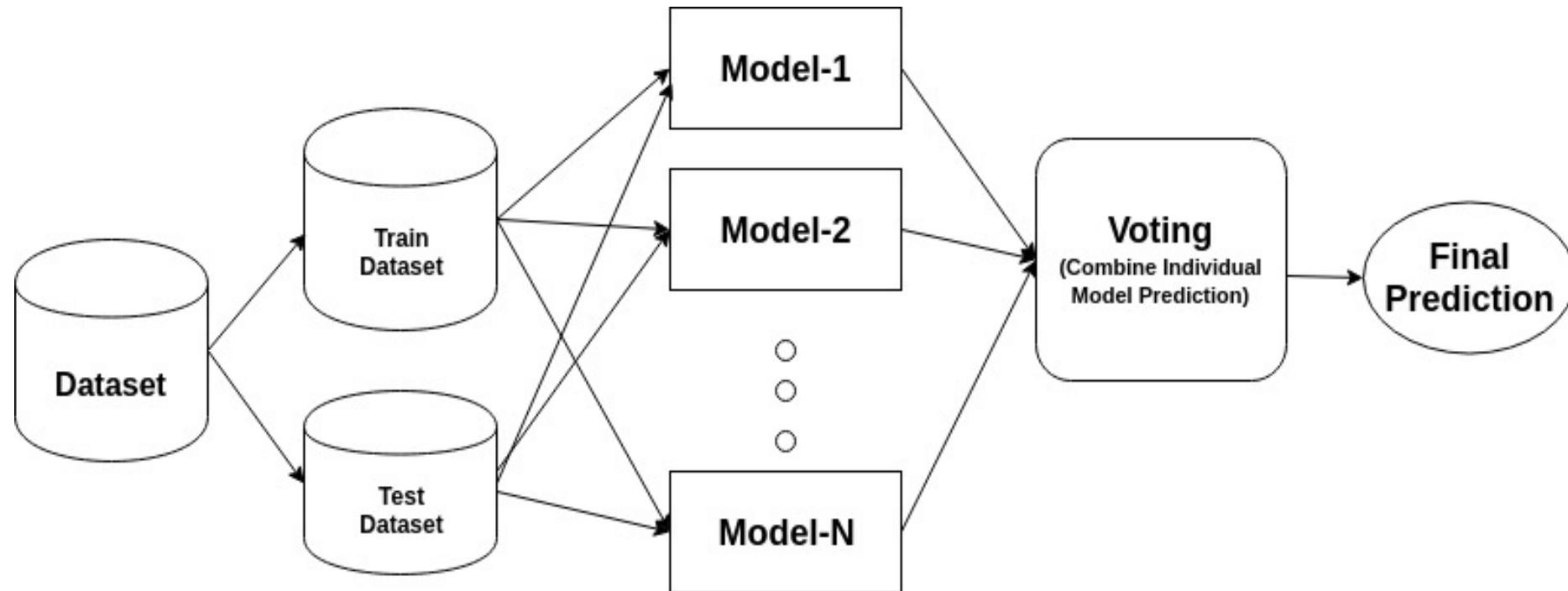
# What are Ensembles?

- Ensembles are committees of multiple models
- Each model makes a prediction or “vote”
- Final prediction is average/majority of votes
  - Majority vote for classification (class prediction)
  - Average prediction for regression problems (value prediction)



- What benefits does this bring?
- Why not just train one smart model?

# Ensemble



[www.datacamp.com](http://www.datacamp.com)

# Motivation

- Assume one model and 5 test cases

Truth	1	0	1	1	0	Accuracy
Model 1	1	0	0	1	1	60%

# Motivation

- Add another 4 models, each with same accuracy, but with variance (models do not give identical predictions)

Truth	1	0	1	1	0	Accuracy
Model 1	1	0	0	1	1	60%
Model 2	0	1	1	1	0	60%
Model 3	0	0	1	0	0	60%
Model 4	1	1	1	1	1	60%
Model 5	1	0	0	0	0	60%
Vote 1-5	1	0	1	1	0	100%

- No one model is very accurate, learns everything
- Performance of ensemble outperforms individuals
- Usually more reliable/robust than individual models



# What makes a good ensemble?

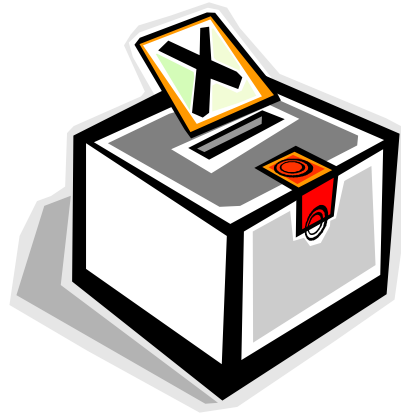
- The term *ensemble* is usually reserved for methods that generate multiple hypotheses using the same base learner

*“A necessary and sufficient condition for an ensemble of classifiers to be more accurate than any of its individual members is if the classifiers are **accurate** and **diverse**”*

-- Tom Dietterich (2000)

# How to get suitable diverse models?

- **Ensembles tend to yield better results when there is a significant diversity among the models**
- **Bagging is one way of introducing diversity**
  - Train many models with different random samples
  - Usually applied to decision trees, but can be used with any method

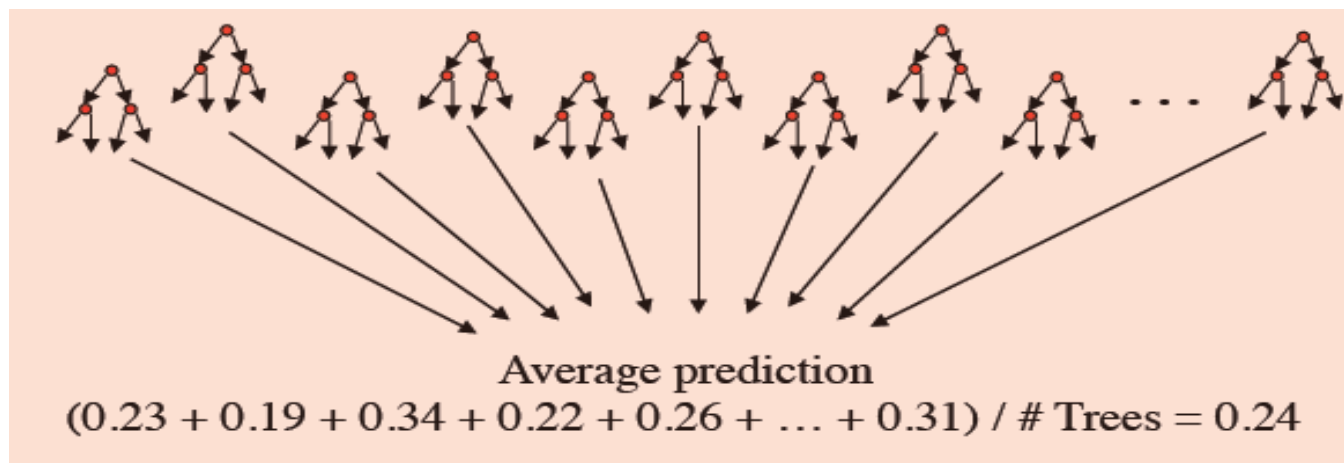


When using bagging, each learned model has a vote of equal value

- **Where do we get many different random samples?**

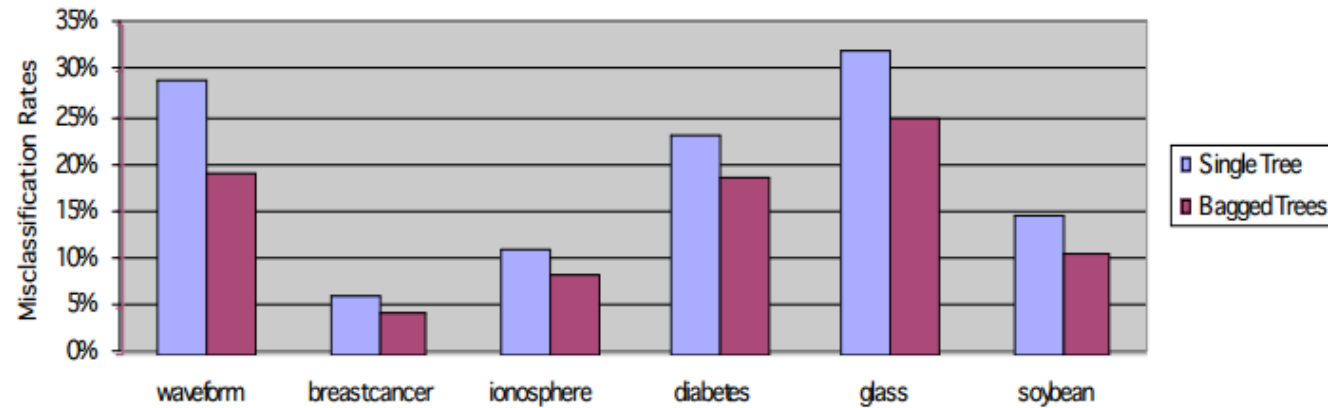
# Bagging: Bootstrap Aggregating

- Draw N (say 100) bootstrap samples (sampling with replacement) from the training data, Train models (eg. NN, decision tress) on each sample
- **Algorithm:**
  - Randomly draw 67% (say, two thirds) of the training data
  - Train a model on this sample
  - Repeat this N times to get N models
- **Take the un-weighted average prediction of all models**

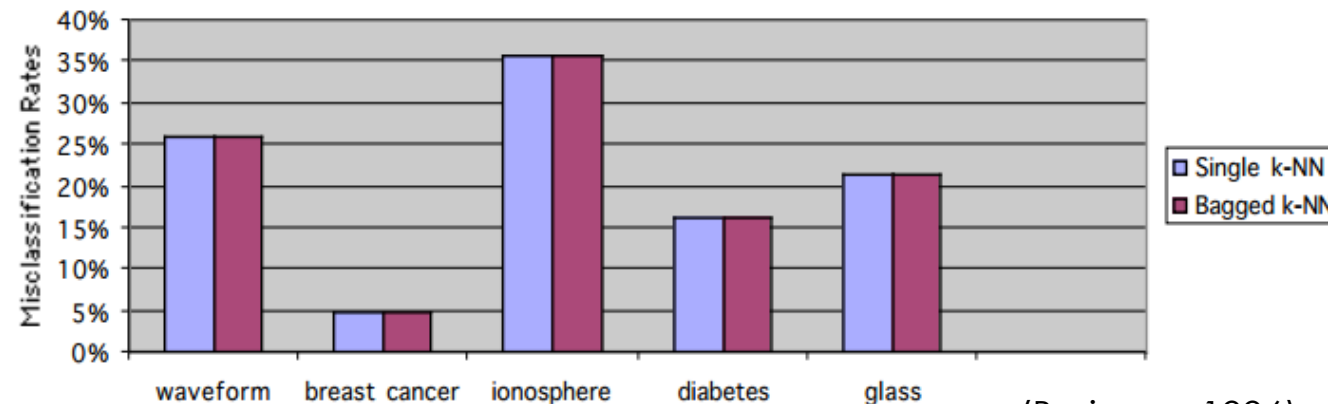


# Examples of Bagging

Single and Bagged Decision Trees (50 Bootstrap Replicates)  
Test Set Average Misclassification Rates over 100 Runs



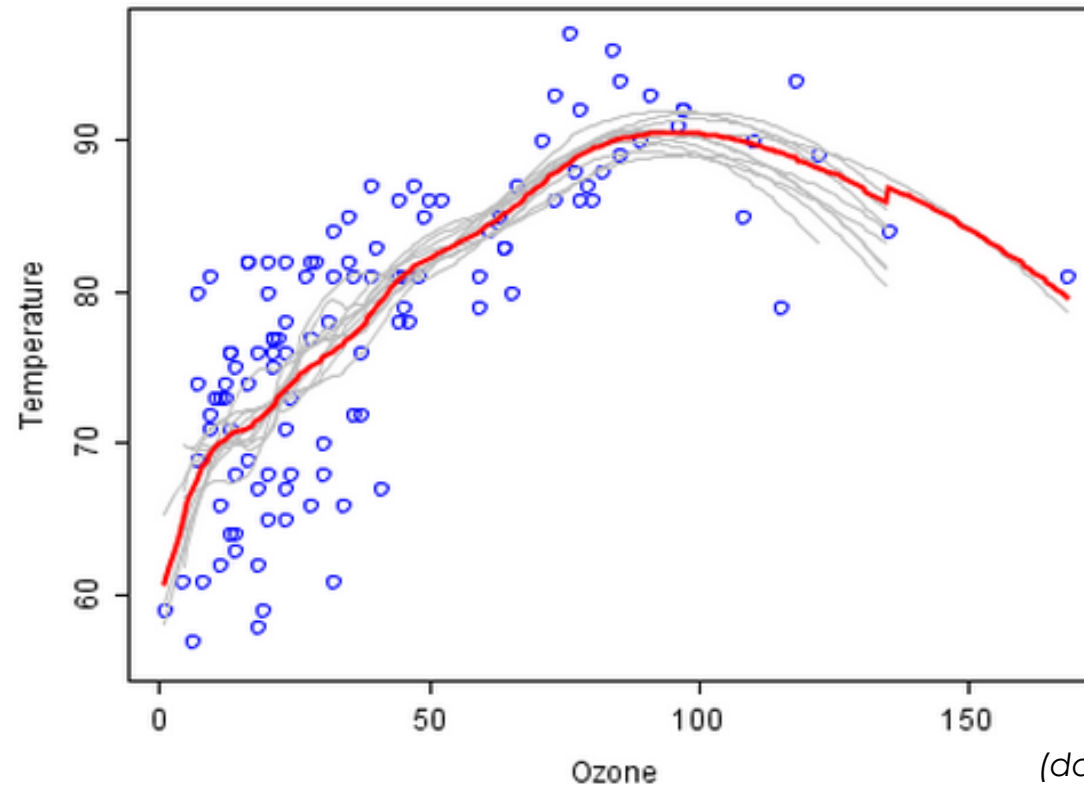
Single and Bagged k-NN (100 Bootstrap Replicates)  
Test Set Average Misclassification Rates over 100 Runs



