

# ML classifiers: ensemble methods

Christian Salvatore  
Scuola Universitaria Superiore IUSS Pavia

[christian.salvatore@iusspavia.it](mailto:christian.salvatore@iusspavia.it)

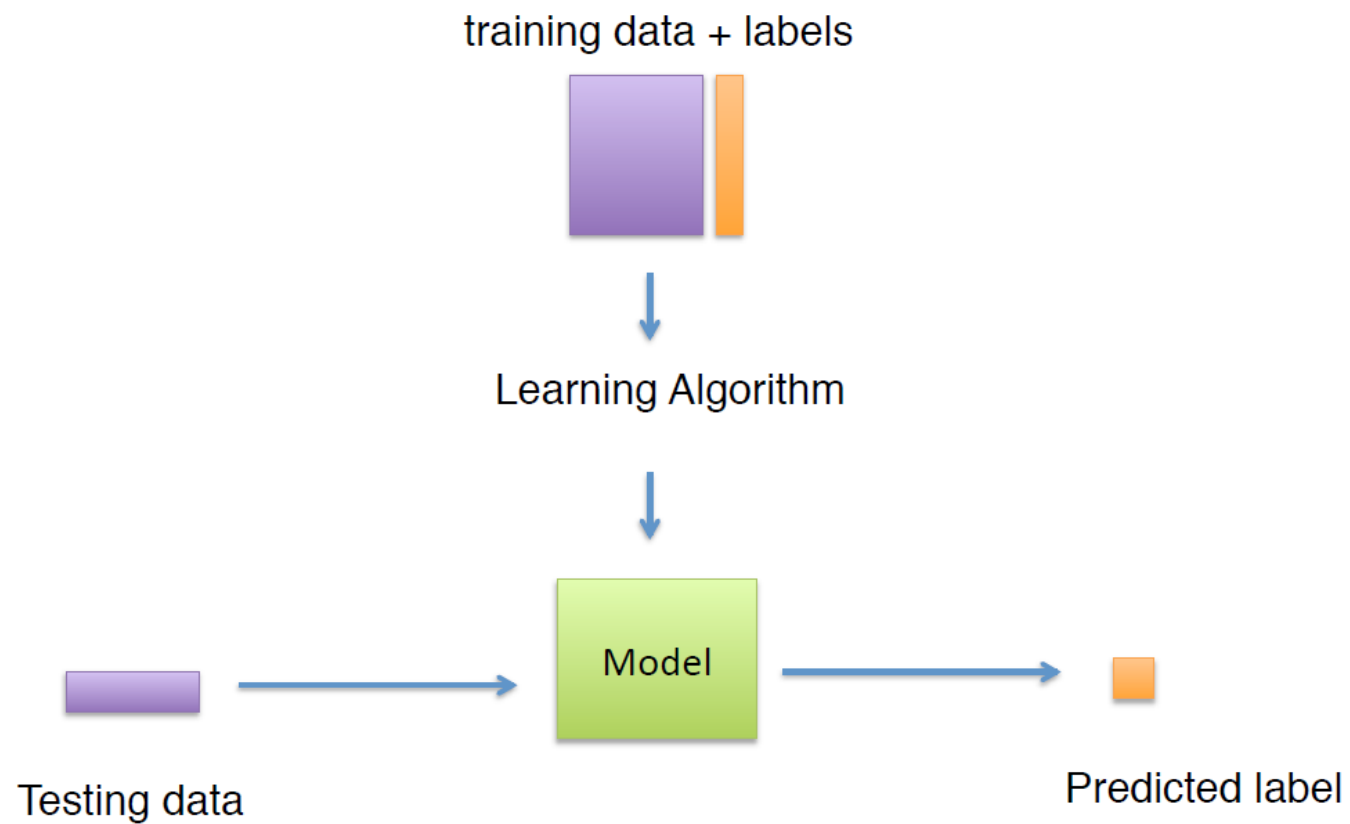
CLASSIFICATION

ENSEMBLES OF CLASSIFIERS

WITH SOME SLIDES FROM PROF. GAVIN BROWN

# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN



## Neural Network Ensembles

LARS KAI HANSEN AND PETER SALAMON

**Abstract**—We propose several means for improving the performance and training of neural networks for classification. We use crossvalidation as a tool for optimizing network parameters and architecture. We show further that the remaining residual “generalization” error can be reduced by invoking ensembles of similar networks.

**Index Terms**—Crossvalidation, fault tolerant computing, neural networks, *N*-version programming.

### I. INTRODUCTION

RECENT schemes for training neural networks involving hidden neurons have caused a resurgence of interest in nonalgorithmic supervised learning. A supervised learning scheme is implemented using a *database* which consists of a set of input patterns (a sample from the set of possible inputs) together with the corresponding targets (classifications). The objective of the training is to

performance of the network, we can optimize such performance by varying network characteristics and architecture.

A residual error will typically remain even after optimizing all available network characteristics [6]. To further reduce this error we propose to use a device from fault tolerant computing [7]. We run not a single network but an *ensemble* of networks, each of which have trained on the same database. The basic idea is to classify a given input pattern by obtaining a classification from each copy of the network and then using a consensus scheme to decide the collective classification by voting.

### II. CROSSVALIDATION FOR NETWORK OPTIMIZATION

For supervised learning we employ a database as described above including a representative sample of the

International Journal of Forecasting 5 (1989) 559–583  
North-Holland

559

## Combining forecasts: A review and annotated bibliography

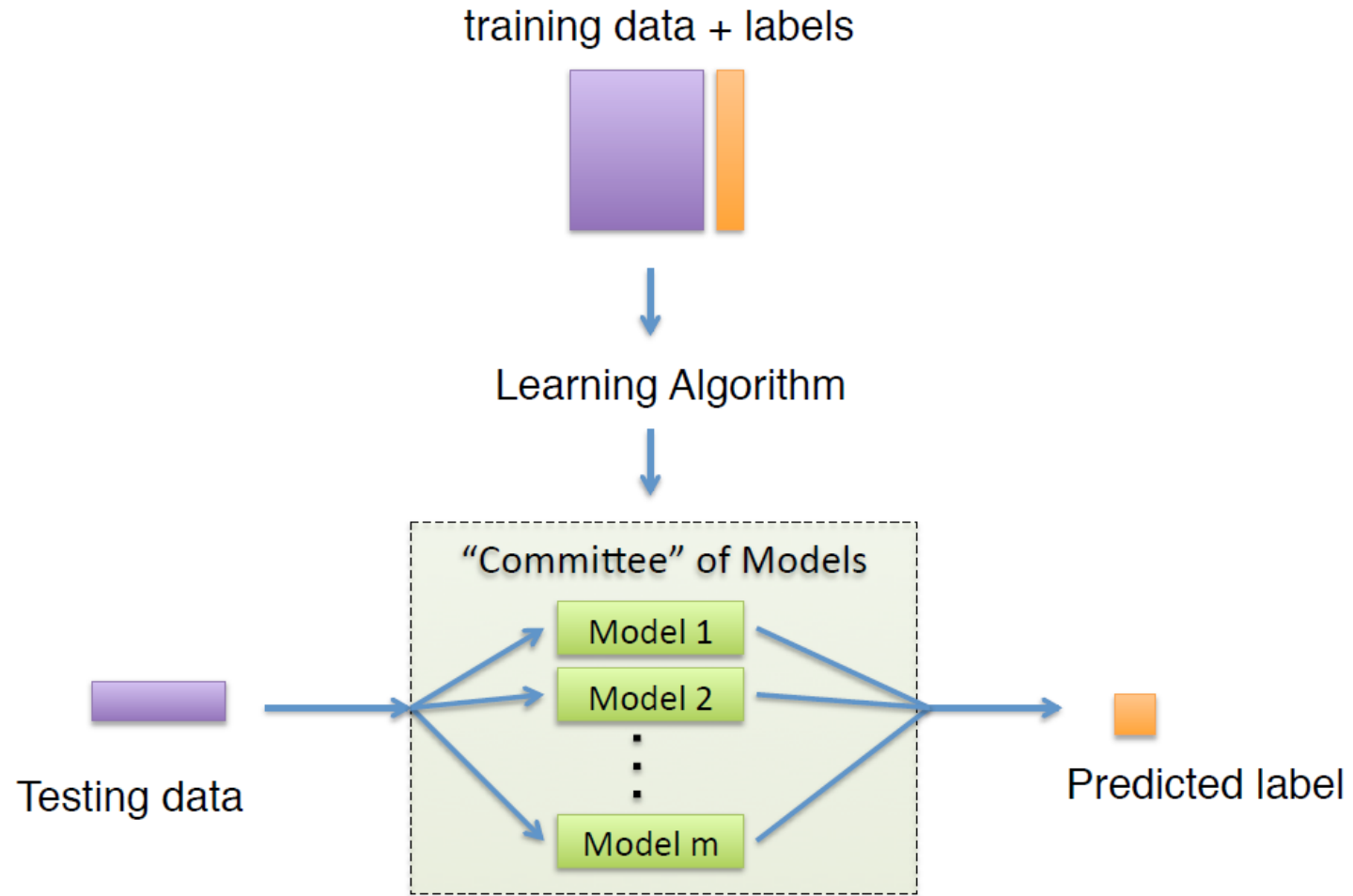
Robert T. CLEMEN \*

College of Business Administration, University of Oregon, Eugene, OR 97403-1208, USA

**Abstract:** Considerable literature has accumulated over the years regarding the combination of forecasts. The primary conclusion of this line of research is that forecast accuracy can be substantially improved through the combination of multiple individual forecasts. Furthermore, simple combination methods often work reasonably well relative to more complex combinations. This paper provides a review and annotated bibliography of that literature, including contributions from the forecasting, psychology, statistics, and management science literatures. The objectives are to provide a guide to the literature for students and researchers and to help researchers locate contributions in specific areas, both theoretical and applied. Suggestions for future research directions include (1) examination of simple combining approaches to determine reasons for their robustness, (2) development of alternative uses of multiple forecasts in order to

# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN

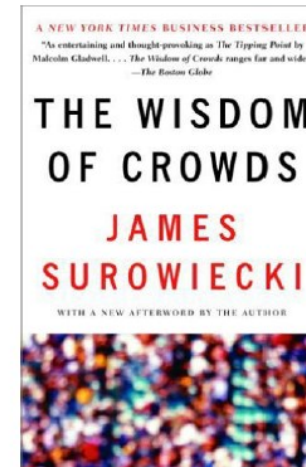


## Combining general predictions?

1906 ... county fair in Cornwall, England.

Competition: guess the weight of the cow!

Francis Galton recorded some statistics on the day....



NATURE

[MARCH 7, 1907]

mean  
of the  
ire for  
month

1 year-  
Both  
years.  
Bulletin  
ontains  
station  
ults of  
ions in  
151 and  
y in a  
ation is  
and is  
l appli-  
o with  
empera-  
ximum  
bsolute  
bsolute  
il rain-  
nt was  
Most  
tember.  
it being  
J. D.