(Breiman, 1996)

# Examples of Bagging

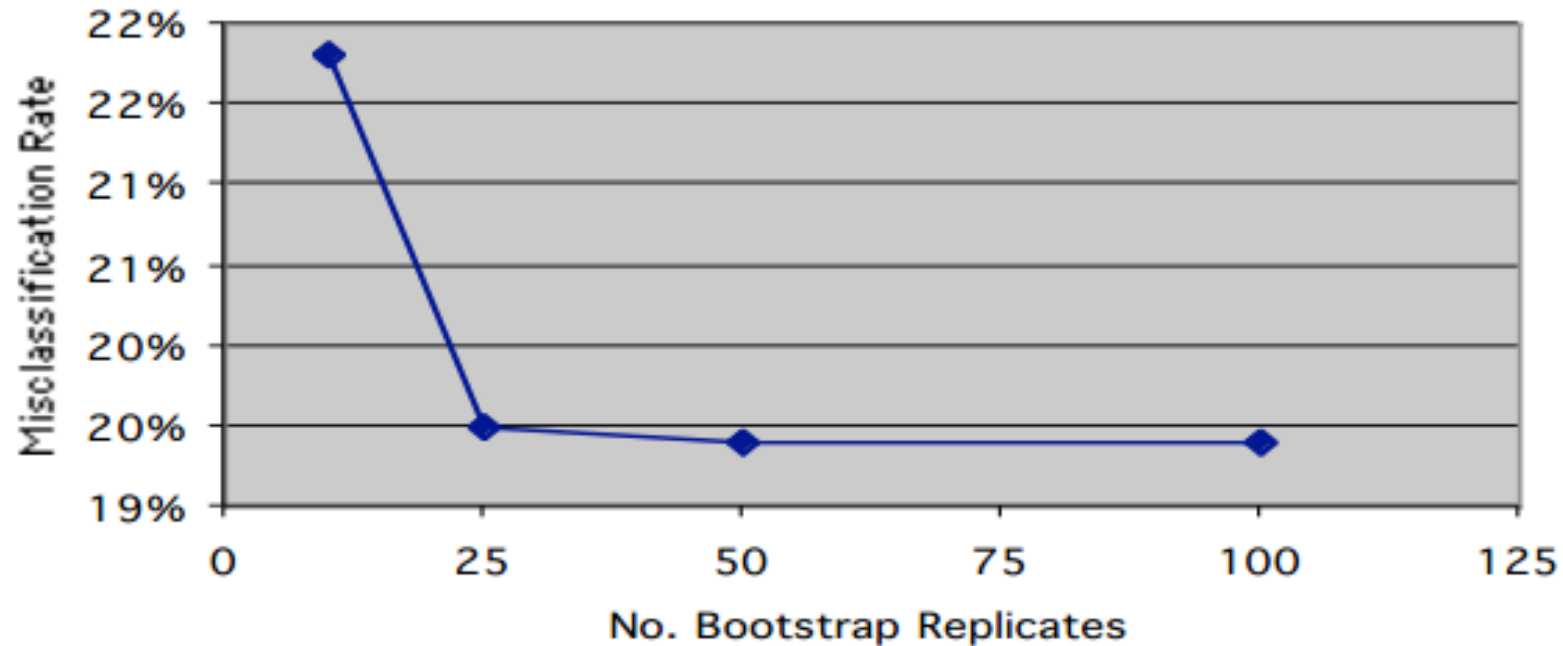
- Ensemble of 10 regression smoothers built from 10 bootstrap samples, each drawing 100 training data.



(data from Rousseeuw and Leroy).

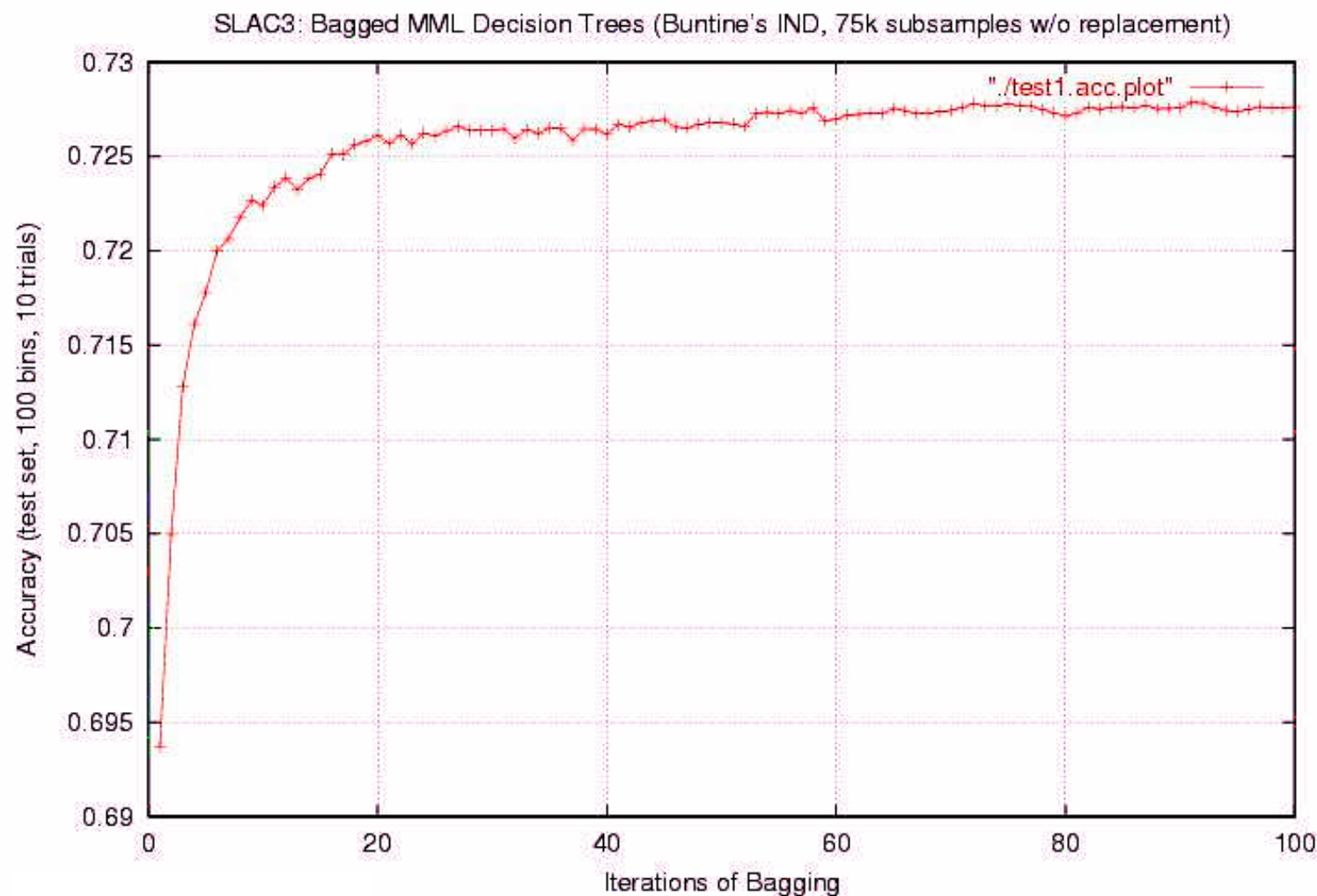
# How Many Bagged Models Are Required?

- Example from the soybean data set:



- Depends on data and problem, but generally, < 50 models should work well and often < 25 is adequate

# How Many Bagged Models Are Required?



# Why does it work? Model Error Reduction

- Assume we build a prediction model  $\hat{f}(X)$  to estimate a function  $f(X)$  where  $X$  is the set of input variables and  $Y$  is the variable to be predicted
- Then the expected squared prediction error at point  $x$  is:

$$Err(x) = E[(Y - \hat{f}(x))^2]$$

- We can decompose this into 3 components

$$Err(x) = \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\left(\hat{f}(x) - E[\hat{f}(x)]\right)^2\right] + \sigma_e^2$$

$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Noise, cannot be  
reduced by the  
model



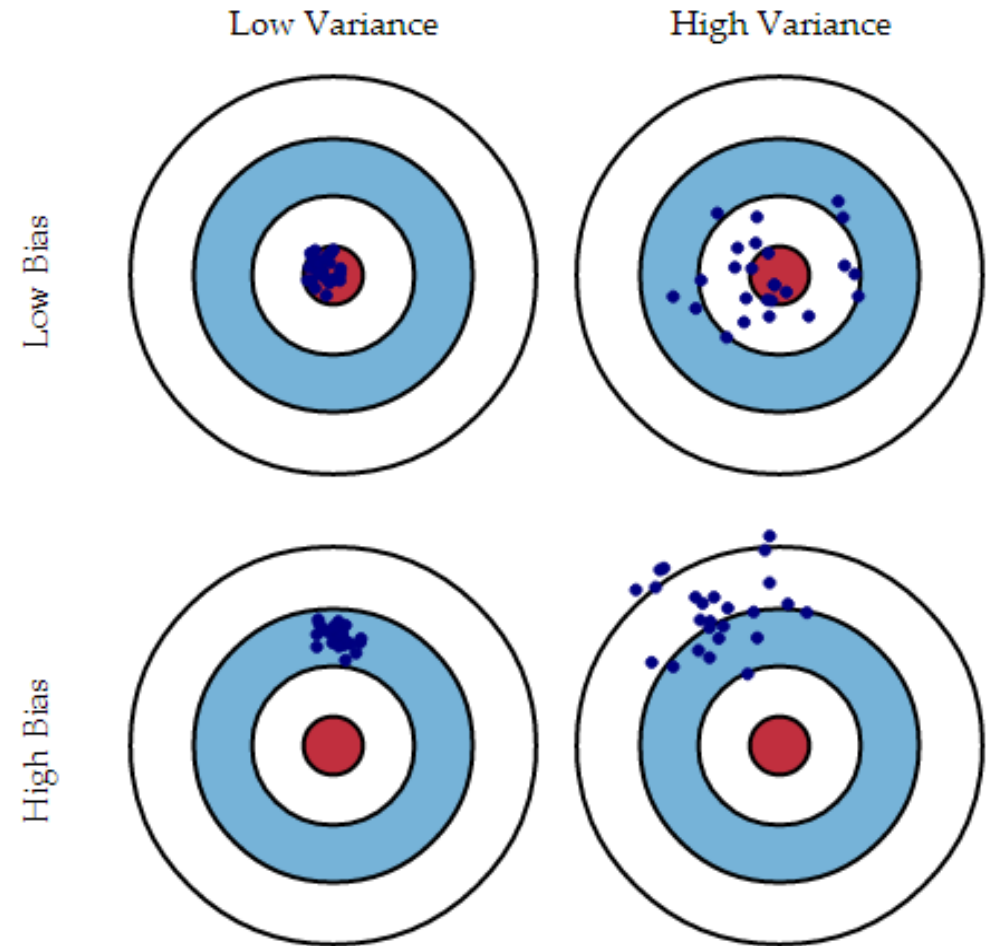
# Model Error: Bias versus Variance

- **Error due to Bias:**

- The difference between the expected (or average) prediction of the model and the correct value