*Distribution of the estimates of the dressed weight of a particular living ox, made by 787 different persons.*

Degrees of the length of Array 0°-100°	Estimates in lbs.	Centiles		Excess of Observed over Normal
		Observed deviates from 1207 lbs.	Normal p.e = 37	
5	1074	-133	-90	+43
10	1109	-98	-70	+28
15	1126	-81	-57	+24
20	1148	-59	-46	+13
$q_1$ 25	1162	-45	-37	+8
30	1174	-33	-29	+4
35	1181	-26	-21	+5
40	1188	-19	-14	+5
45	1197	-10	-7	+3
$m$ 50	1207	0	0	0
55	1214	+7	+7	0
60	1219	+12	+14	-2
65	1225	+18	+21	-3
70	1230	+23	+29	-6
$q_3$ 75	1236	+29	+37	-8
80	1243	+36	+46	-10
85	1254	+47	+57	-10
90	1267	+52	+70	-18
95	1293	+86	+90	-4

$q_1$ ,  $q_3$ , the first and third quartiles, stand at 25° and 75° respectively.  
 $m$ , the median or middlemost value, stands at 50°.  
The dressed weight proved to be 1198 lbs.



787 guesses.

Truth      1198 lb  
(~543kg)

Median    1207 lb  
Mean      1197 lb

results  
eutsche  
second

## Combining Votes

In 1786 Nicolas de Condorcet (political theorist) asked how do parliaments behave when voting?

...assuming  $M$  independent voters...

If a single voter has a probability  $\epsilon$  of making an error, then

$$p(\text{exactly } k \text{ errors}) = \binom{M}{k} \epsilon^k (1 - \epsilon)^{(M-k)}$$

and...

$$p(\text{majority vote error}) = \sum_{k \geq \lceil \frac{M+1}{2} \rceil} \binom{M}{k} \epsilon^k (1 - \epsilon)^{(M-k)}$$



Marquis de  
Condorcet

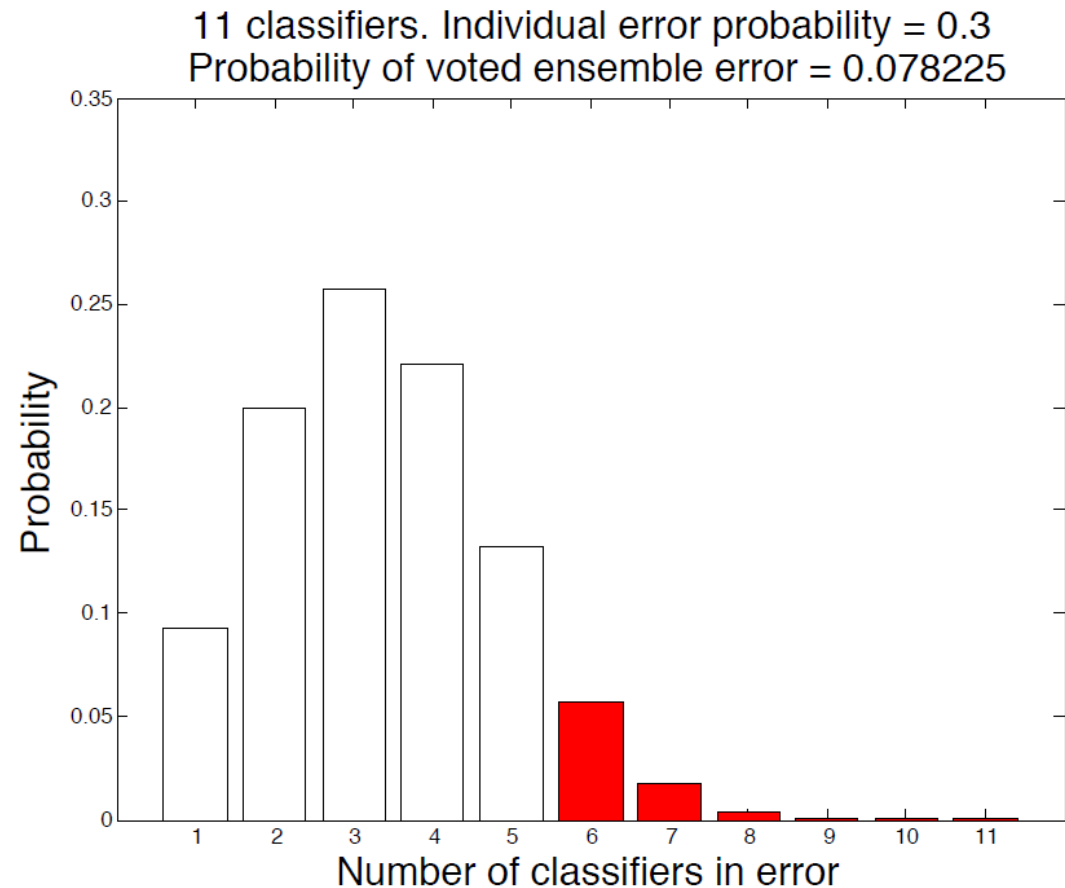




# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN

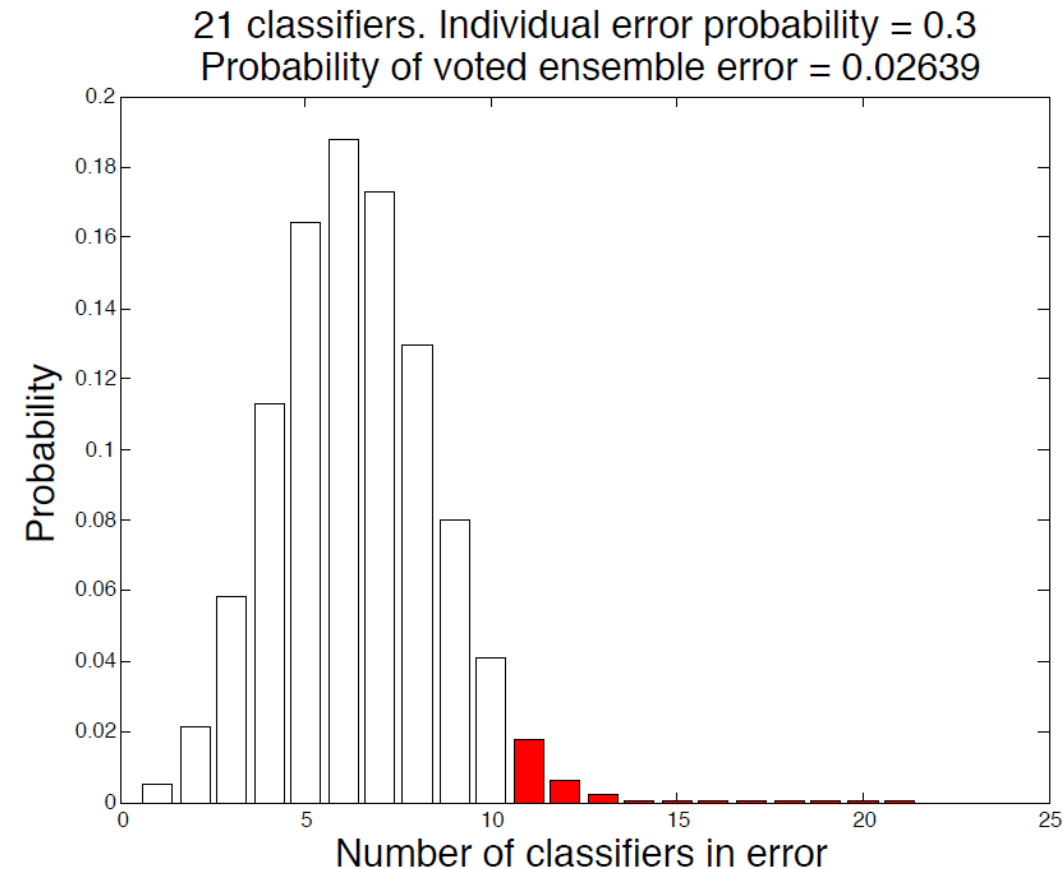
$$p(\text{majority vote error}) = \sum_{k \geq \lceil \frac{M+1}{2} \rceil} \binom{M}{k} \epsilon^k (1 - \epsilon)^{M-k}$$



# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN

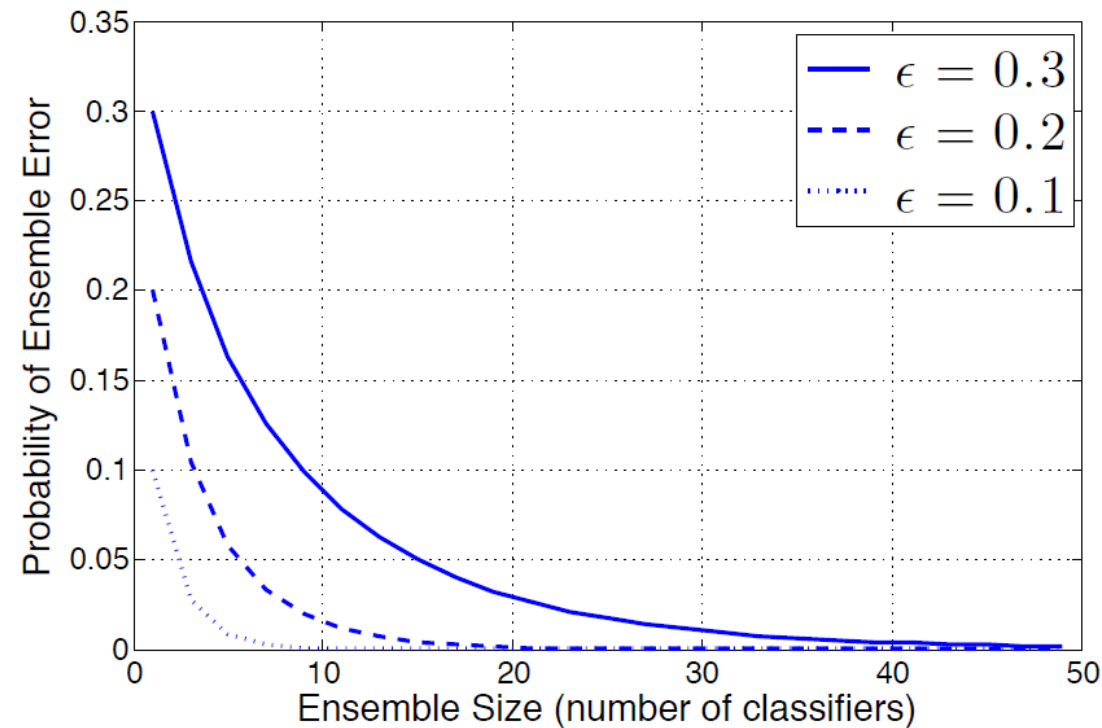
$$p(\text{majority vote error}) = \sum_{k \geq \lceil \frac{M+1}{2} \rceil} \binom{M}{k} \epsilon^k (1 - \epsilon)^{M-k}$$



# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN

$$p(\text{majority vote error}) = \sum_{k \geq \lceil \frac{M+1}{2} \rceil} \binom{M}{k} \epsilon^k (1 - \epsilon)^{M-k}$$



Virtually ZERO error by  $M = 50$  !!

$$\varphi(\mathbb{E}[X]) \leq \mathbb{E}[\varphi(X)]$$

← Jensen's inequality

$$\phi\left(\sum_{i=1}^M \lambda_i x_i\right) \leq \sum_{i=1}^M \lambda_i \phi(x_i) \quad \sum_i \lambda_i = 1$$

$$\text{es... } \phi(z) = (z-t)^2$$

$$\text{e } \lambda_i = \frac{1}{M} \text{ per tutti i valori di } i$$

→

$$\left(\sum_i \lambda_i x_i - t\right)^2 \leq \sum_i \lambda_i (x_i - t)^2$$

$$\left(\sum_i \frac{x_i}{M} - t\right)^2 \leq \sum_i \frac{(x_i - t)^2}{M}$$

$$\left(\frac{1}{M} \sum_i x_i - t\right)^2 \leq \frac{1}{M} \sum_i (x_i - t)^2$$

The squared error of a linear combination of predictions...

...is guaranteed to be less than or equal to...

...the average squared error of the individual predictions.

→

$$\left(\sum_i d_i x_i - t\right)^2 \leq \sum_i d_i (x_i - t)^2$$

$$\left(\sum_i \frac{x_i}{M} - t\right)^2 \leq \sum_i \frac{(x_i - t)^2}{M}$$

$$\left(\frac{1}{M} \sum_i x_i - t\right)^2 \leq \frac{1}{M} \sum_i (x_i - t)^2$$

Training *different* classifiers . . .

but how “different” the single classifiers should be?

Training *different* classifiers . . .

but how "different" the single classifiers should be?

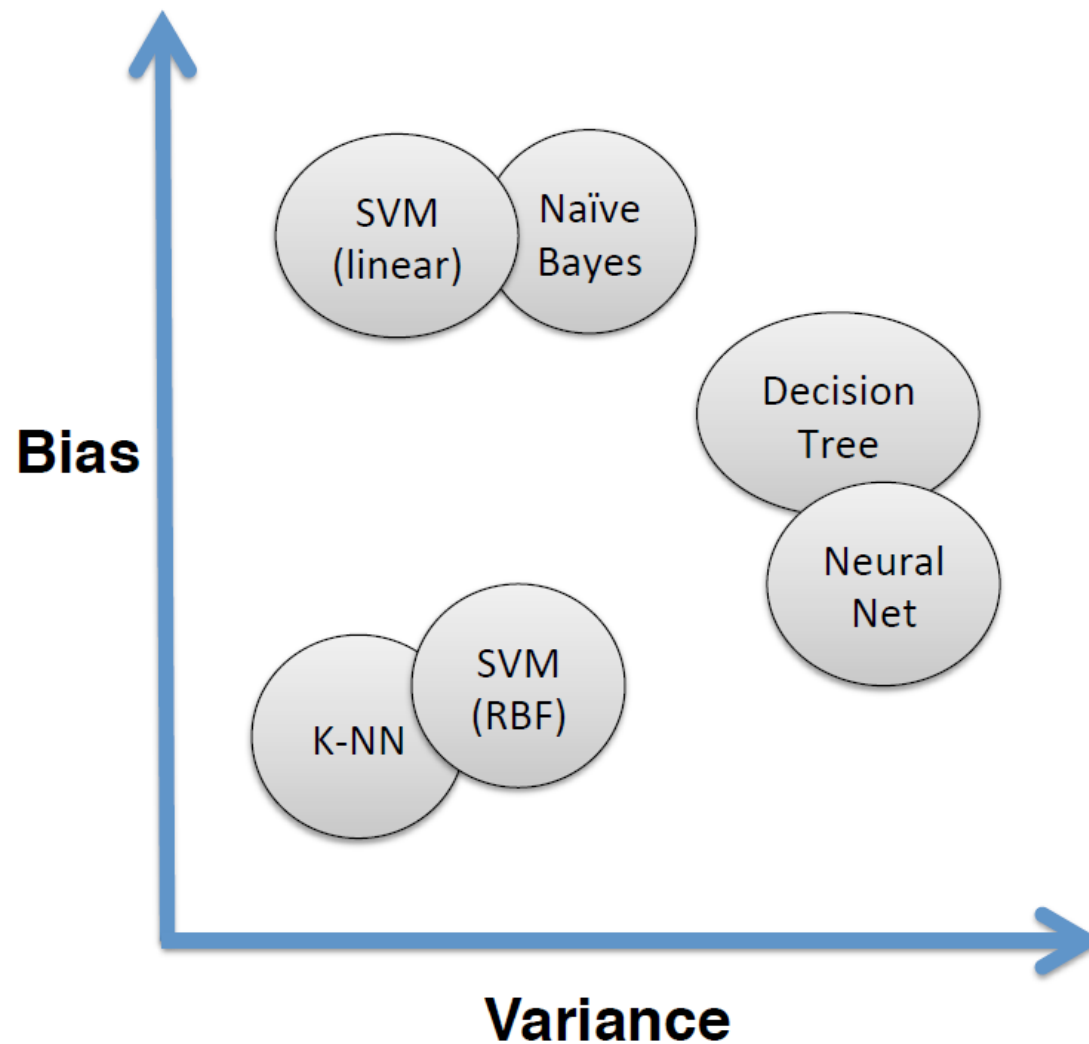
DIFFERENT  
(?)

ACCURATE  
(?)

INDEPENDENT  
(?)

# Ensembles of Classifiers

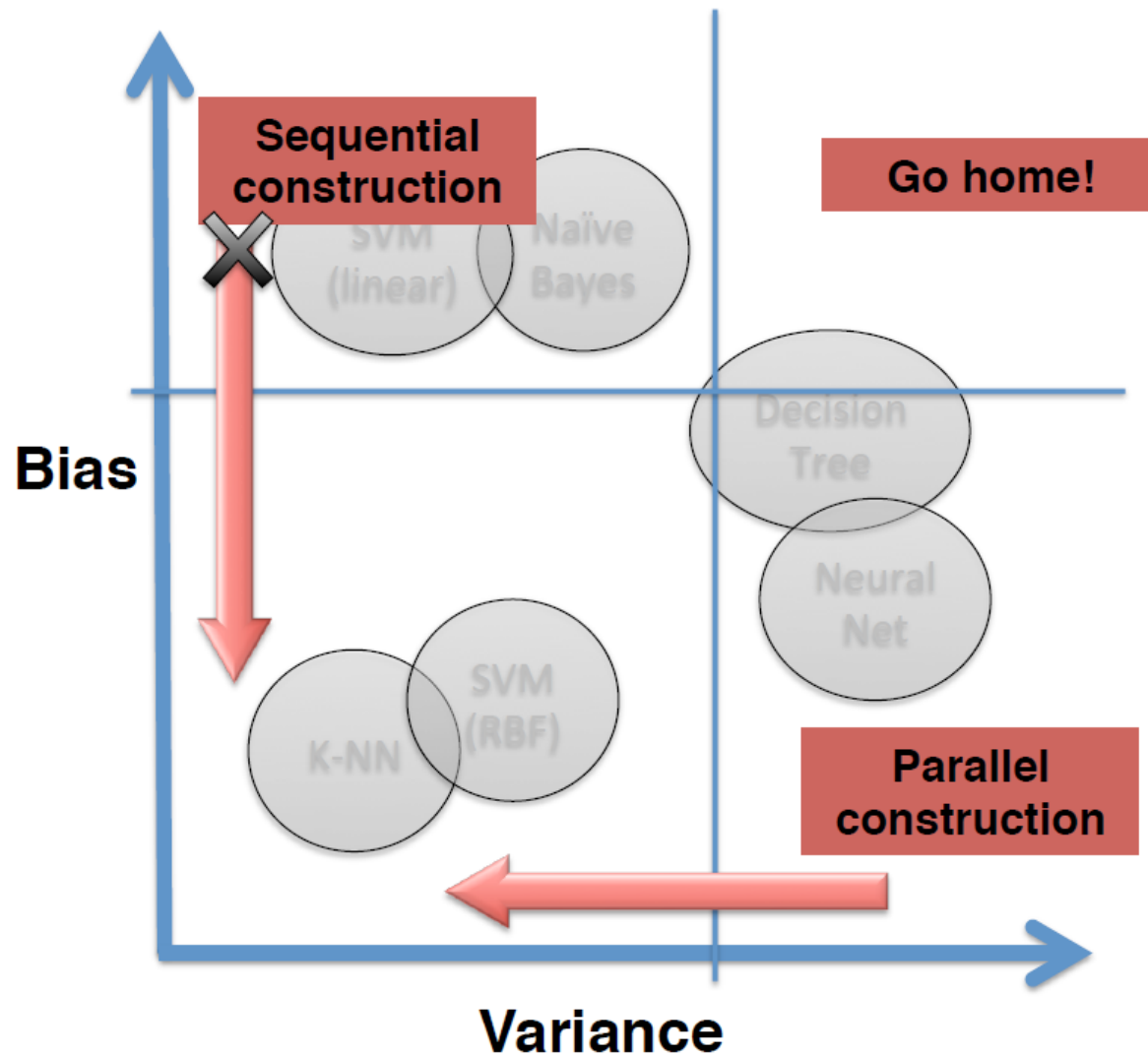
WITH SOME SLIDES FROM PROF. GAVIN BROWN





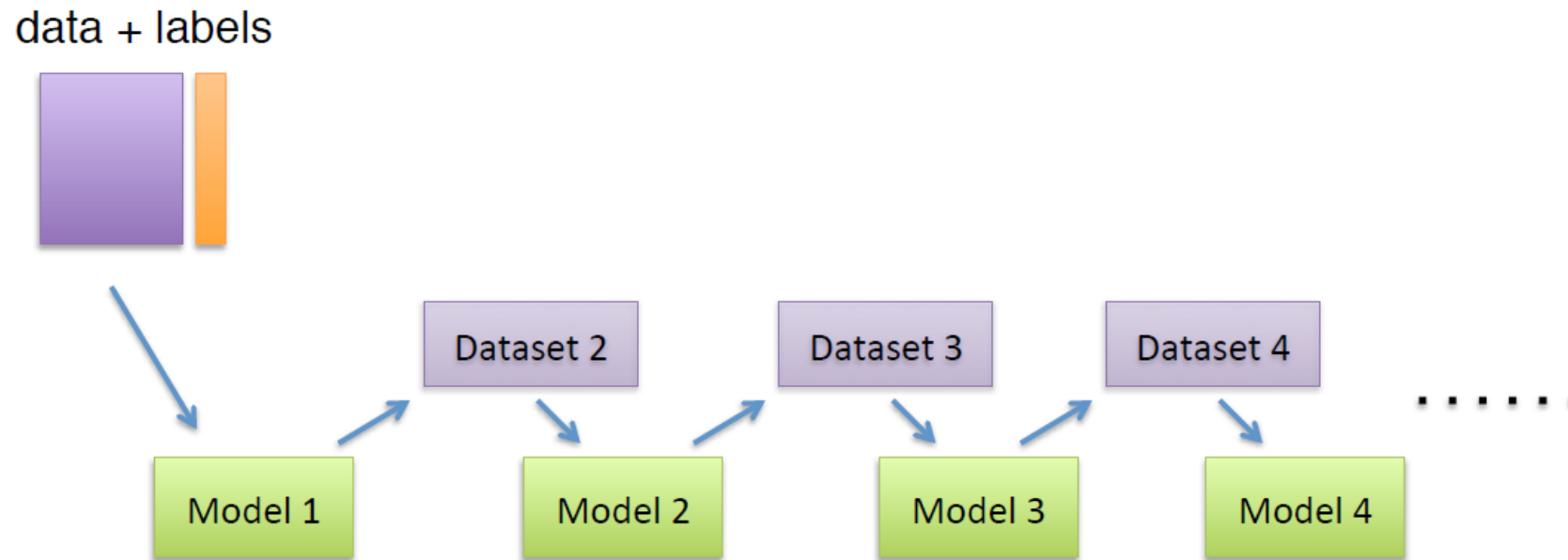
# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN