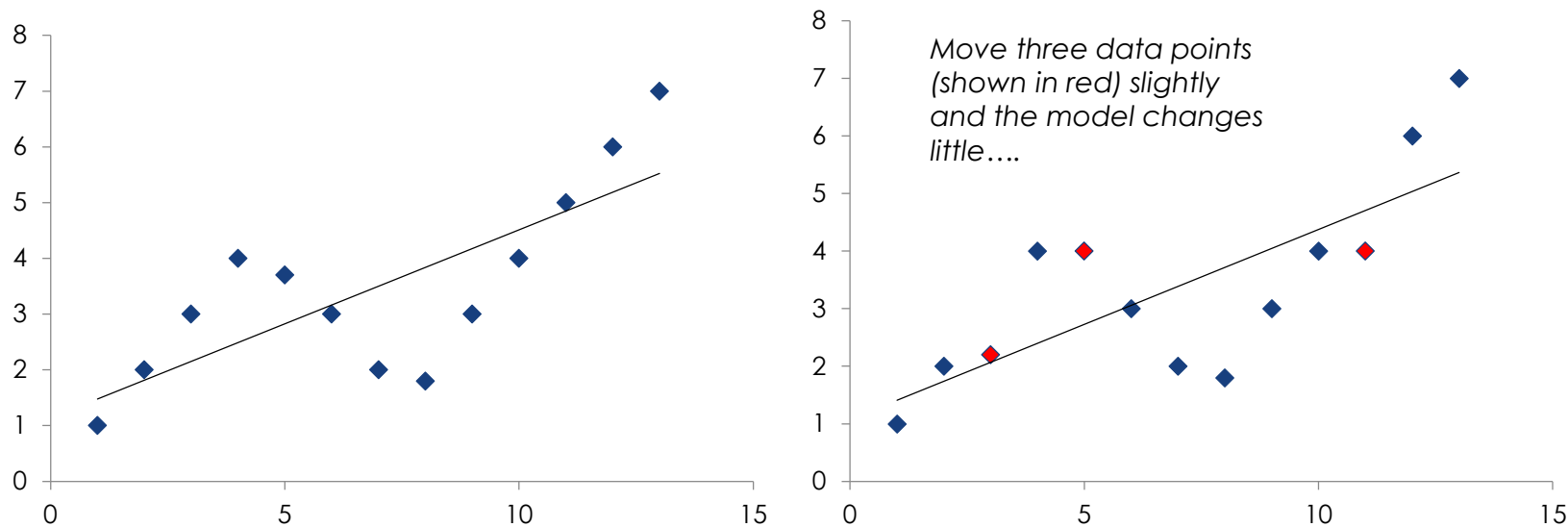
- **Error due to Variance:**

- The variability of the model prediction for a given data point



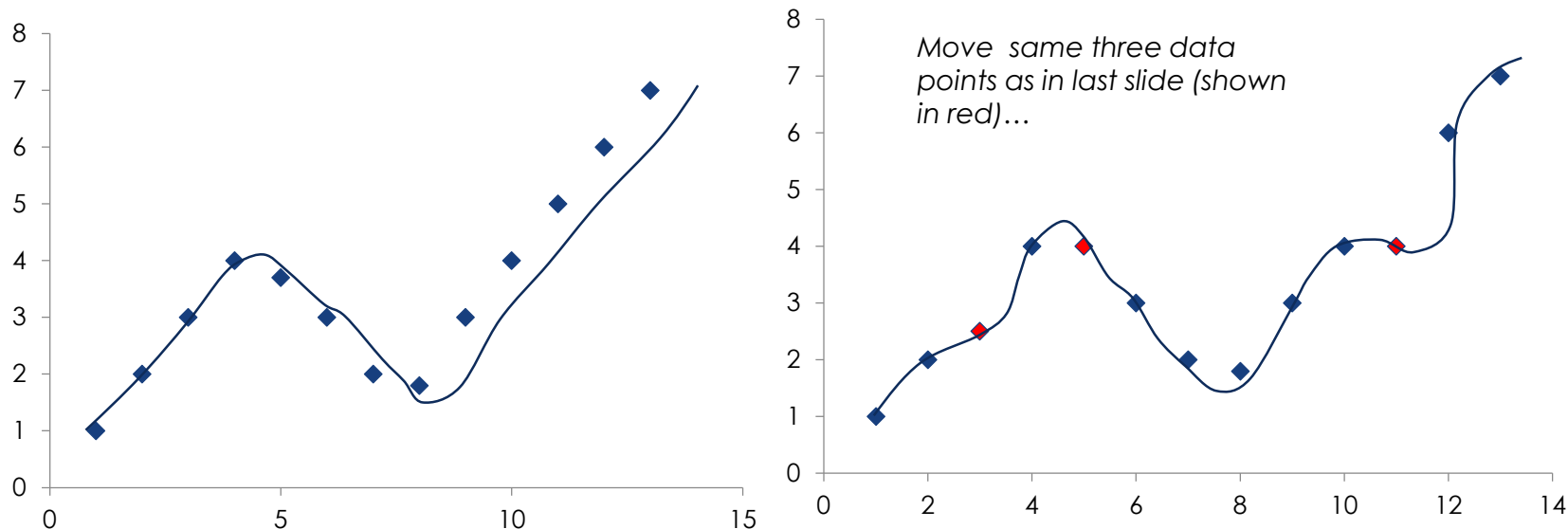
# Model Error: Bias versus Variance

- **Models that under-fit the data tend to have:**
  - High bias ~ the model doesn't fit the training data very well
  - Low variance – removing/changing a few training data points won't change the model or predictions much

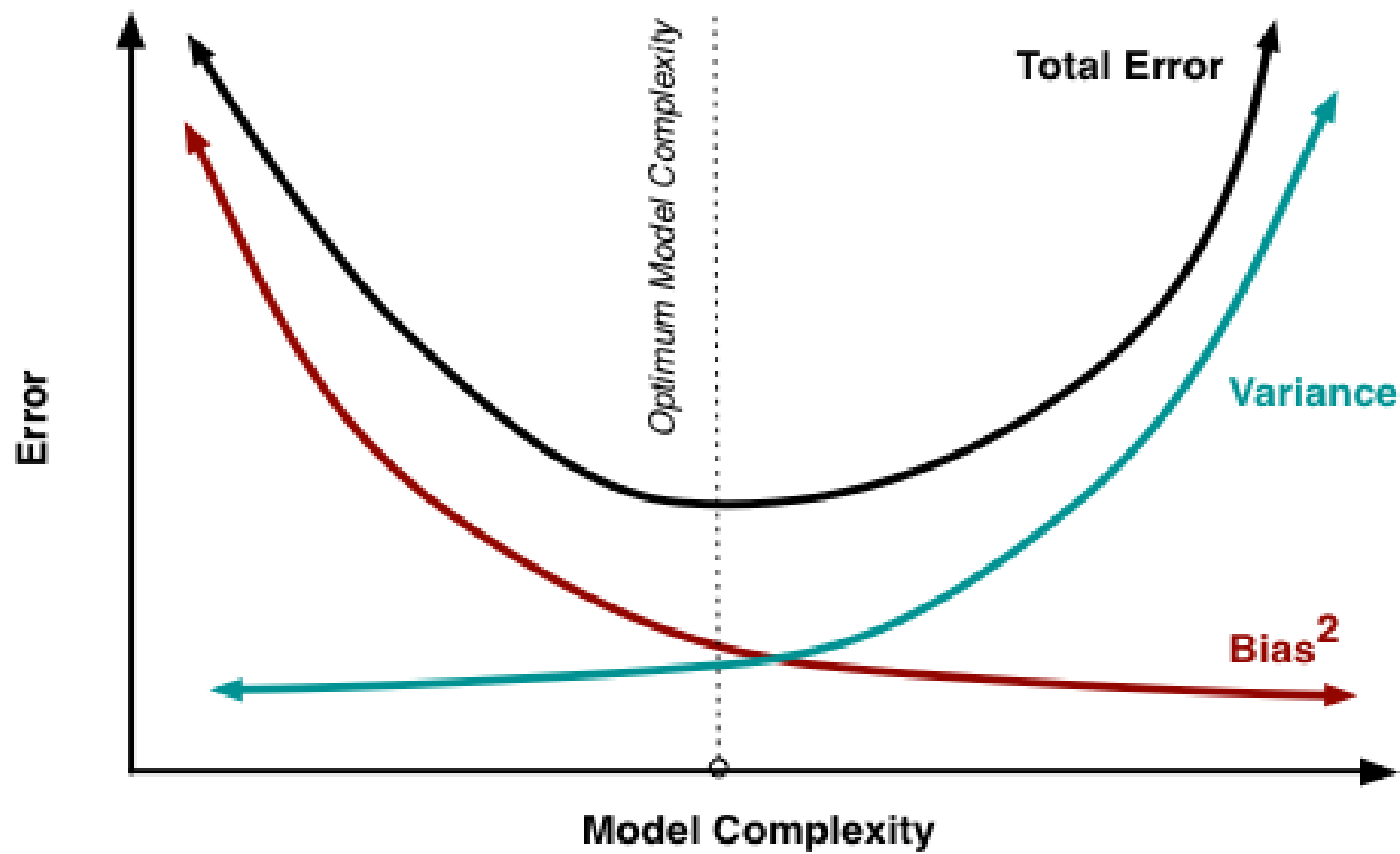


# Model Error: Bias versus Variance

- **Models that over-fit the data tend to have:**
  - Low bias ~ the model fits the training data very well
  - High variance – removing/changing a few training data points can change the model and hence the predictions a lot



# Tradeoff between Bias and Variance



# Tradeoff between Bias and Variance

- Reducing bias & variance is important for prediction accuracy
- Tradeoff:
  - bias vs. variance
  - high complexity models vs. low complexity models
  - most errors due to over-fitting vs. most errors due to under-fitting
  - choice: smart twitchy (sensitive) models vs. less smart but stable models
- Clearly we want smart models, but...
  - Can we reduce variance without increasing bias?
  - Can we reduce over-fitting without under-fitting?

**YES!**

# Reduce Variance Without Increasing Bias

- **Averaging reduces variance:**

$$Var(\bar{X}) = \frac{Var(X)}{N}$$

- **Average models to reduce model variance**
- **For large N, residual model error mainly due to bias!**
- **In practice:**
  - models are correlated, so reduction is smaller than 1/N
  - variance of models trained on fewer training cases is usually larger
  - only works with some learning methods: very stable learning methods have such low variance to begin with, that bagging does not help much.

# Can Bagging Hurt?

- **Each base classifier is trained on less data**
  - E.g. only about 67% of the data points are in any one bootstrap sample
- **If data is poor, then losing this much data can hurt accuracy**
- **Bagging usually helps, but sometimes not much...**

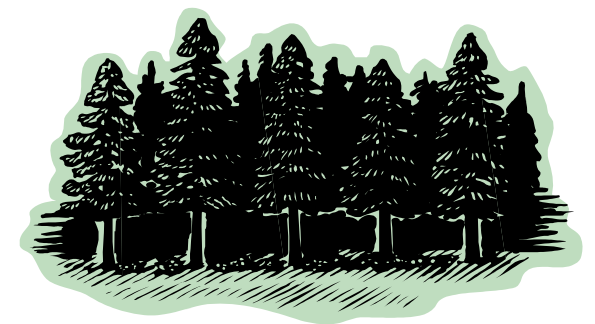
# Ways to create Model Diversity

- **Manipulating the training data (e.g. bagging)**
- **Manipulating the input features**
- **Varying the classifier type, architecture**



# Random Forests

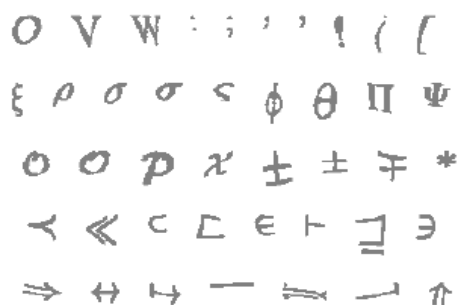
- Draw 1000+ bootstrap samples of data
- Draw random sample of available features at each tree split
  - Randomisation (hence model diversity) now occurs in two places:  
Random sampling of *training data* + Random sampling of *feature set*
  - Training speed increases due to less computation at each tree split (less features to evaluate the splitting cost function for)
- Train trees on each sample/attribute set -> 1000+ trees
- Use un-weighted voting to get final prediction (as with bagging)



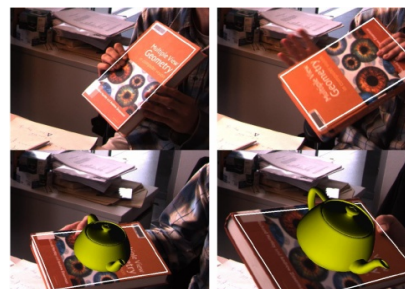
# Random Forests

- **Usually works better than bagging**
  - robust to noise, easy to use, surprisingly high accuracy
  - but.. lots of trees means hard to interpret (becomes a black box)
- **Variance of RF trees is higher than Bagged Trees**
  - typically needs 10X as many trees
  - trees should be (generally) unpruned (to encourage diversity)
  - RF needs 100's to 1000's
- **Extra parameter to tune:  $p(\text{feat})$** 
  - probability of getting to use feature at each split
  - fortunately, usually not too sensitive
  - Breiman suggests  $\text{SQRT}(N)$ ,  $N$ : total number of features
- **Unlike Bagging and Boosting, RF is for decision trees only**

# Random Forests in Vision



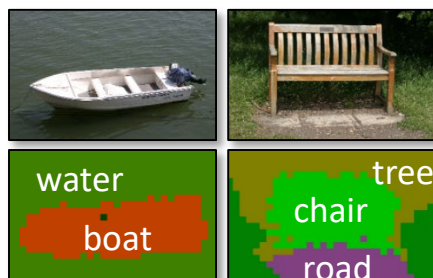
[Amit & Geman, 97]  
**digit recognition**



[Lepetit *et al.*, 06]  
**keypoint recognition**



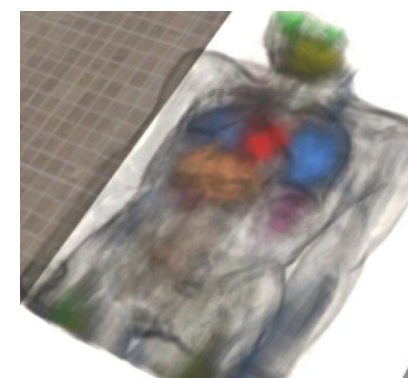
[Moosmann *et al.*, 06]  
**visual word clustering**



[Shotton *et al.*, 08]  
**object segmentation**



[Rogez *et al.*, 08]  
**pose estimation**



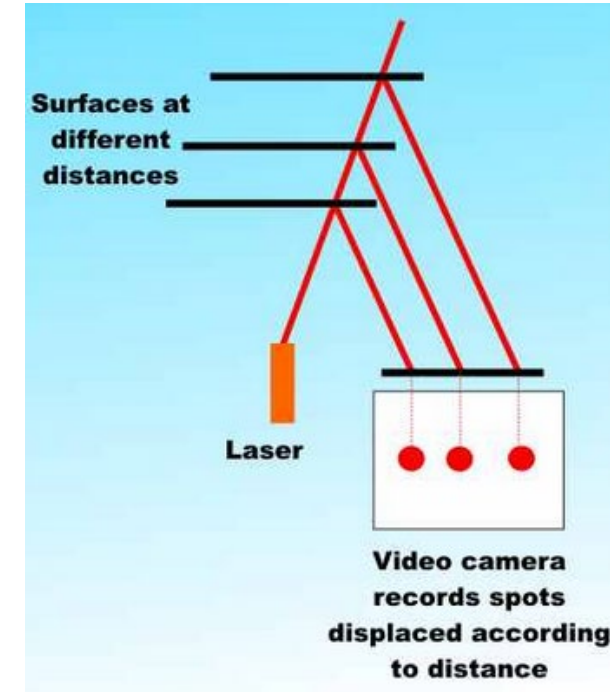
[Criminisi *et al.*, 09]  
**organ detection**

(Among many others...)

# Kinect's Decision Forest

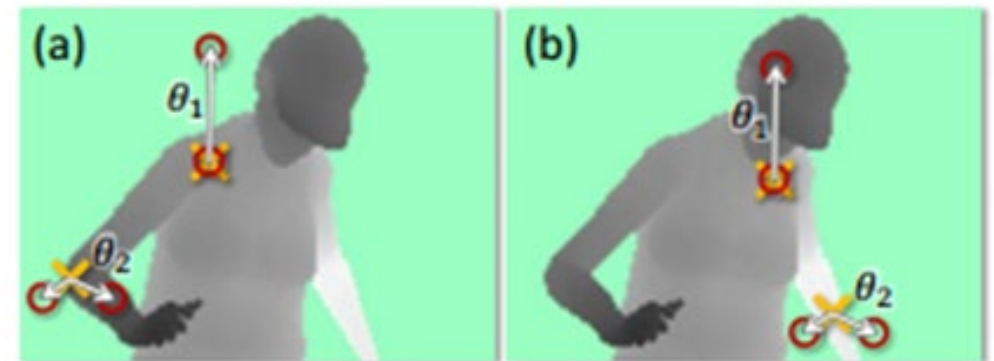
- **Step1: Generate a 3D image**

- Kinect uses "structured light" ~ If you have a light source offset from a detector by a small distance then the projected spot of light is shifted according to the distance it is reflected back from.



- **Step2: Compute Features**

- Compute the difference in depth (z) to two pixels that are close together in (x,y). If difference is small then they likely belong to same object. Repeat many times.



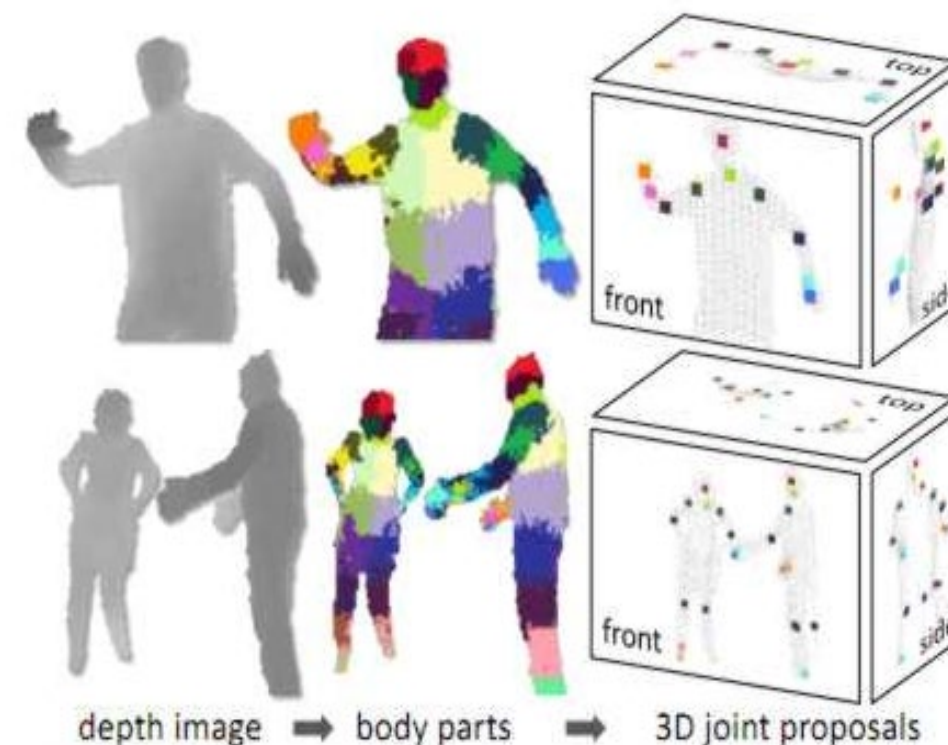
# Kinect's Decision Forest

- **Step3: Build Forest**

- Each tree was trained on features that were pre-labeled with the target body parts.
- Training just 3 trees using 1 million test images took a day using a 1000 core cluster.
- The trained classifiers assign a probability of a pixel being in each body part

- **Step4: Execute the Forest**

- Picks areas of maximum probability for each body part type.



<http://www.youtube.com/watch?v=HNkbG3KsY84>

# Testing Random Forests

- No need for separate test set

- Method:

- Test each tree against the data left over after the bootstrap sample was taken; this is called the OOB (out-of-bag) data

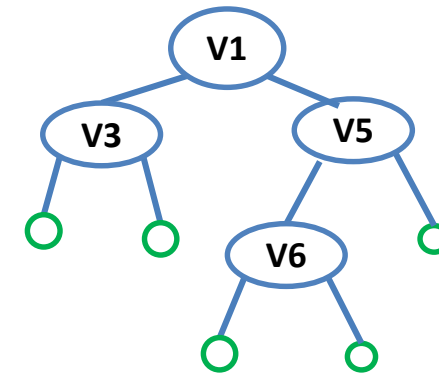
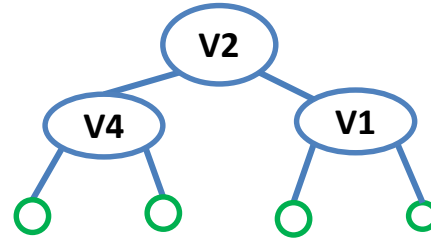
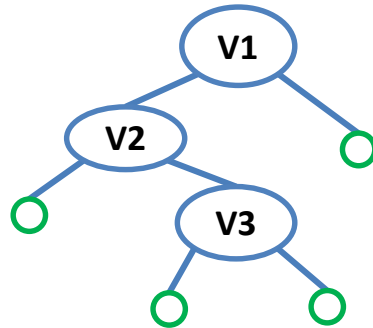


- If each bootstrap sample takes 67% of the training data, then after building  $T$  trees every training example will have been OOB (and hence a valid test example) for about  $T/3$  times.
- For each training example, take the majority vote of all  $T/3$  test predictions to get the forest's prediction\* and compare with the actual class value. Output 1 if forest prediction  $\neq$  actual, else output 0
- Sum over all training examples to get error estimate for the forest

\* For regression problems, average all  $T/3$  test predictions to get the forest prediction.  
Then compute MSE as  $\sum_{\text{training examples}} (\text{prediction} - \text{actual})^2$

# Measuring Variable Importance

- For a single tree the order in which the variables occur in the tree is a measure of their relative importance to the prediction



- For a forest?
  - Naïve method: Count the number of times the variable occurs in all of the trees, more occurrences => more important
  - Better method: sum the total reduction in impurity (eg. the decreases in the Gini index) for all nodes that test the variable



# Measuring Variable Importance

- **Permutation Method**

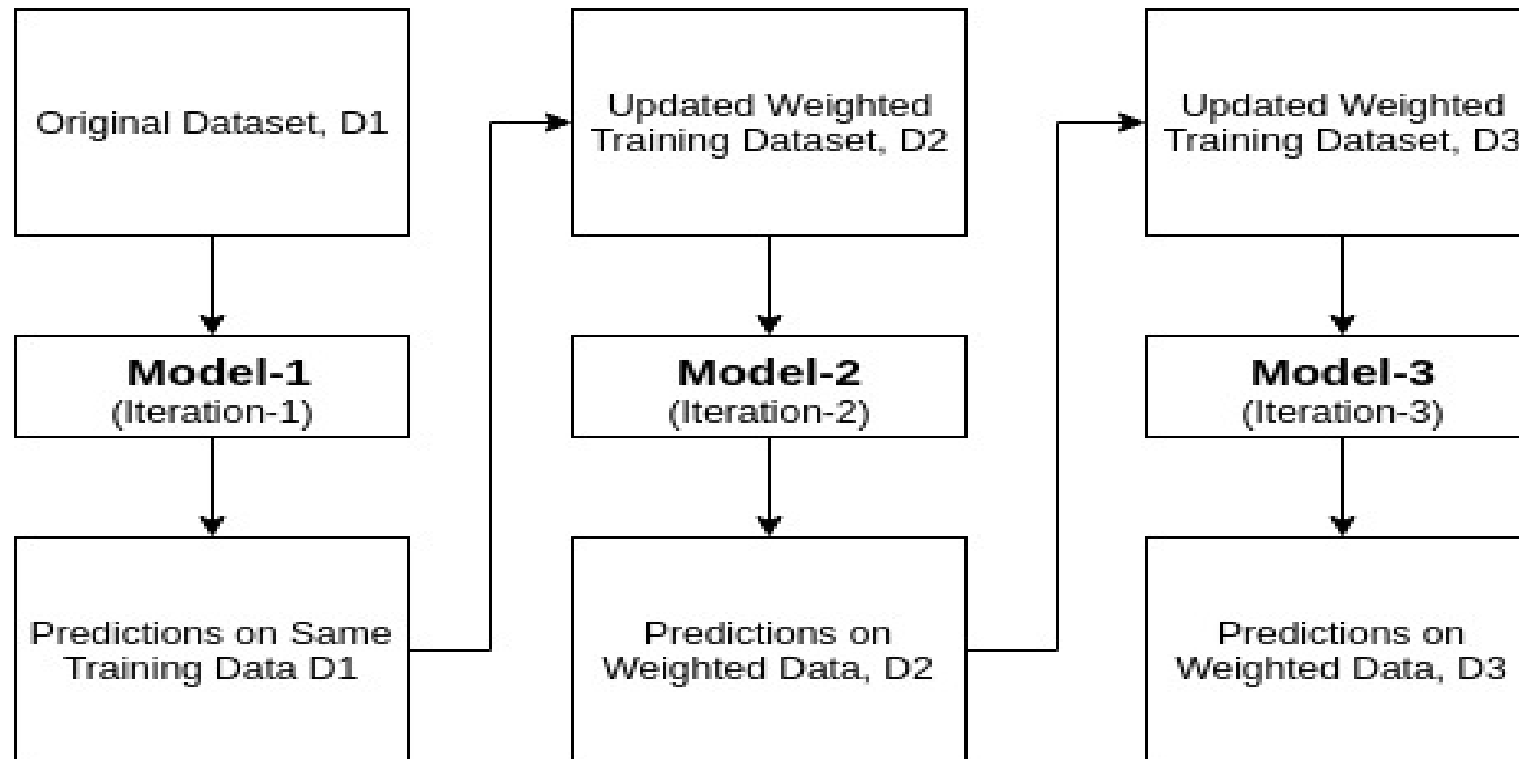
- Randomly shuffle the values of a given input variable to “break” the bond of the variable to the response. Then the difference of the model accuracy before and after the shuffling is a measure of how important the variable is for predicting the response
- Detailed Steps
  1. Test every tree on its own OOB examples. For each training example  $\mathbf{e}$  count the votes for the correct class (call this  $\text{NormalCorrectVotes}_e$ )
  2. For each input variable  $\mathbf{v}$ :
    - For each tree  $\mathbf{t}$ :
      - Randomly permute the values for  $\mathbf{v}$  in the OOB examples and retest the tree
    - For each training example, count the votes for the correct class
    - $\text{Importance}_v = \text{average} \left[ \frac{\text{NormalCorrectVotes}_e - \text{ShuffledCorrectVotes}_e}{\text{TotalVotes}_e} \right]$



- **Can a set of weak learners create a single strong learner?**
  - A weakly learned model is only slightly better than random guessing
  - A strongly learned model is arbitrarily well-correlated with the truth
- **Boosting essentials:**
  - Build a model (but don't 100% over-fit the data!)
  - Increase weights of the training examples the model gets wrong
  - Retrain a new model using the weighted training set
  - Repeat many times...



# Boosting



[www.datacamp.com](http://www.datacamp.com)

# Basic Boosting Algorithm

1. Weight all training samples equally
2. Train model on train set
3. Compute error of model on train set
4. *Increase weights on train cases that the model gets wrong!*
5. Train new model on re-weighted train set
6. Re-compute errors on weighted train set
7. Increase weights more on cases it still gets wrong
8. Repeat until tired (100+ iterations)
9. Final model: *weighted* prediction of each model (aka base models)

# AdaBoost\* (Adaptive Boosting)

Given:  $(x_1, y_1), \dots, (x_m, y_m)$  where  $x_i \in \mathcal{X}$ ,  $y_i \in \{-1, +1\}$ . ← The training examples

Initialize:  $D_1(i) = 1/m$  for  $i = 1, \dots, m$ . ← The training example weights

For  $t = 1, \dots, T$ :

- Train weak learner using distribution  $D_t$ . ← Train a model (build a classifier)
- Get weak hypothesis  $h_t : \mathcal{X} \rightarrow \{-1, +1\}$ .
- Aim: select  $h_t$  with low weighted error:

$$\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].$$

← Goal of classifier is to reduce weighted error relative to  $D_t$

- Choose  $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$ .
- Update, for  $i = 1, \dots, m$ :

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

← Cases where the prediction does not equal the real class value

← Re-weight the examples

where  $Z_t$  is a normalization factor (chosen so that  $D_{t+1}$  will be a distribution).