# Ensembles of Classifiers

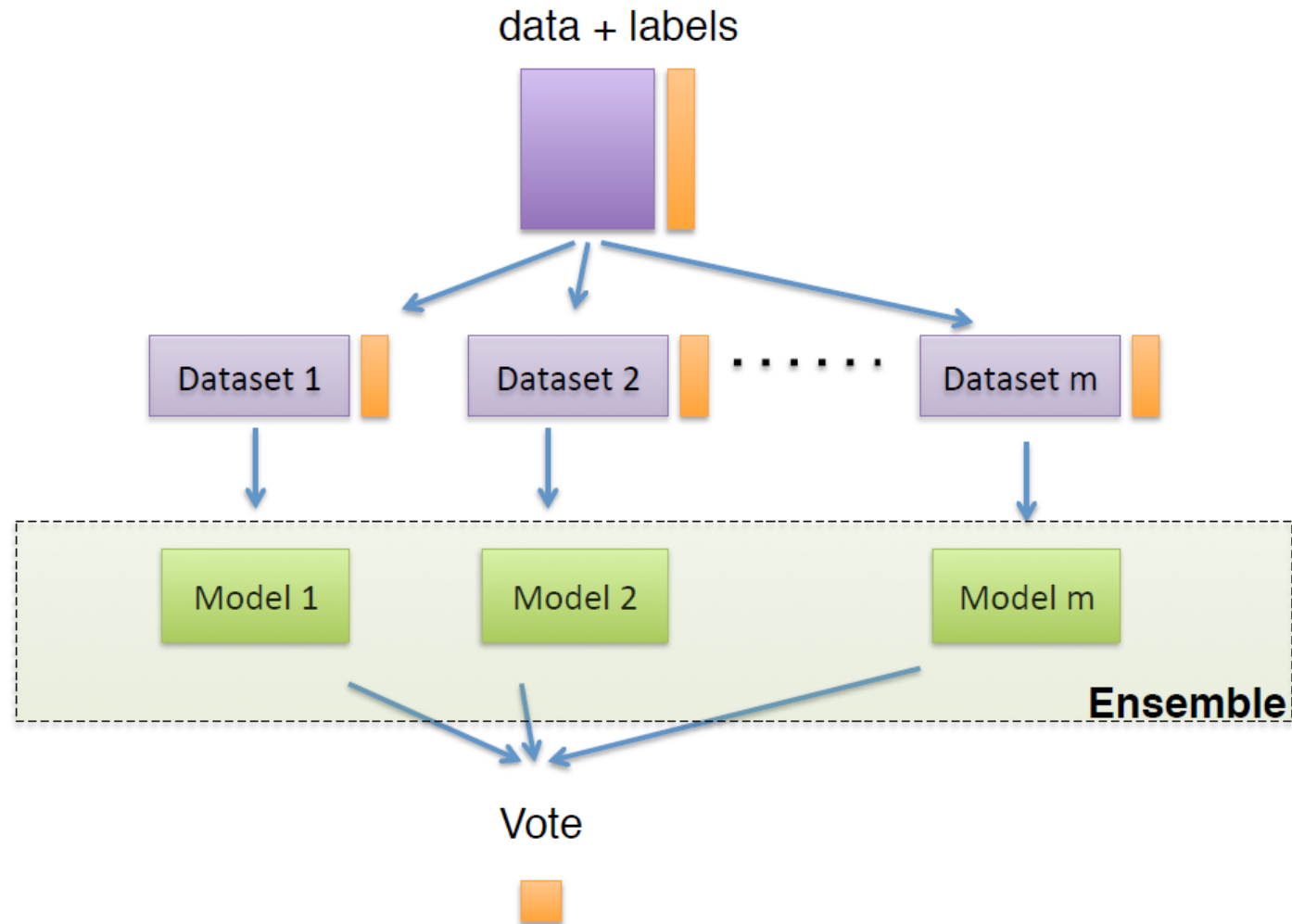
WITH SOME SLIDES FROM PROF. GAVIN BROWN

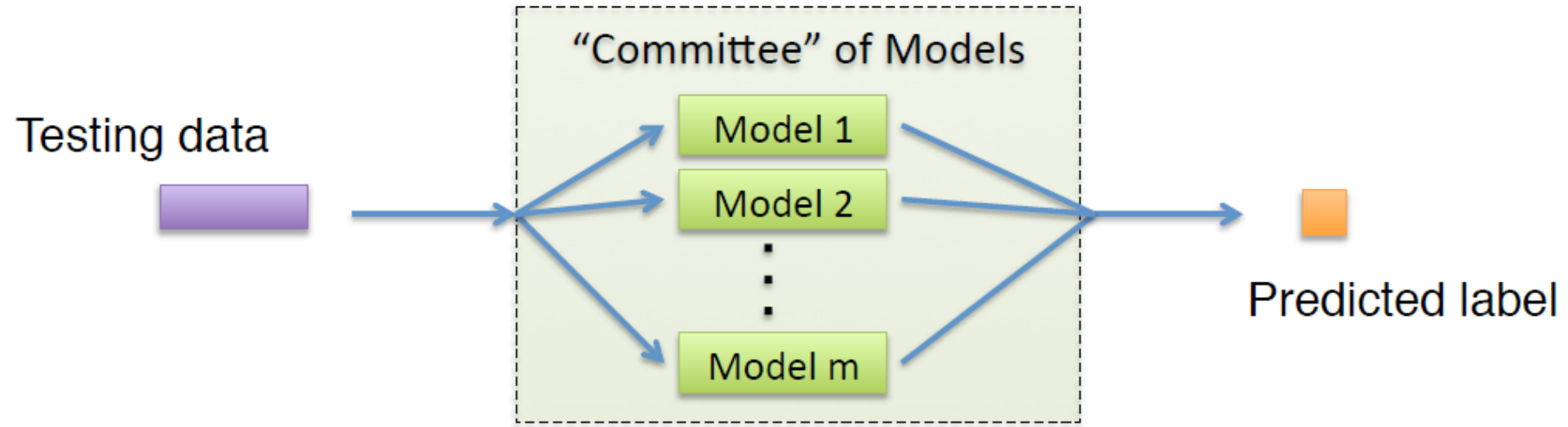


Each model corrects the mistakes of its predecessor.

# Ensembles of Classifiers

WITH SOME SLIDES FROM PROF. GAVIN BROWN





Decisions of individuals combined.

At testing phase is the same...

Which are the most popular methods to build  
“ensemble” classifiers?