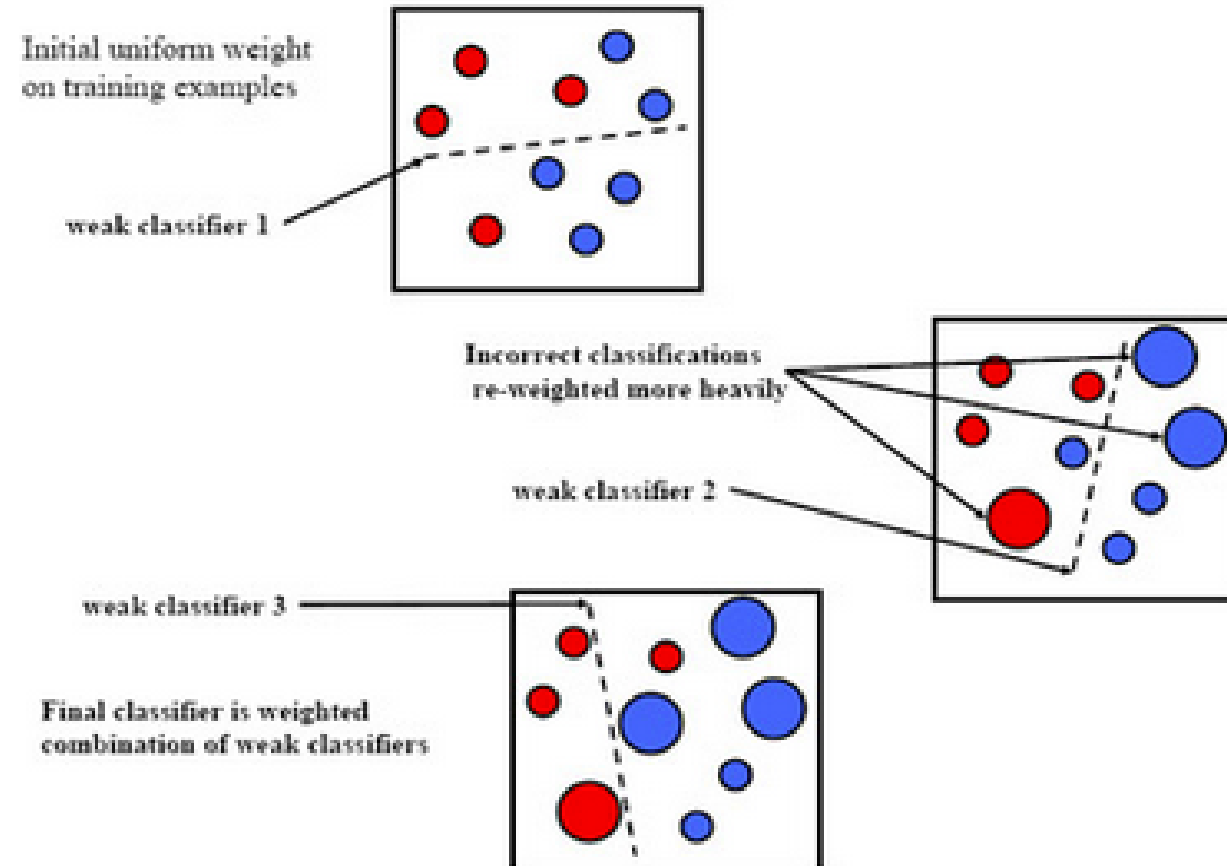
Output the final hypothesis:

$$H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right).$$

← The “final” prediction is the weighted average of all of the weak classifiers

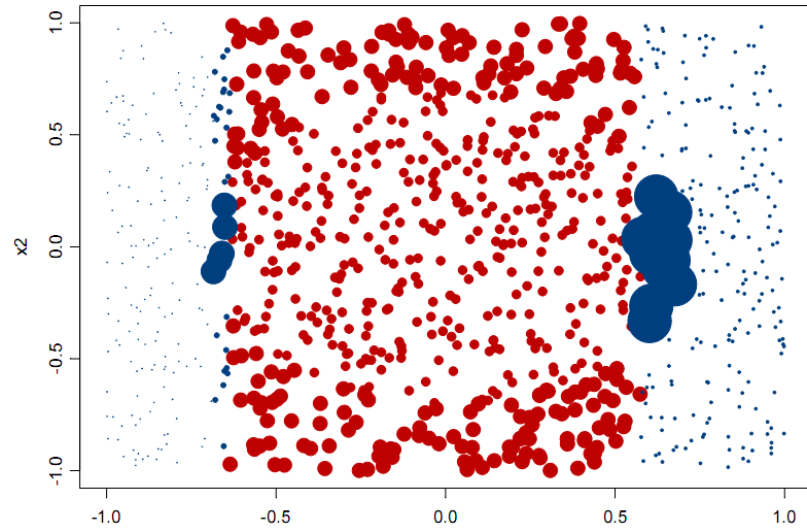
\*Freund and Schapire, 94

# AdaBoost - Conceptual



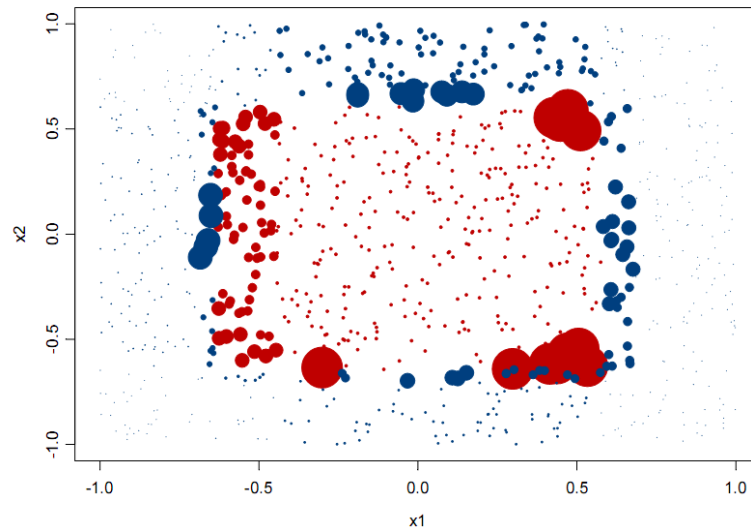
$$H(x) = \text{sign}(\alpha_1 h_1(x) + \alpha_2 h_2(x) + \alpha_3 h_3(x))$$

# AdaBoost

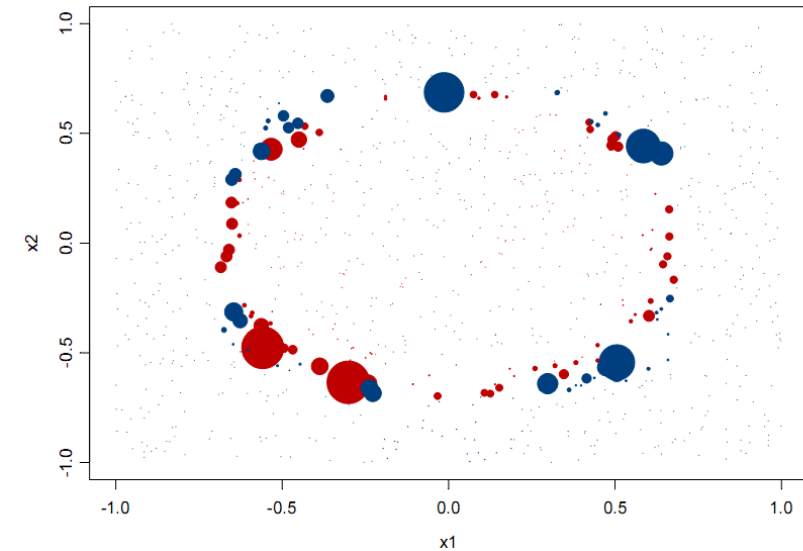


Classifications (colors) and  
Weights (size) after *1 iteration*  
of AdaBoost

*3 iterations*



*20 iterations*

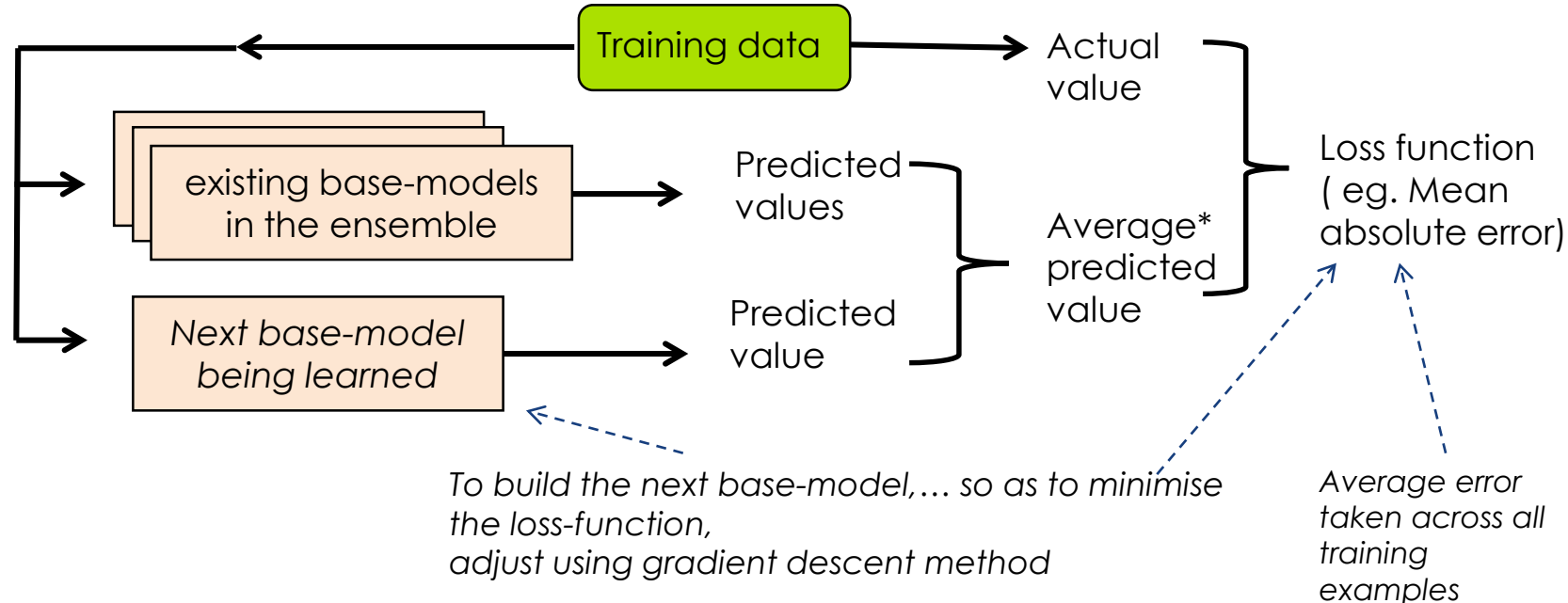


Source: Elder, John. From Trees to Forests and Rule Sets - A Unified Overview of Ensemble Methods. 2007.

- **How might we use the adjusted weights in algorithms?**
  - Neural Networks – Scale learning rate by weight
  - Decision Trees – instance membership is scaled by weight
  - $k$ -NN – node vote is scaled by weight

# Gradient Boosting

- Treat learning as Gradient Descent optimisation
- Like Adaboost, models are learned and added to the ensemble sequentially





# Gradient Boosting

Input: training set  $\{(x_i, y_i)\}_{i=1}^n$ , a differentiable loss function  $L(y, F(x))$ , number of iterations  $M$ .

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma).$$

2. For  $m = 1$  to  $M$ :

1. Compute so-called *pseudo-residuals*:

$$r_{im} = - \left[ \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \dots, n.$$

2. Fit a base learner (e.g. tree)  $h_m(x)$  to pseudo-residuals, i.e. train it using the training set  $\{(x_i, r_{im})\}_{i=1}^n$ .

3. Compute multiplier  $\gamma_m$  by solving the following **one-dimensional optimization** problem:

$$\gamma_m = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)).$$

4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output  $F_M(x)$ .

Note: Ever since its introduction in 2014, XGBoost has become a widely used and popular tool which implements the gradient boosting algorithm.

# Boosting versus Bagging/RF

- **In practice bagging/RF almost always helps**
- **Bagging doesn't work as well with stable models**
  - Boosting and RF might still help.
- **Often, boosting helps more than bagging**
  - Boosting might hurt performance on noisy datasets
  - Bagging/RF don't have this problem.
- **Bagging/RF is easier to parallelize**

- **Ensembles**

- Using multiple models to reduce variance and increase accuracy
- Works best if models don't agree with each other (need model variance)
- Usually refers to multiple models of same type
- Bagging & Boosting are the most popular generic methods
- Random Forest increasingly popular

- **Ensemble Creation Approaches**

- A good goal is to get less correlated errors between models
- Injecting randomness – initial weights, different learning parameters, etc.
- Different Training sets – Bagging, different features, etc.
- Forcing differences – different objective functions, auxiliary tasks

- **Ensemble Combining Approaches**

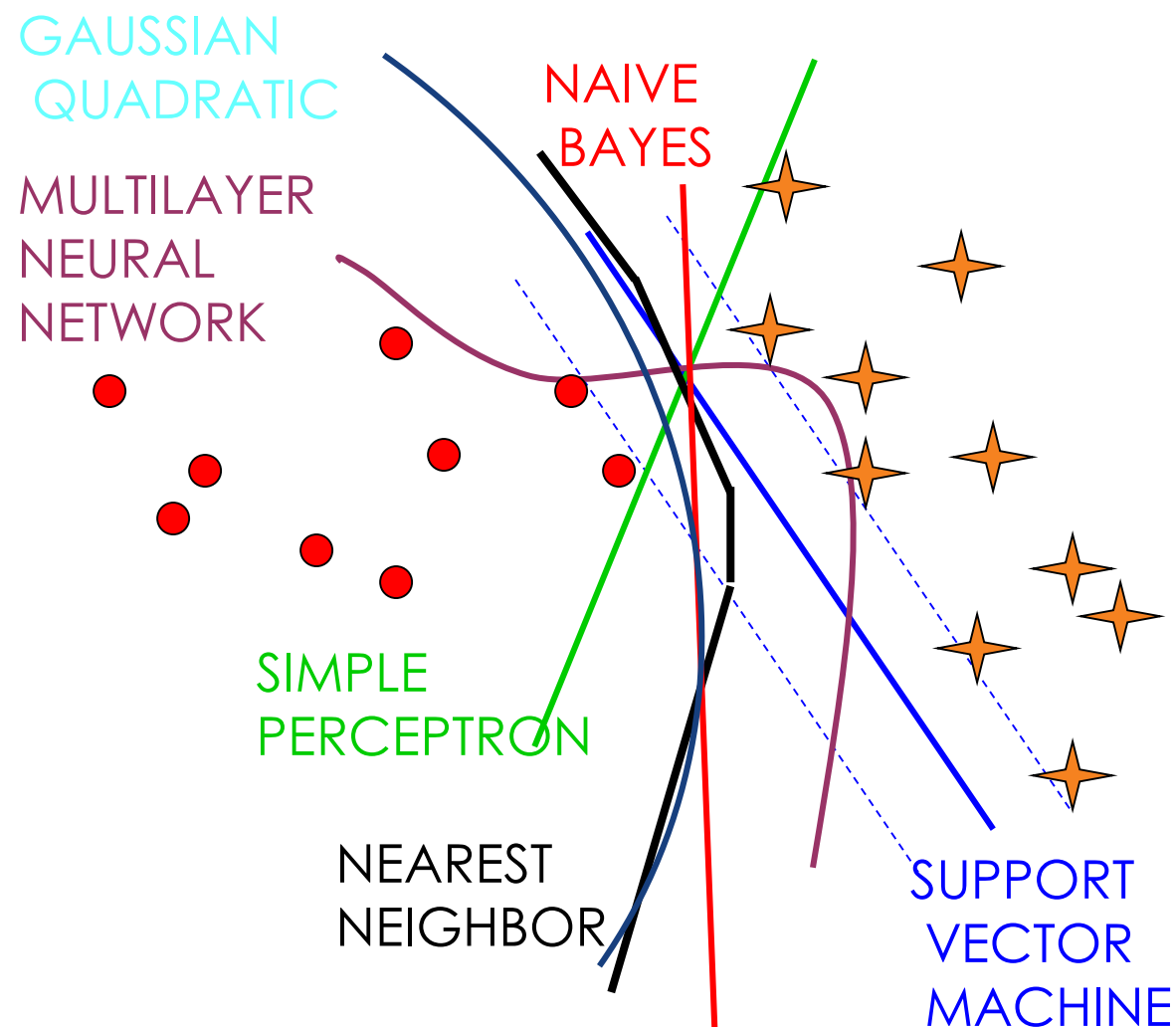
- Unweighted Voting (e.g. Bagging)
- Weighted voting – based on accuracy, heuristics, etc. (e.g. Boosting)
- Stacking - Learn the combination function

# Hybrid Machine Learning Models

- Both ensemble models and hybrid models make use of the information fusion concept but in slightly different way.
- Ensemble Models - multiple but homogeneous, weak models are combined
- Hybrid Models combine different, heterogeneous machine learning approaches.
- They both may considerably improve quality of reasoning and boost adaptivity of the entire solutions.
- They both have found applications in numerous real world problems ranging from person recognition, medical diagnosis, bioinformatics, recommender systems and text/music classification to financial forecasting.