BAGGING

BOOSTING

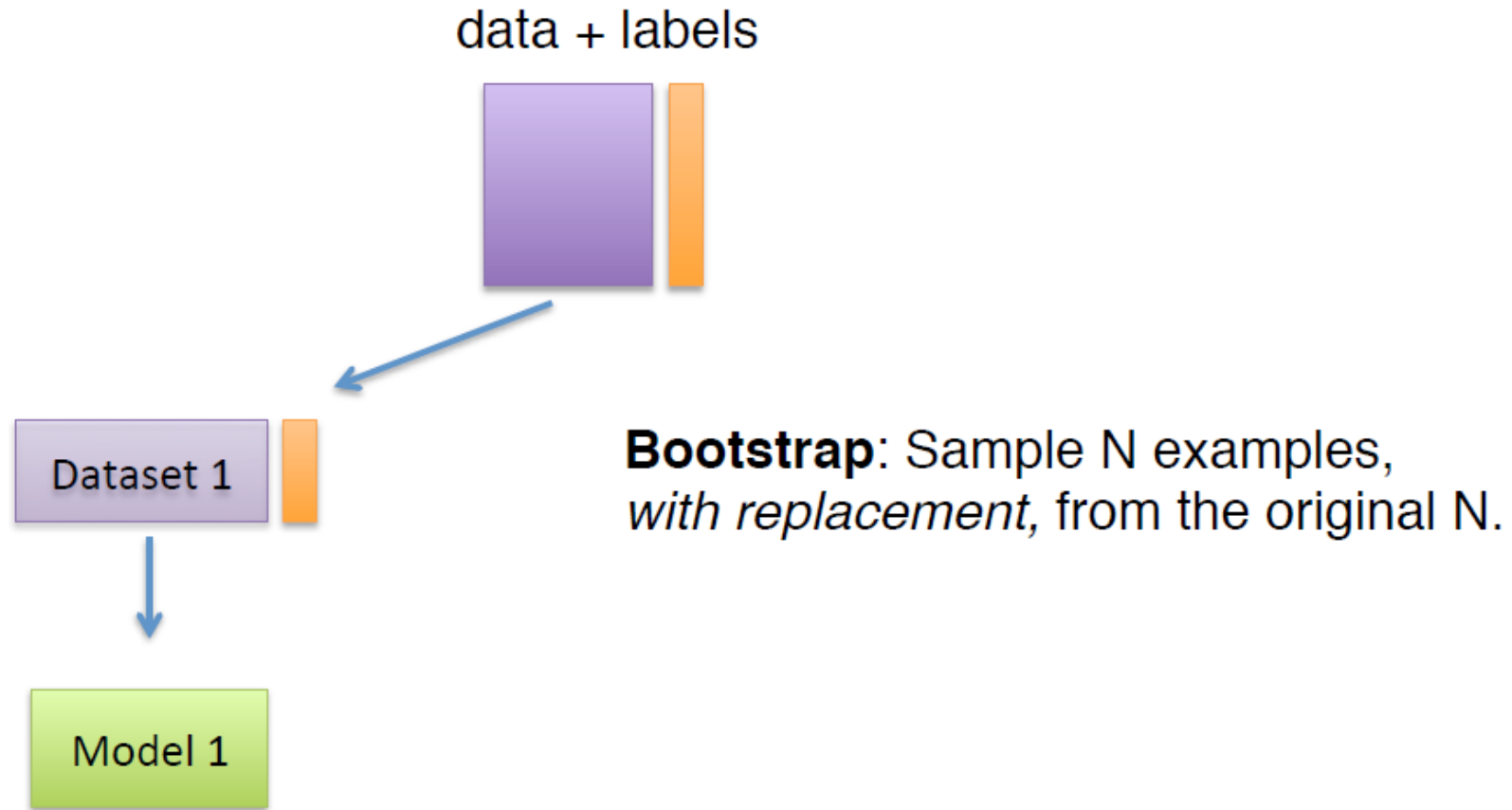
STACKING

BLENDING

# Ensembles of Classifiers | Bagging

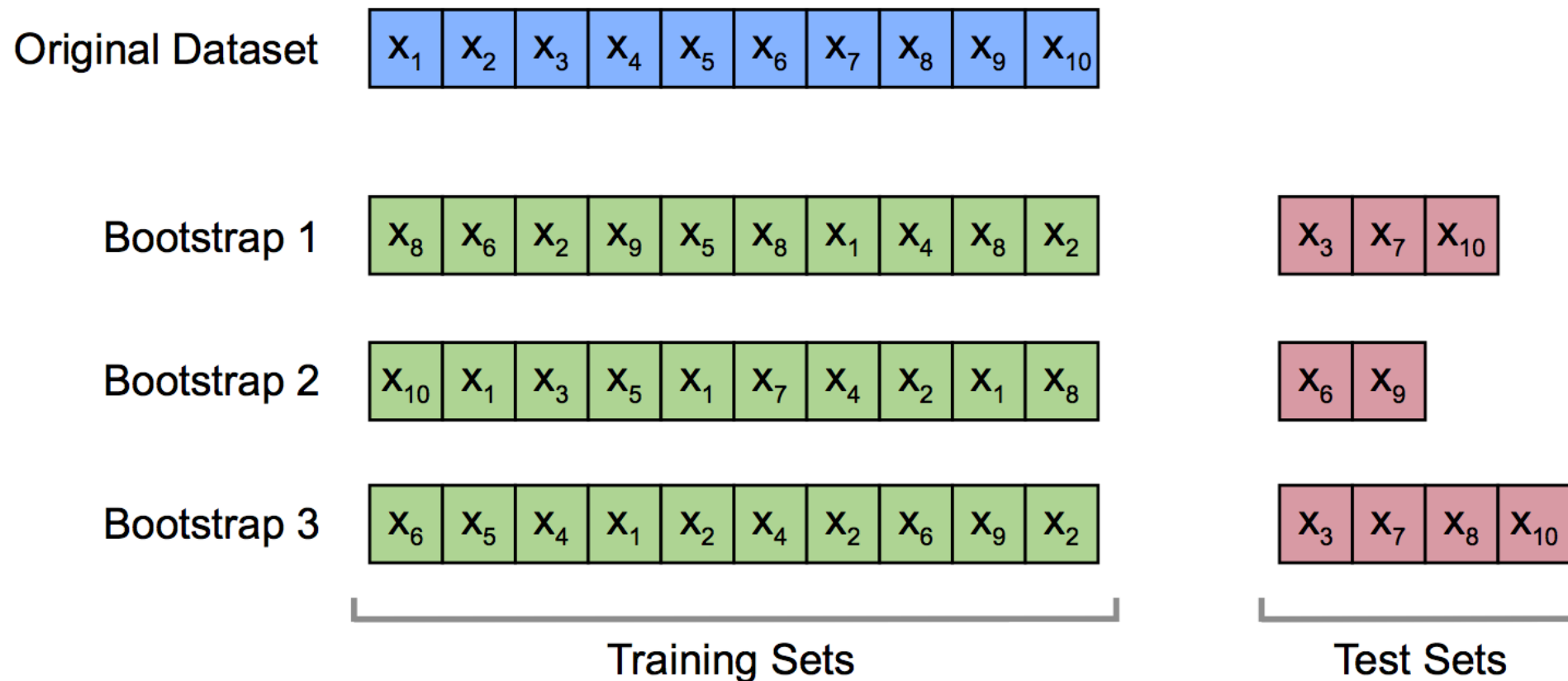
WITH SOME SLIDES FROM PROF. GAVIN BROWN

BAGGING = BOOTSTRAP AGGREGATING



# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN



This work by Sebastian Raschka is licensed under a Creative Commons Attribution 4.0 International License.

# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN

Probability of including any given example in a bootstrap:

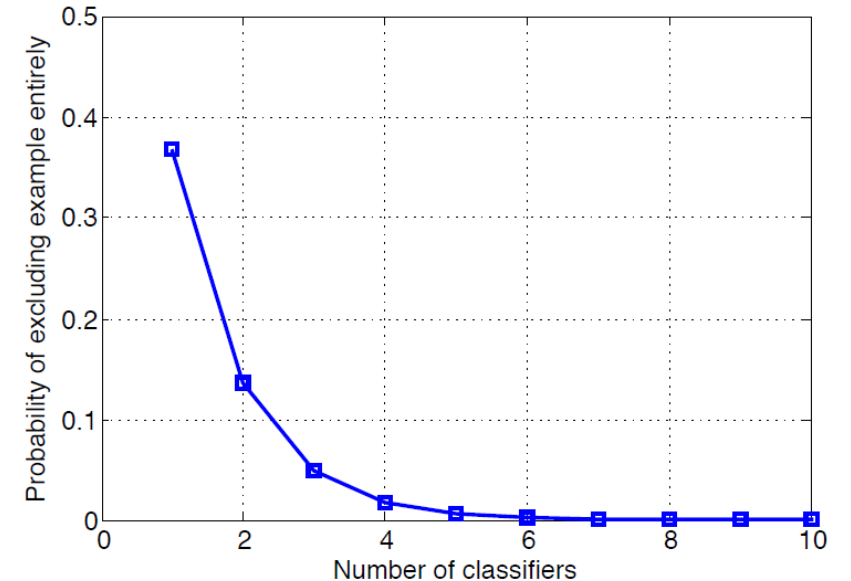
$$p = 1 - \left(1 - \frac{1}{N}\right)^N$$

$$\lim_{N \rightarrow \infty} \left\{ 1 - \left(1 - \frac{1}{N}\right)^N \right\} = 1 - e^{-1} \approx 0.6321$$

~63% of the data included.

... 37% left out!?

PROBABILITY OF EXCLUDING A SAMPLE  
FROM THE WHOLE ENSEMBLE





---

**Bagging** (input training data+labels  $T$ , number of models  $M$ )

---

```
for  $j = 1$  to  $M$  do  
    Take a bootstrap sample  $T'$  from  $T$   
    Build a model using  $T'$ .  
    Add the model to the set.  
end for  
return set of models
```

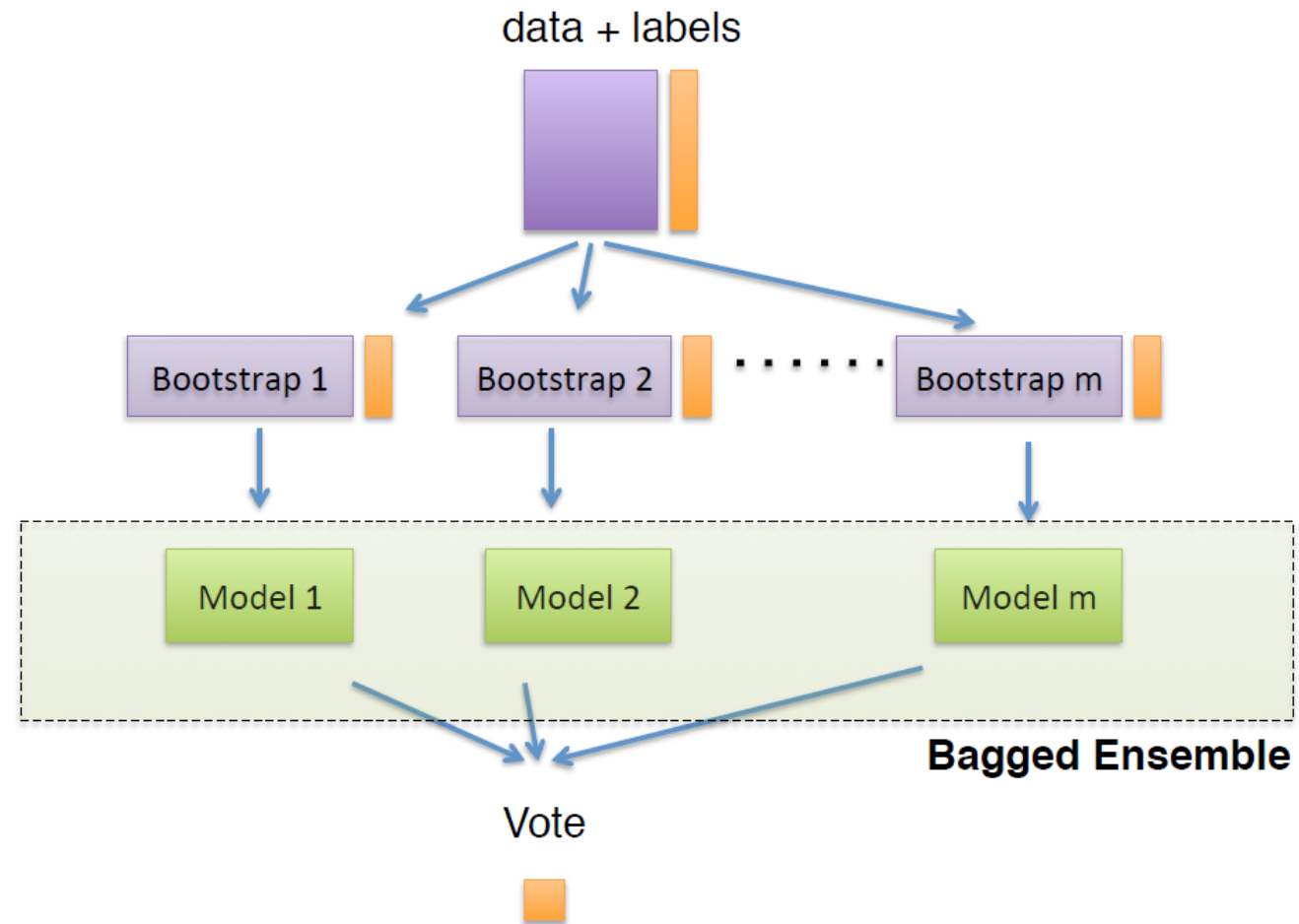
For a test point  $\mathbf{x}$ , get a response from each model, and take a majority vote.

---

Breiman, 1996  
(who will implement random forests 4 years later...)

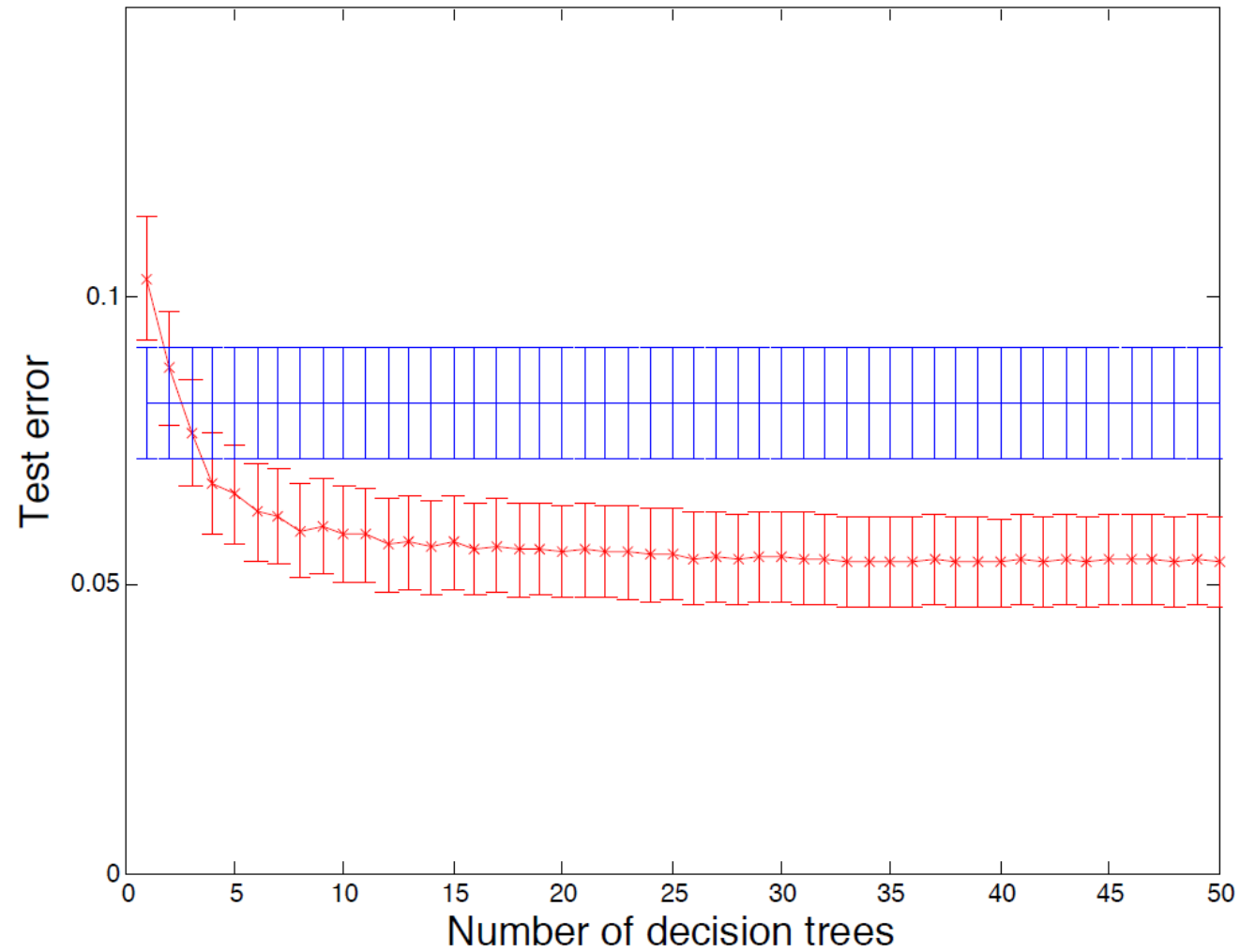
# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN



# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN



# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN

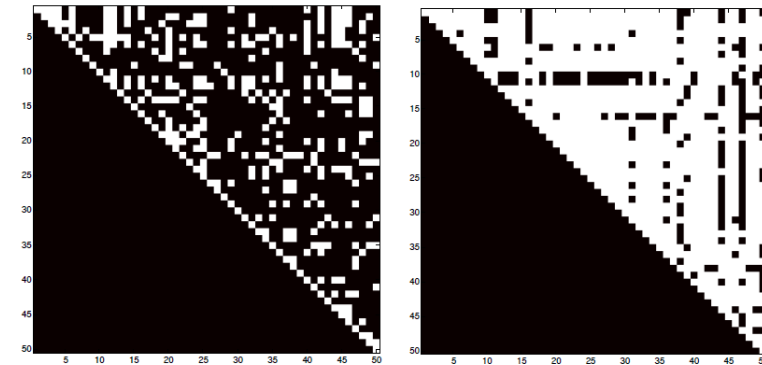
Training *different* classifiers . . .

but how “different” the single classifiers should be?

DIFFERENT  
(!)

ACCURATE  
(~)

INDEPENDENT  
(?)



Dependencies between 50 bagged classifiers – CART (left), Gaussian Naive Bayes (right).

( $\chi^2$  test,  $\alpha = 0.05$ )

# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN

A note on model stability

Some models are almost completely unaffected by bootstrapping (similar to each other on test data)

STABLE MODELS are not the first choice for parallel ensemble methods

Neural Networks  
Logistic Regression  
kNN  
SVM  
Decision Trees



# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN

A note on model stability

Some models are almost completely unaffected by bootstrapping (similar to each other on test data)

STABLE MODELS are not the first choice for parallel ensemble methods

kNN  
SVM  
Decision Trees

Logistic Regression

Neural Networks

STABLE

UNSTABLE



# Ensembles of Classifiers | Bagging

WITH SOME SLIDES FROM PROF. GAVIN BROWN

A note on model stability

Some models are almost completely unaffected by bootstrapping (similar to each other on test data)

STABLE MODELS are not the first choice for parallel ensemble methods



## Stacking or Stacked Generalization

"Given multiple machine learning models that are skillful on a problem, but in different ways, how do you choose which model to use (trust)?"

Use a second-level (meta) machine-learning model that learns when to use (trust) each model in the ensemble



Architecture of a stacking model:

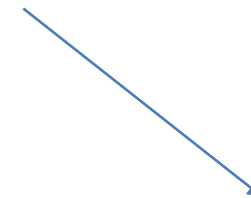
- two or more base models (often referred to as LEVEL-0 models), which learn to discriminate training data at a single-classifier level and whose predictions are computed and recorded (to be used at level 1)
- one meta-model (LEVEL 1) that combines the predictions of the base models (referred to as a LEVEL-1 model) by learning how to best combine LEVEL-0 models (or their predictions)

Architecture of a stacking model:

- two or more base models (often referred to as LEVEL-0 models), which learn to discriminate training data at a single-classifier level and whose predictions are computed and recorded (to be used at level 1)
- one meta-model (LEVEL 1) that combines the predictions of the base models (referred to as a LEVEL-1 model) by learning how to best combine LEVEL-0 models (or their predictions)

SHOULD BE:

1. sufficiently accurate
2. Sufficiently different (assumptions about the predictive task)
3. sufficiently independent from each other (uncorrelated in their predictions)



Architecture of a stacking model:

- two or more base models (often referred to as LEVEL-0 models), which learn to discriminate training data at a single-classifier level and whose predictions are computed and recorded (to be used at level 1)
- one meta-model (LEVEL 1) that combines the predictions of the base models (referred to as a LEVEL-1 model) by learning how to best combine LEVEL-0 models (or their predictions)

LEARN classifiers' combination  
through cross validation (or similar)

SHOULD BE:

1. sufficiently accurate
2. Sufficiently different (assumptions about the predictive task)
3. sufficiently independent from each other (uncorrelated in their predictions)

# Ensembles of Classifiers | Blending

WITH SOME SLIDES FROM PROF. GAVIN BROWN

“ *Blending is a word introduced by the Netflix winners. It is very close to stacked generalization, but a bit simpler and less risk of an information leak. [...] With blending, instead of creating out-of-fold predictions for the train set, you create a small holdout set of say 10% of the train set. The stacker model then trains on this holdout set only.*

— [Kaggle Ensemble Guide](#), MLWave, 2015.

## The BellKor 2008 Solution to the Netflix Prize

Robert M. Bell  
AT&T Labs - Research  
Florham Park, NJ

Yehuda Koren  
Yahoo! Research  
Haifa, Israel

Chris Volinsky  
AT&T Labs - Research  
Florham Park, NJ

[BellKor@research.att.com](mailto:BellKor@research.att.com)

## 1. Introduction

Our  $RMSE=0.8643^2$  solution is a linear blend of over 100 results. Some of them are new to this year, whereas many others belong to the set that was reported a year ago in our 2007 Progress Prize report [3]. This report is structured accordingly. In Section 2 we detail methods new to this year. In general, our view is that those newer methods deliver a superior performance compared to the methods we used a year ago. Throughout the description of the methods, we highlight the specific predictors that participated in the final blended solution. Nonetheless, the older methods still play a role in the blend, and

- **Blending:** Stacking-type ensemble where the meta-model is trained on predictions made on a holdout dataset.
- **Stacking:** Stacking-type ensemble where the meta-model is trained on out-of-fold predictions made during k-fold cross-validation.

# Ensembles of Classifiers | Boosting

WITH SOME SLIDES FROM PROF. GAVIN BROWN

---

Define a distribution over the training set,  $D_1(i) = \frac{1}{N}, \forall i$ .

**for**  $t = 1$  to  $T$  **do**

    Build a model  $h_t$  from the training set, using distribution  $D_t$ .

    Update  $D_{t+1}$  from  $D_t$ :

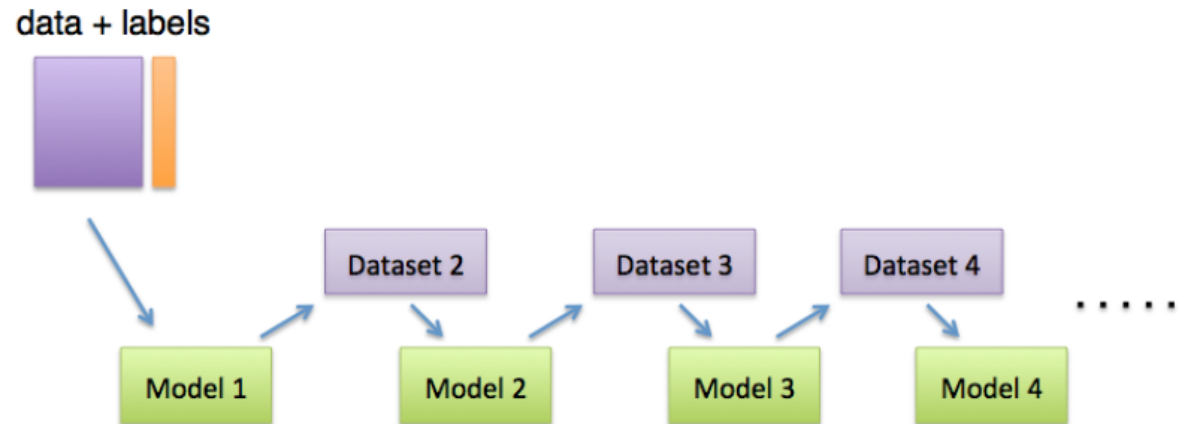
        Increase the weight on examples that  $h_t$  incorrectly classifies.

        Decrease the weight on examples that  $h_t$  correctly classifies.

**end for**

For a new testing point  $(\mathbf{x}', y')$ , we take a weighted majority vote from  $\{h_1, \dots, h_T\}$ .

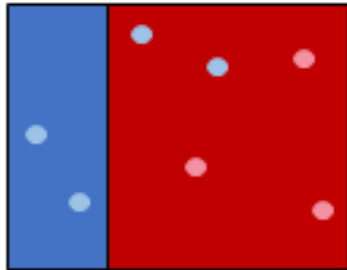
---



## ADABOOST

### AdaBoost:

- Combining **weak learners** (decision trees)
- Assigning **weights to incorrect values**
- **Sequential tree growing** considering past mistakes