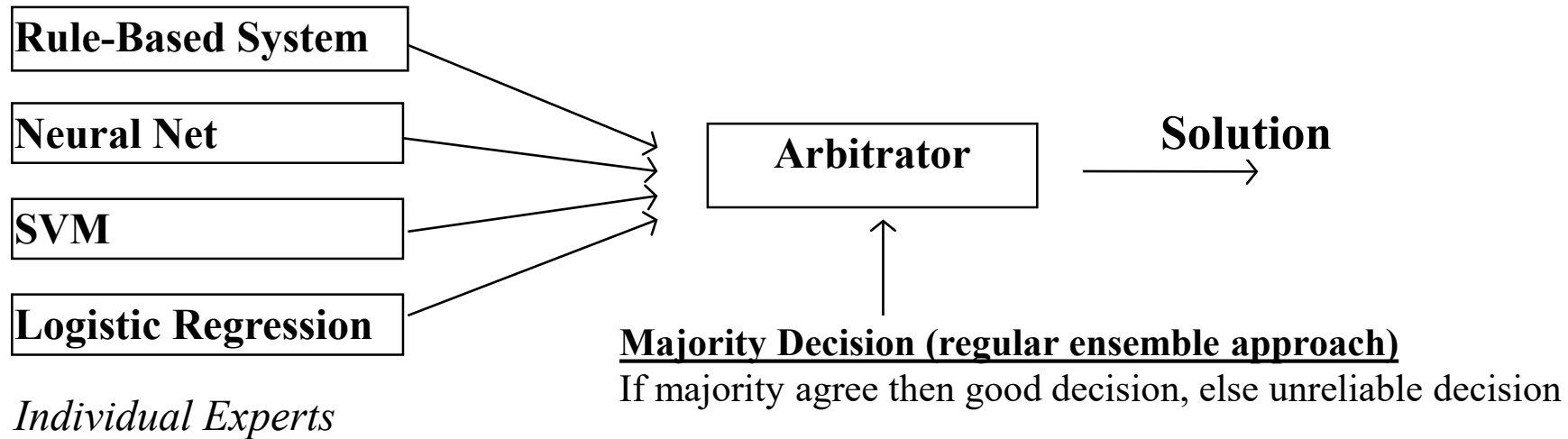
- In “Multiple Classifier Systems” approach, each classifier is an expert in certain situations
  - Each model type has different strengths and weaknesses
  - Usually have relatively small number of experts
  - Allows for more model combination methods apart from averaging

# Multiple Classifier Systems Example



# Multiple Classifier Systems Example

- Different solution strategies (experts) offer alternative solutions. Another process decides which solution to accept or how to combine the solutions, e.g. majority vote algorithm.
- This architecture is also known as stacking\*



OR...

**Weighted Decision**

Weight expert judgements according to circumstances

**Best Expert Only**

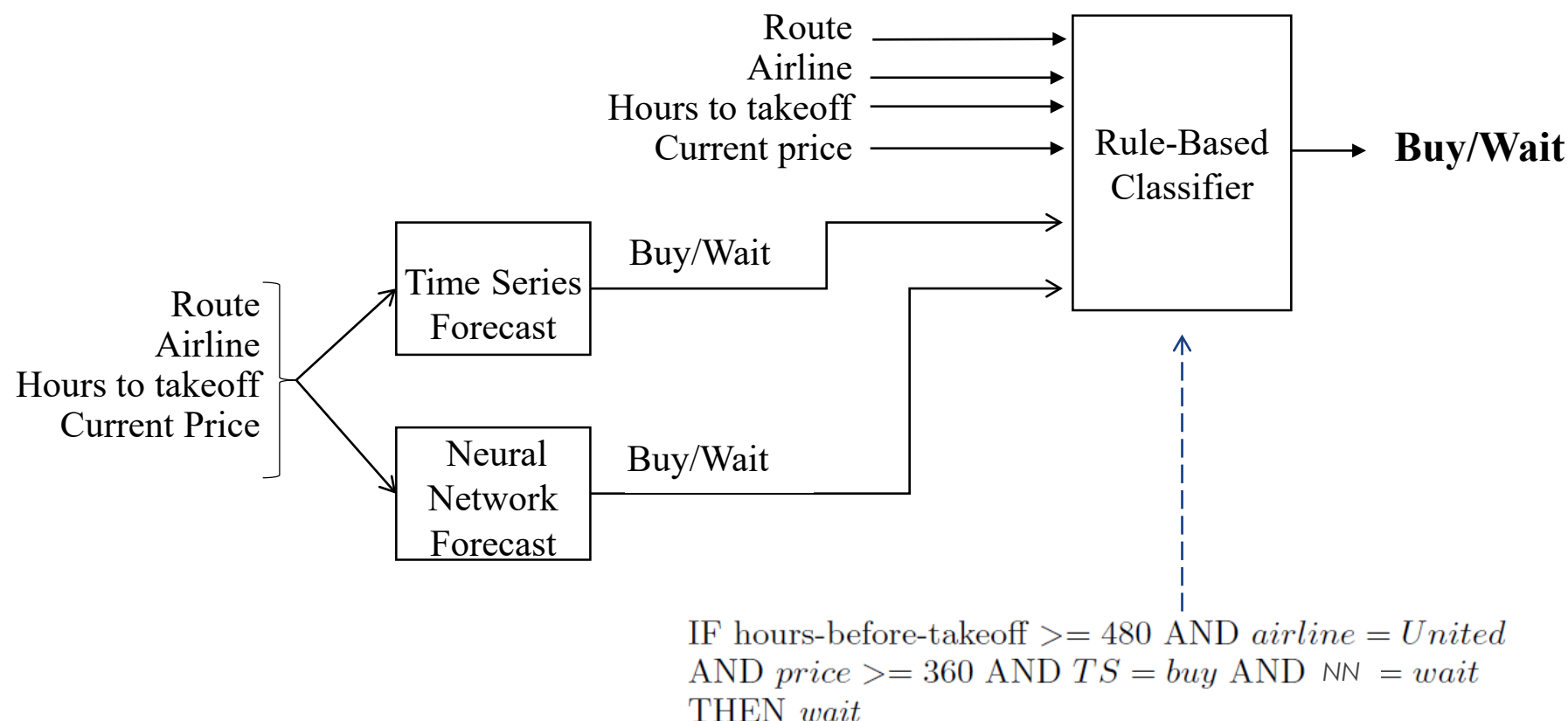
Decide which expert is most appropriate for current situation

\*Stacking (sometimes called stacked generalization) involves training a learning algorithm to combine the predictions of several other learning algorithms (Wikipedia)



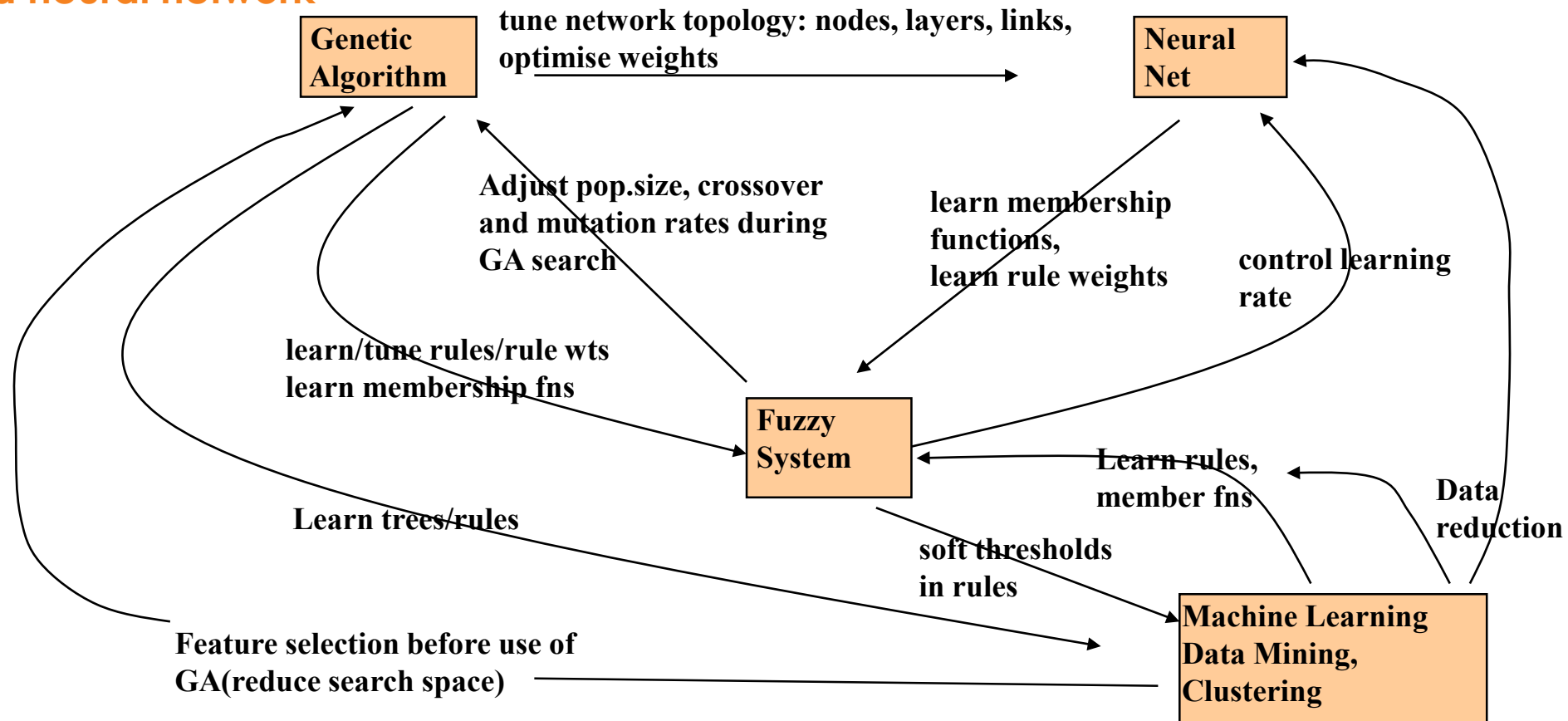
# Multiple Classifier Systems Example

- Airfare Price prediction
- The Experts have same skills (Buy/Wait decision) but sometimes one is better than the other! The arbitrator helps decide which to use and when



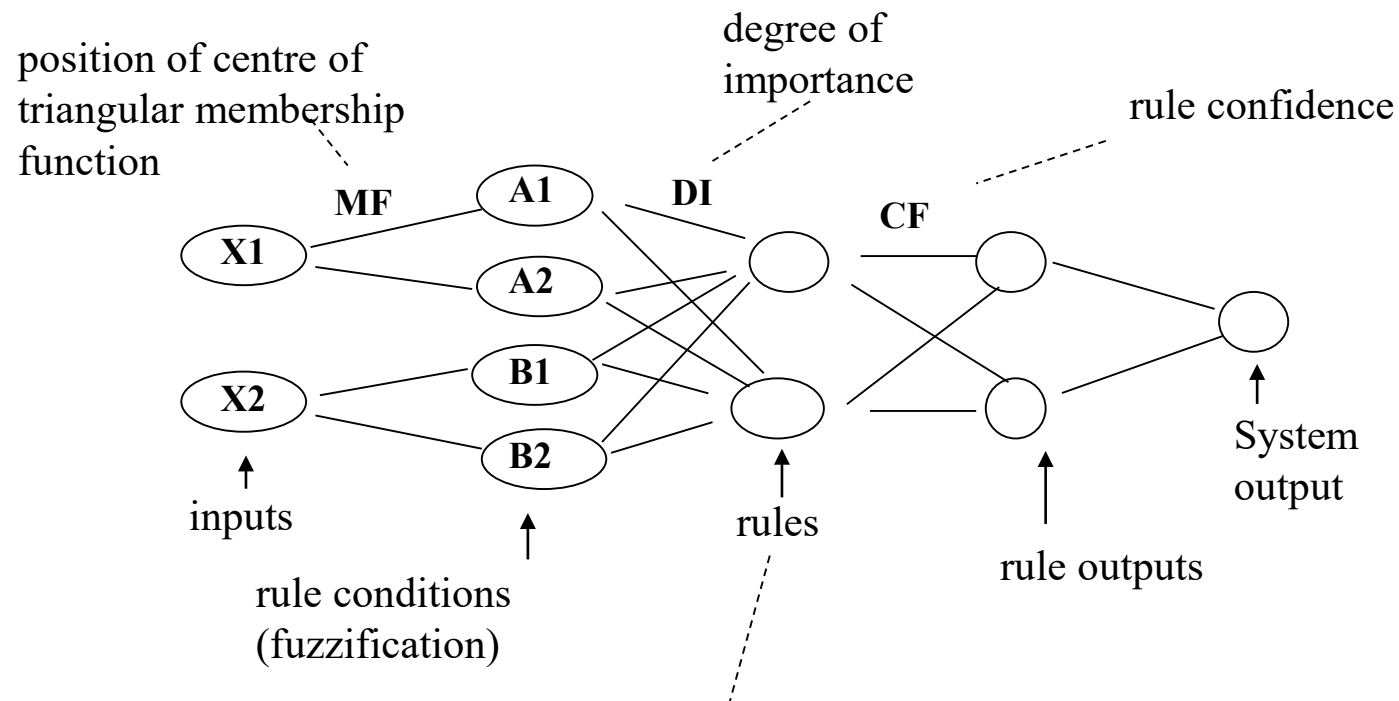
# Self-Tuning Systems

- One technique is used to tune or learn the architecture for another,  
e.g: Neural network is used to learn a Fuzzy System , Genetic algorithm is used to optimise a neural network