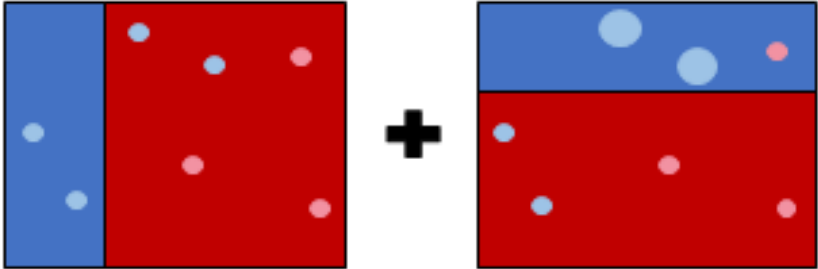


Results of  
tree 1

## ADABOOST

**AdaBoost:**

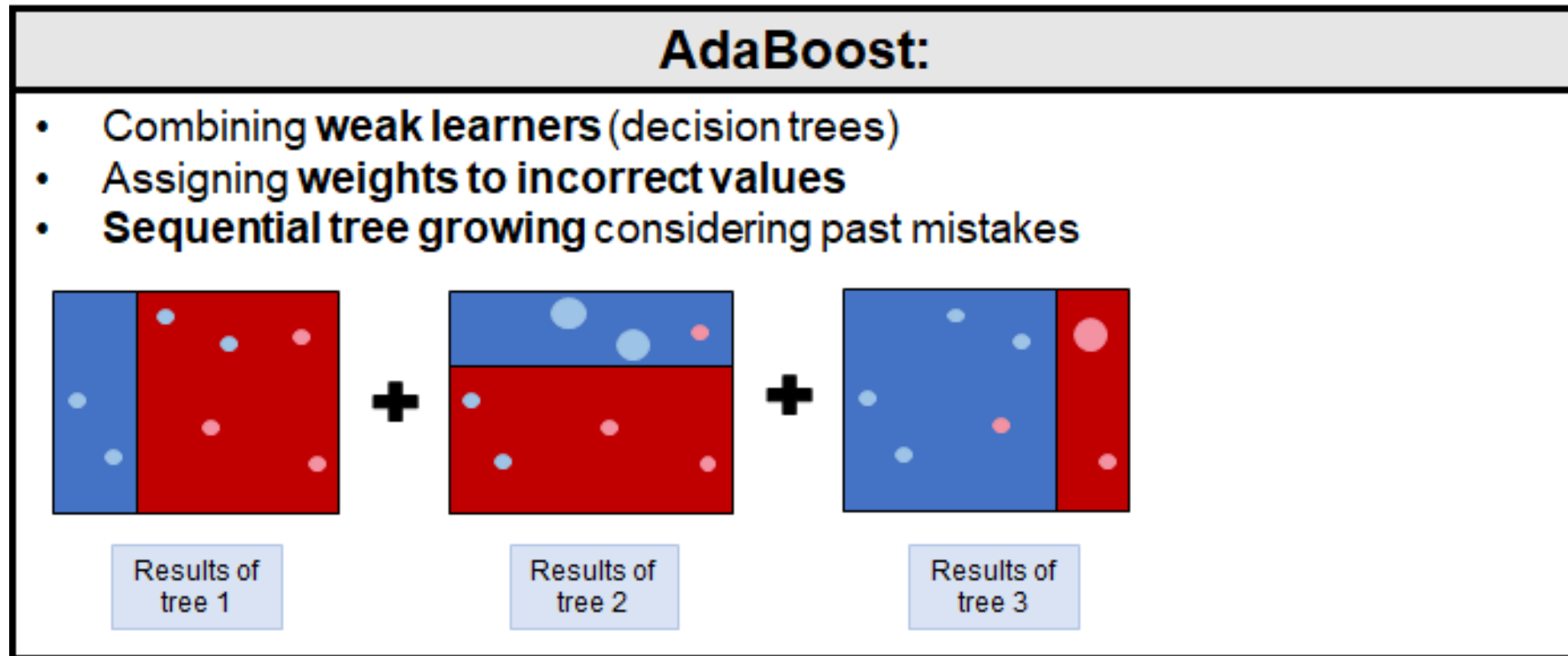
- Combining **weak learners** (decision trees)
- Assigning **weights** to **incorrect values**
- **Sequential tree growing** considering past mistakes



Results of tree 1

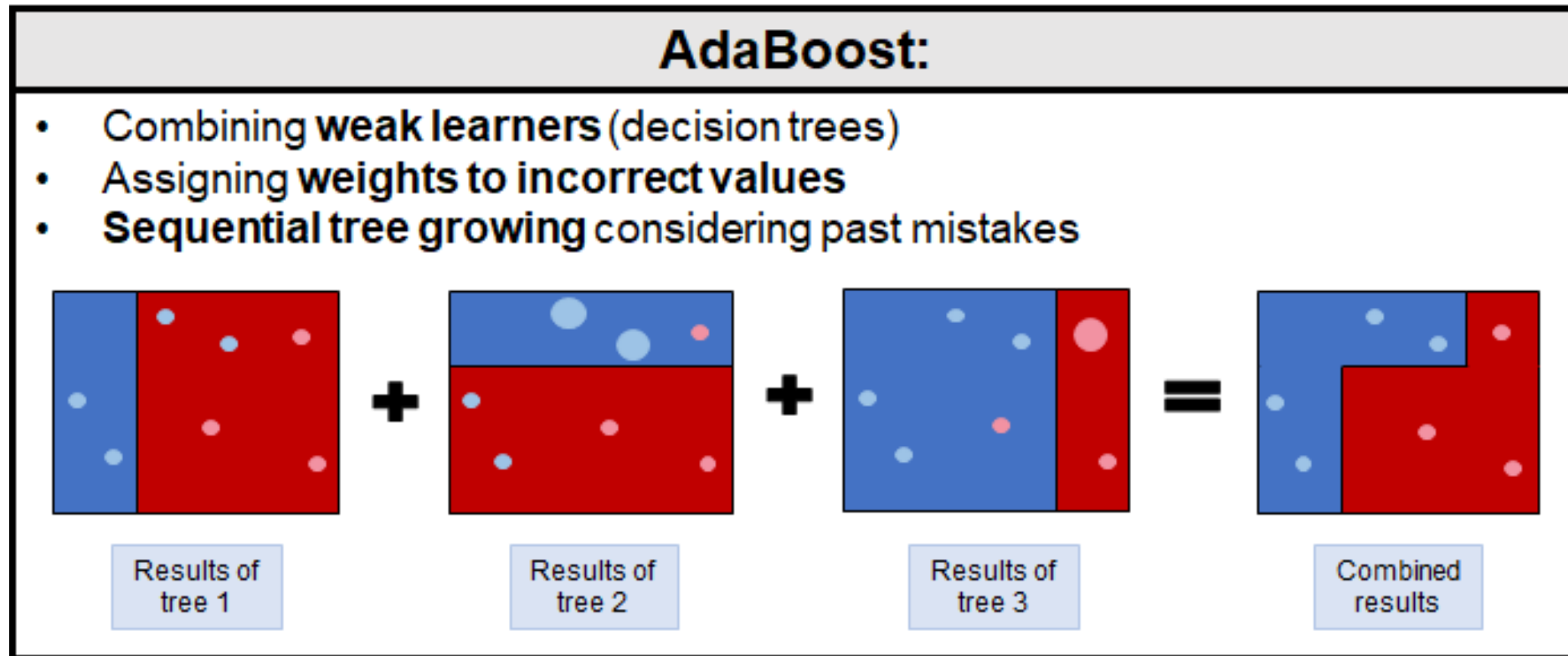
Results of tree 2

## ADABOOST





## ADABOOST



## ADABOOST

Define a distribution over the training set,  $D_1(i) = \frac{1}{N}$ ,  $\forall i$ .

**for**  $t = 1$  to  $T$  **do**

Build a classifier  $h_t$  from the training set, using distribution  $D_t$ .

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

Update  $D_{t+1}$  from  $D_t$  :

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

**end for**

$$H(x') = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x') \right)$$

$$\text{where } Z_t = \sum_i D_t(i) e^{-y_i \alpha_t h_t(x_i)}$$

# Ensembles of Classifiers | Boosting

WITH SOME SLIDES FROM PROF. GAVIN BROWN

## ADABOOST

Define a training set  $D_t$  with  $N$  data points  $(x_i, y_i)$   
**for**  $t = 1$  **to**  $T$   
    Build a classifier  $h_t$  from the training set  $D_t$   
    Set  $\alpha_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$   
    Update  $D_{t+1}$  from  $D_t$  :  
        Set  $D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_t}$   
**end for**

$$H(x') = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x') \right)$$

Weight Update

Error rate

$$E_1 = \frac{1}{N} \sum_{i=1}^N e^{-y_i h_1(x_i)}$$

Exponential loss

Sum of the exponential loss on each data point

$$E_2 = \sum_{i=1}^N \underbrace{\frac{1}{N} e^{-y_i \alpha_1 h_1(x_i)} e^{-y_i \alpha_2 h_2(x_i)}}_{\text{constant}}$$

$$E_2 = \sum_{i=1}^N w_2(i) e^{-y_i \alpha_2 h_2(x_i)}$$

$$\text{where } Z_t = \sum_i D_t(i) e^{-y_i \alpha_t h_t(x_i)}$$

## ADABOOST

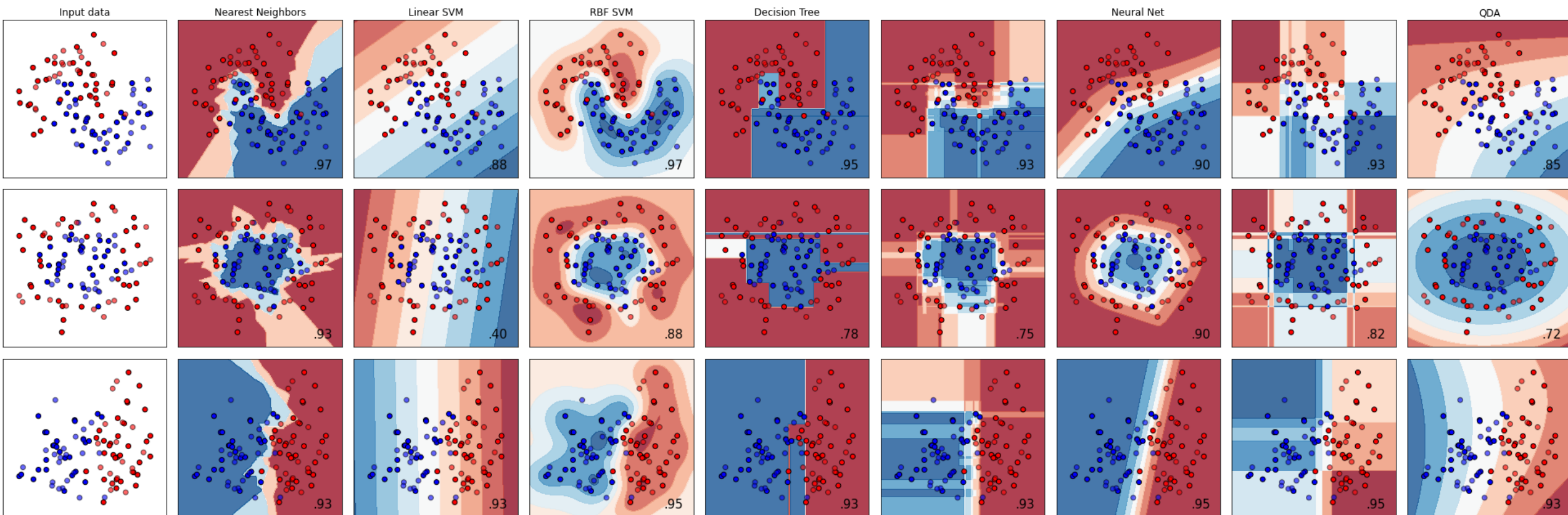
Good performance with non-noisy datasets