# Self-Tuning Systems Example: Neuro-Fuzzy Systems

- **Neural Network is used to represent and “learn” a Fuzzy System**
- **Nodes represent rule inputs, conditions, actions etc**
- **Special training algorithm required**



*Taking only the strongest connection to each condition element yields rules like (other schemes exist):-*

e.g. If X1 is A1 (DI1) And X2 is B1 (DI2) Then output = C ( CF1)

# Self-Tuning Systems Example: GA-ML

- **Example: learning rules to predict type of object**

- Let F1 and F2 be the input variables with  
F1 taking values {small, medium large}  
F2 taking values {sphere, cube, brick, tube}  
Let Class take values {widgets, gadgets}

- Then the chromosome

F1	F2	Class
110	0001	0

Represents the rule

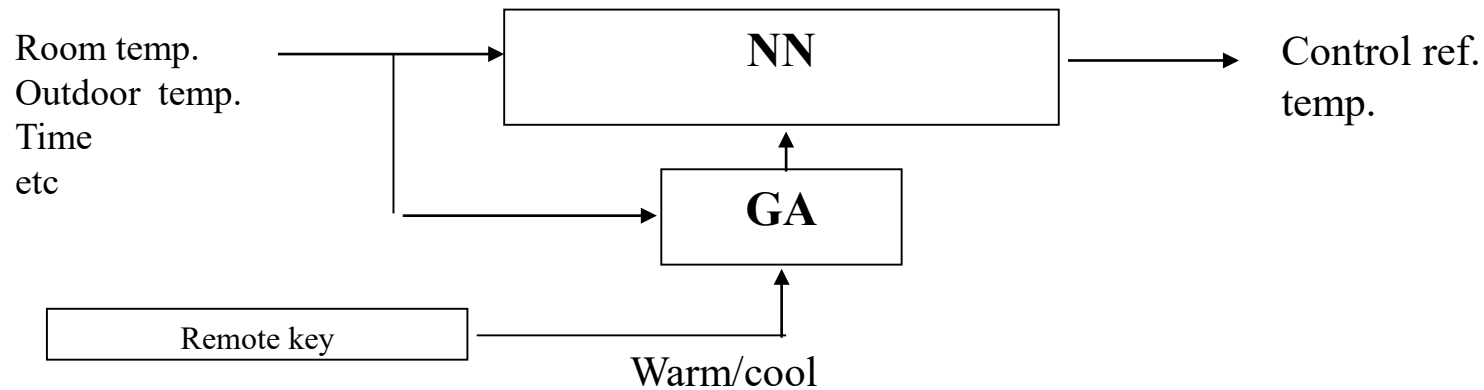
**If F1 = small or medium and F2 = tube then widget**

# Self-Tuning Systems Example: GA-Neural

- **NNs are generated/tuned by GAs**
- **GA chromosome represents NN topology**
  - number of hidden layers, hidden nodes and number of links and/or weights
- **Pros: GA can avoid local minima more than back-prop**
- **Cons: size of chromosome gets prohibitively large if topology is being learned**

# Self-Tuning Systems Example: GA-Neural

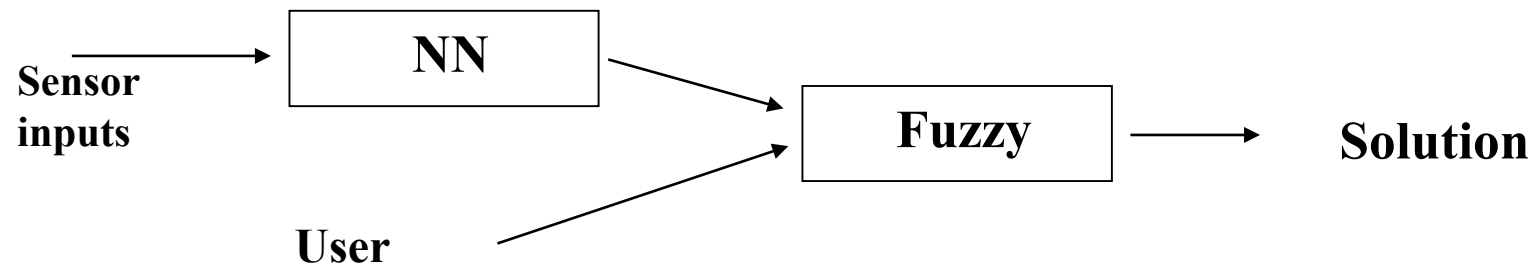
- LG Electric developed an air-con controlled by an NN
- If the user wants the air-con to adapt to their preferences then a GA is used to change the number of neurons and weights



# Co-operating Experts

## “Pooled Expertise”

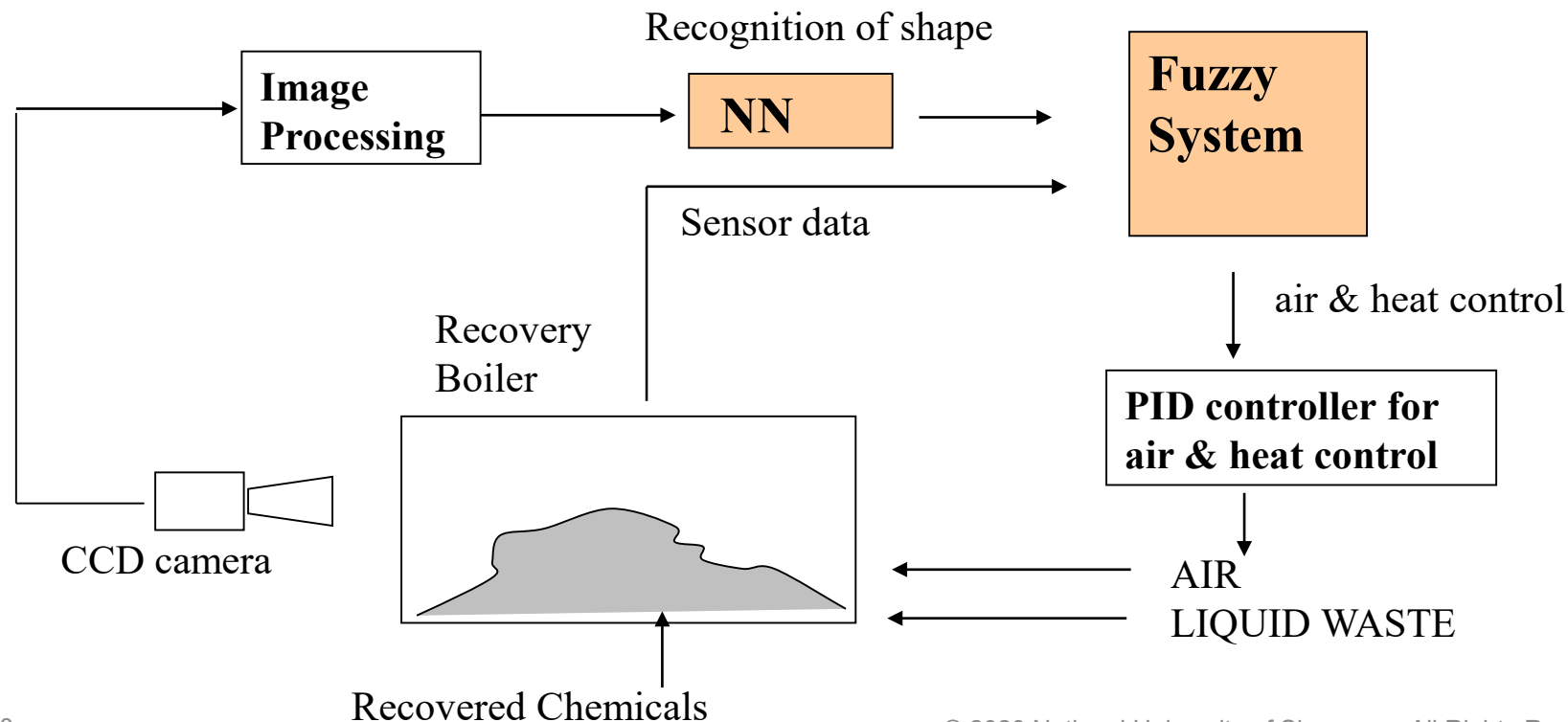
- Different techniques work together as a team to produce a single solution, no single technique/expert is sufficient alone
- E.g, NN provides input to Fuzzy System



# Co-operating Experts Example

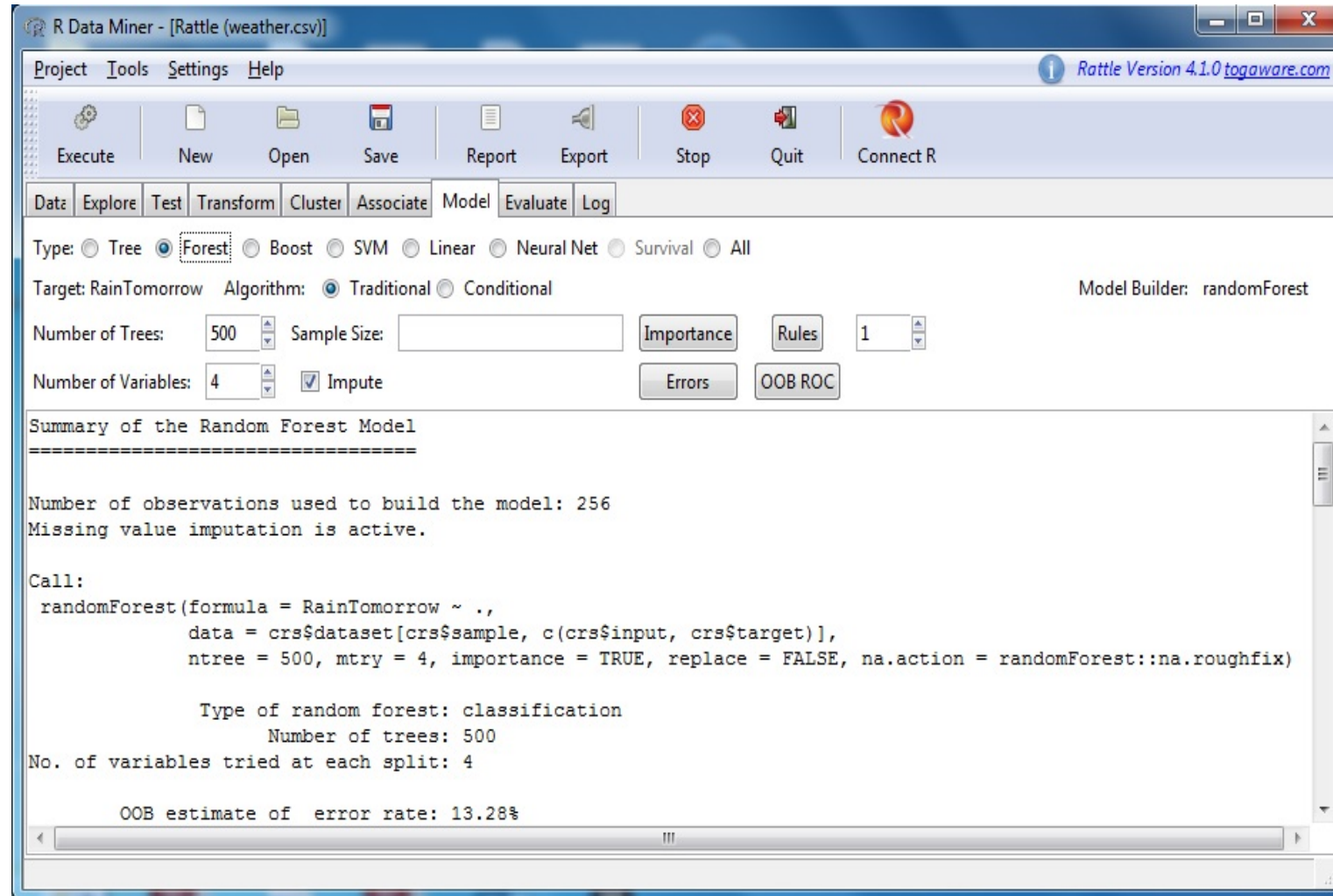
- **Recovery of expensive chemicals at a pulp factory**

- Fuzzy system controls the temp. of liquid waste and air before input to recovery boiler
- Shape of pile in boiler influences the efficiency of the recovery process (deoxidisation). NN recognises the shape of the pile from (edge) image and passes to the Fuzzy system.

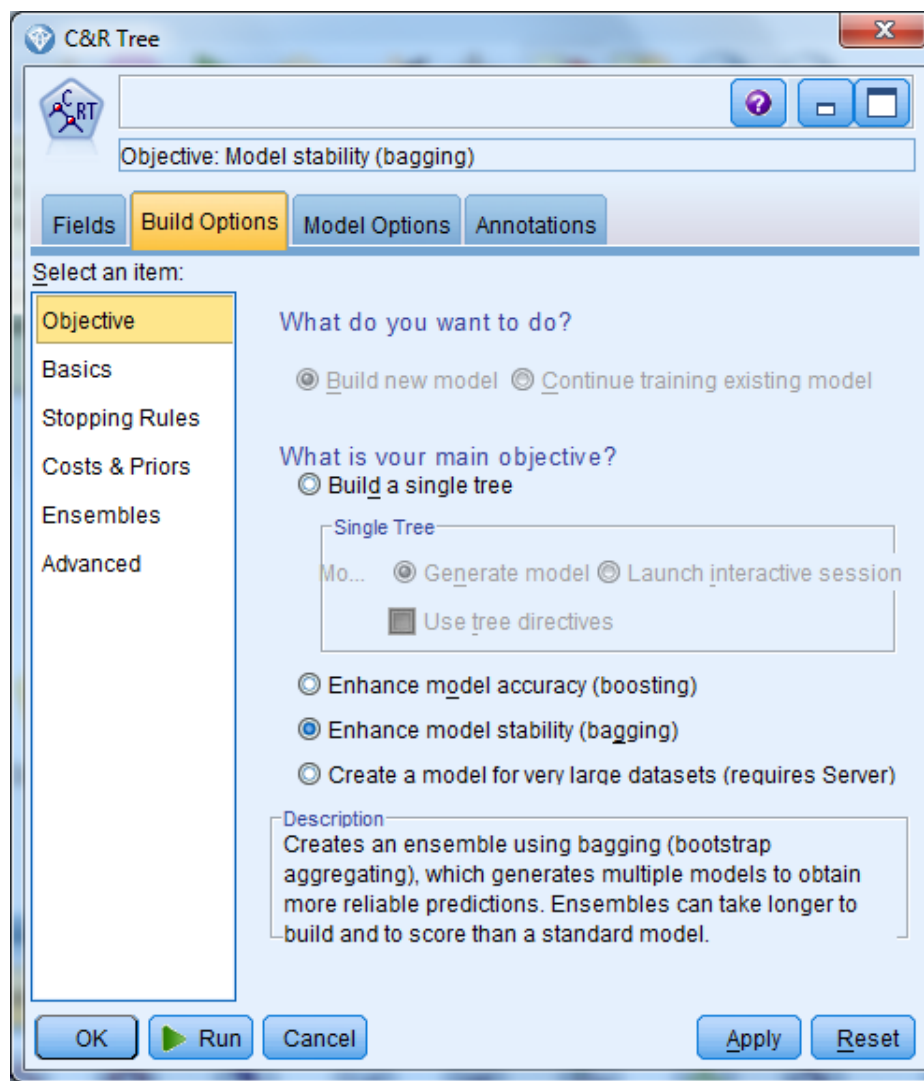




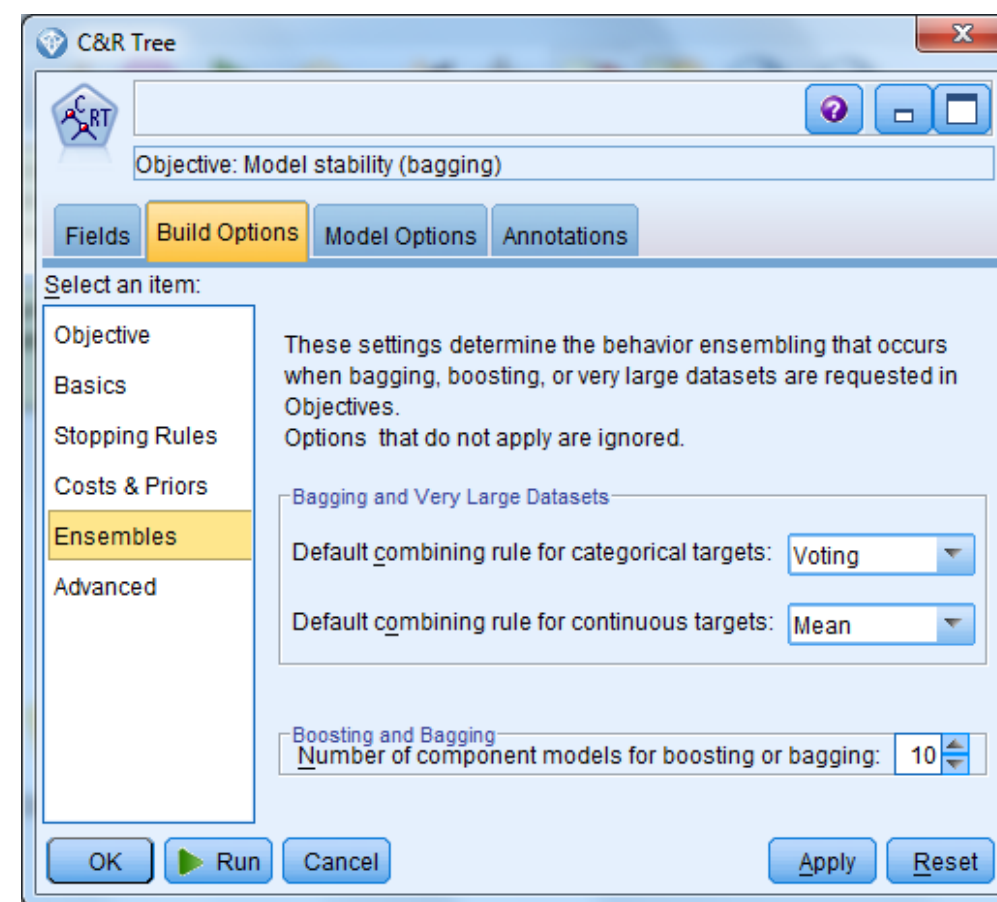
# Bagging & Boosting in R & Rattle



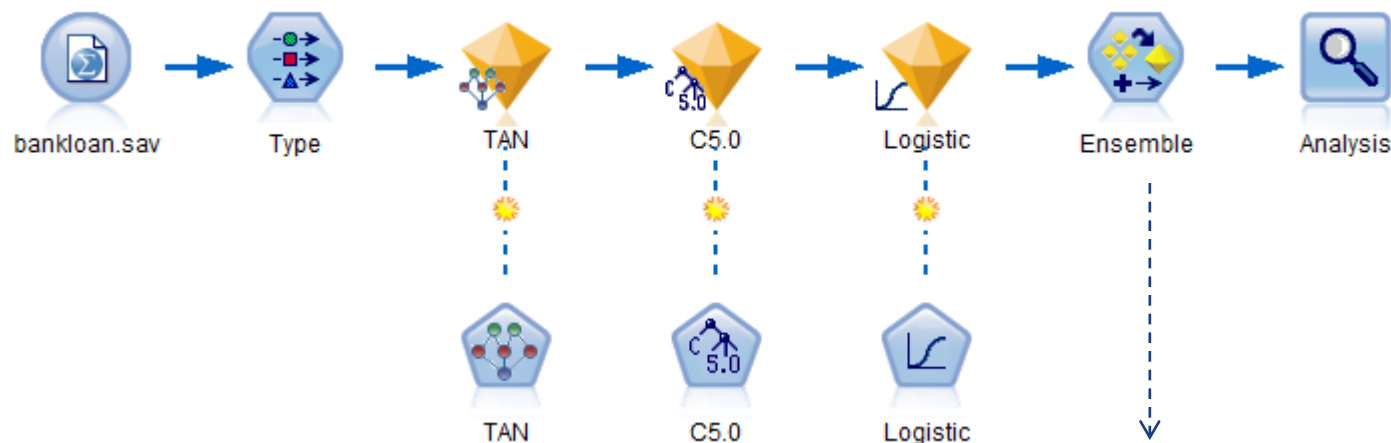
# Bagging & Boosting in SPSS Modeler



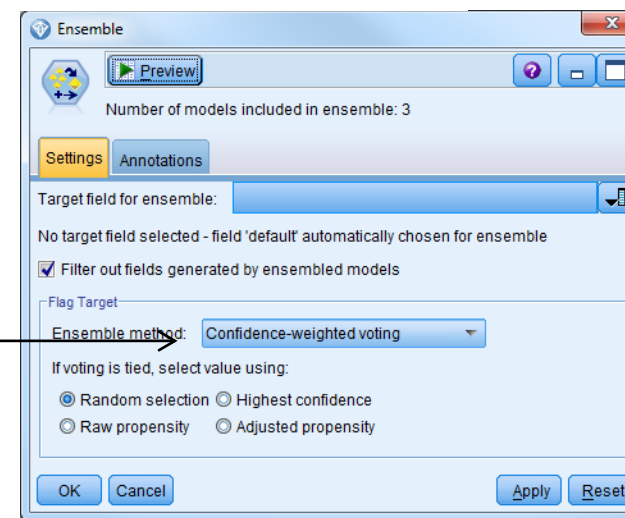
- Some model nodes implement bagging & boosting



# Building Ensembles in SPSS Modeler



Ensemble method	Field name
Voting	
Confidence-weighted voting	
Raw-propensity-weighted voting	\$XFC_<field>
Raw-propensity-weighted voting	
Highest confidence wins	
Average raw propensity	\$XFRP_<field>
Average adjusted raw propensity	\$XFAP_<field>



# Random Forest using Scikit-Learn

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=200, random_state=0)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)


from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

print(confusion_matrix(y_test,y_pred))

print(classification_report(y_test,y_pred))

print(accuracy_score(y_test, y_pred))
```

# AdaBoosting Using Scikit-Learn

```
from sklearn.ensemble import AdaBoostClassifier

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn import metrics


# Load data

iris = datasets.load_iris()

X = iris.data

y = iris.target


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

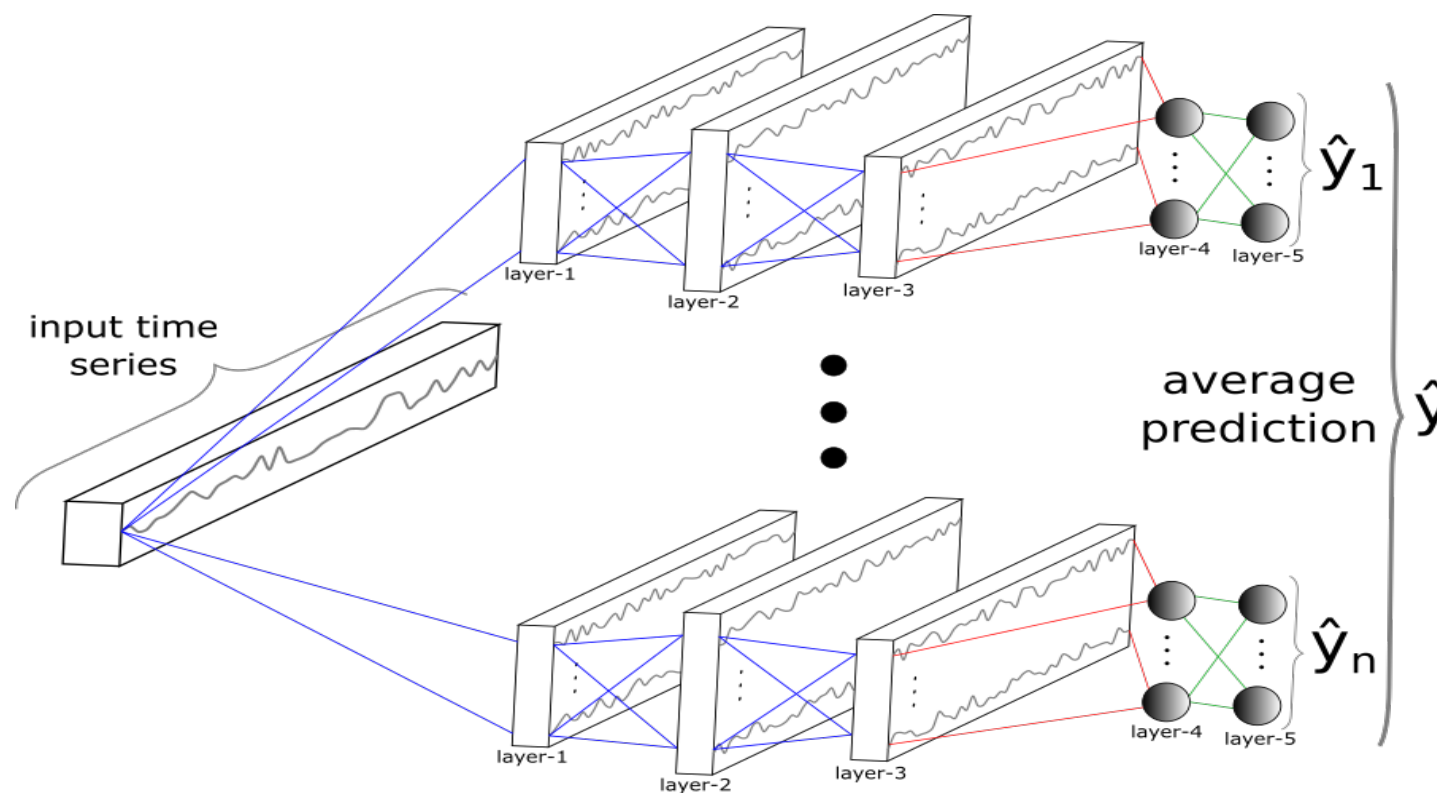

abc = AdaBoostClassifier(n_estimators=50, learning_rate=1)

model = abc.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

# Application Examples

- **Deep Neural Network Ensembles for Time Series Classification**



<https://github.com/hfawaz/ijcnn19ensemble>

# Application Examples

- An Ensembled Neural Network Classifier for Vehicle Classification
- Artificial Neural Network Ensembles and Their Application in Pooled Flood Frequency Analysis
- Ensembling ConvNets using Keras  
<https://towardsdatascience.com/ensembling-convnets-using-keras-237d429157eb>

# 5.2

## Ensemble Workshop



# Random Forest and Boosting

- Open the iPython notebooks provided for Random Forest and Boosting.
- As you run through the notebooks, make sure you understand how each **ensemble** method is implemented. (you can save notes as markdown in the notebook).
- Compare the performance of these models.
- Experiment with different parameter settings.

# NN Ensembles – Averaging and Stacking

- Open the iPython notebooks provided for the two NN ensembles: AverageNNEnsemble and StackingNNEnsemble.
- As you run through the notebooks, make sure you understand how each **ensemble** method is implemented. (you can save notes as markdown in the notebook).
- Compare the performance of these models.
- Experiment with different parameter settings.
- Save your notebook with the cell output and upload it to LumiNUS.