Controversial results with noisy data

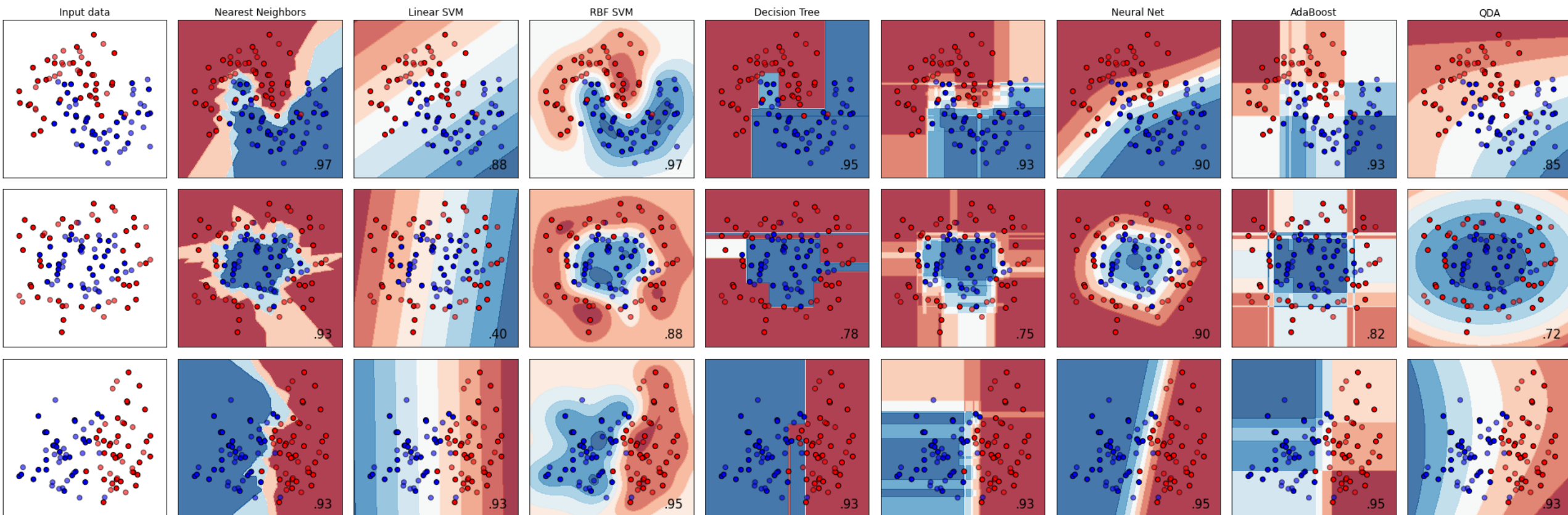
Low number of hyperparameters

Not optimized for computational costs in terms of time

# Which is which | Decision Function



# Which is which | Decision Function



## To Ensemble or Not Ensemble: When does End-To-End Training Fail?

Andrew Webb<sup>1</sup>, Charles Reynolds<sup>1</sup>, Wenlin Chen<sup>1</sup>, Henry Reeve<sup>2</sup>, Dan Iliescu<sup>3</sup>,  
Mikel Luján<sup>1</sup>, and Gavin Brown<sup>1</sup>

<sup>1</sup> University of Manchester, UK

<sup>2</sup> University of Bristol, UK

<sup>3</sup> University of Cambridge, UK

**Abstract.** End-to-End training (E2E) is becoming more and more popular to train complex Deep Network architectures. An interesting question is whether this trend will continue—are there any clear failure cases for E2E training? We study this question in depth, for the specific case of E2E training an *ensemble* of networks. Our strategy is to blend the gradient smoothly in between two extremes: from independent training of the networks, up to to full E2E training. We find clear failure cases, where overparameterized models *cannot be trained E2E*. A surprising result is that the optimum can sometimes lie in between the two, neither an ensemble or an E2E system. The work also uncovers links to Dropout, and raises questions around the nature of ensemble diversity and multi-branch networks.

CLASSIFICATION

RANDOM FOREST



# Random Forest

Designed by Leo Breiman in 2001, it is a multiple classifier in which the individual classifiers are classification trees (estimators)

- Generates multiple trees but always starting from the same training set
- To grow the tree, choose at each level the feature that best separates the classes (leaves) and determine the subdivision threshold
- To classify a new pattern (test set) you visit the tree, and once you reach a leaf, you classify the pattern based on the most common class in the node among the patterns in the training set
- The decision is made by merging the classifiers (e.g. majority vote rule)

# Random Forest | Bagging

- For each node the choice of the best feature on which to partition is not made on the entire set of  $d$  features (dimensionality of the patterns), but on a random subset of  $d'$  features:

Typical value  $d' = \sqrt{d}$

- In the absence of this measure (also known as feature bagging) many trees they would choose the same variables (the most discriminating ones) with a high probability.

- At the same time the training is also not done on the entire set of  $n$  patterns, but on one random subset of  $n'$  patterns.

Typical value  $n' = 2/3 n$

Bagging = Bootstrap AGGREGatING

# Random Forest | Bagging

- Random Forest operates two types of bagging simultaneously: one on the patterns of the training set and one on features.
- Performance can be estimated with separate validation set (k-fold cross validation) or, thanks to bagging, with Out-Of-Bag (OOB) technique which does not require separate validation set.

In fact, each pattern  $x$  can be used to estimate performance starting from the trees alone in whose training  $x$  was not involved.

# Random Forest | Importance

Mean decrease in the Gini index

Measure similar to that used in decision trees: for each tree and for each variable  $i$ , each node in which the split involves that variable is evaluated, calculating the difference in the Gini index (impurity) before and after the split (mother node and child nodes), weighing it by the number of samples in the node. Then the average is taken over the trees.

The higher the values, the greater the relevance of the feature.

# Random Forest

---

**Random Forests** (input training data+labels  $T$ , number of trees  $M$ )

---

**for**  $j = 1$  to  $M$  **do**

    Take a bootstrap sample  $T'$  from  $T$

    Build a decision tree using  $T'$ , but, at every split point:

- Choose a random fraction  $K$  of the remaining features,
- Pick the best feature (minimising cost) from that subset.

    Add the tree to the set, *without pruning*

**end for**

**return** set of trees

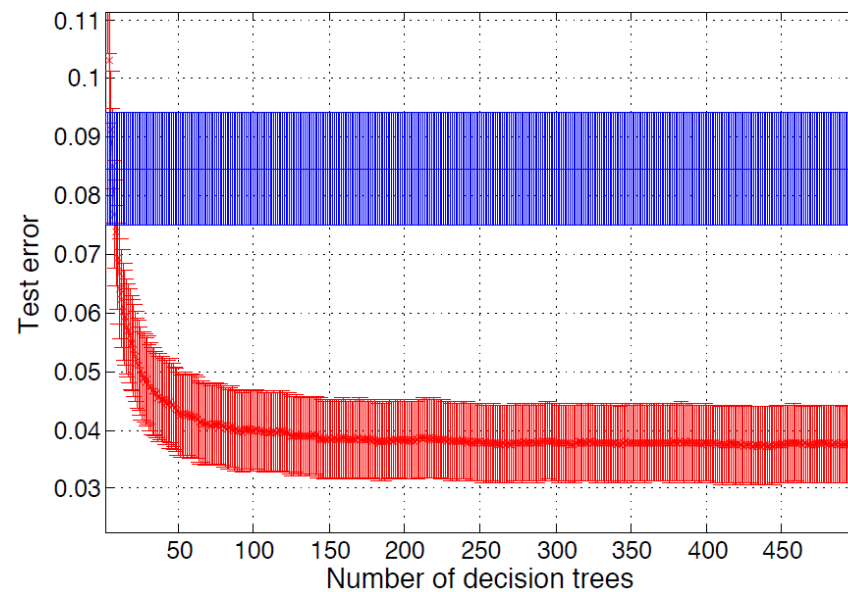
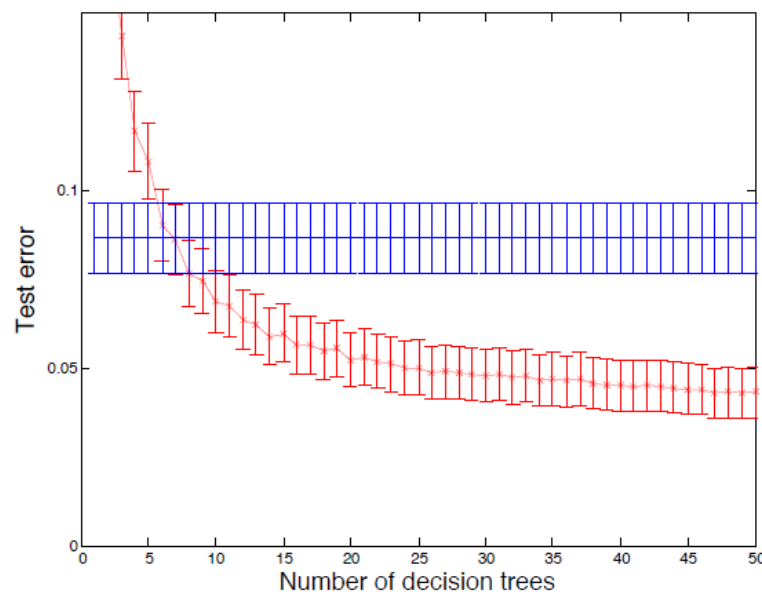
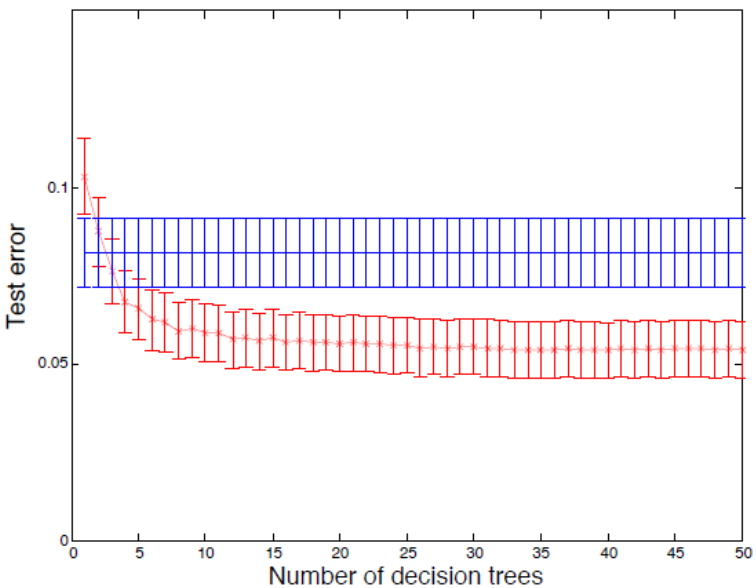
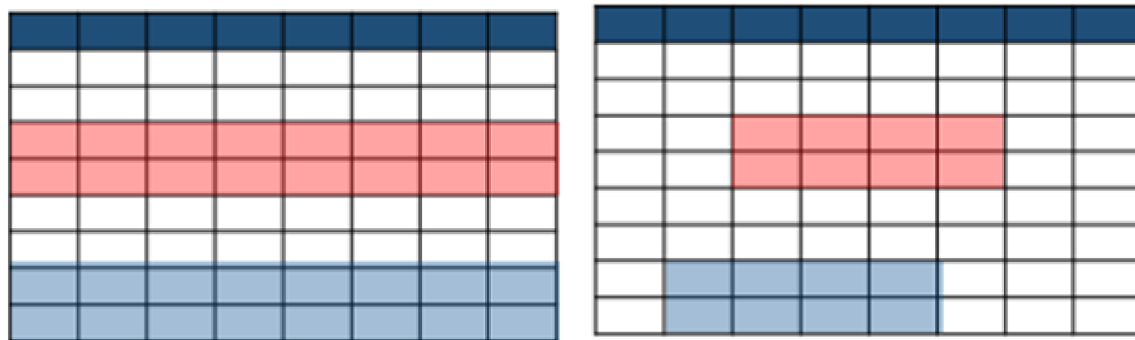
For a test point  $\mathbf{x}$ , get a response from each tree, and take a majority vote.

# Random Forest | Main features

- Improves the performance of a single decision tree learned on all data
- It loses the easy interpretability of the trees and scales less well
- It modestly increases the time complexity compared to the trees, given that each individual tree is learned only on a subset of the data and at each split only a subset of the predictors is learned taken into consideration
- It can handle even large data before offering noticeable slowdowns
- Does not present overfitting problems as the number of trees increases. This is due to the fact that only a small portion of the predictors are used at each split, other than the fact that RF predicts by means Aggregation
- It is more stable to changes in input data, thanks to bagging
- Learned trees are independent
- This makes the method easily parallelizable with appropriate hardware (e.g. multiple cores or processors)

# Random Forest vs. (simple) Bagging

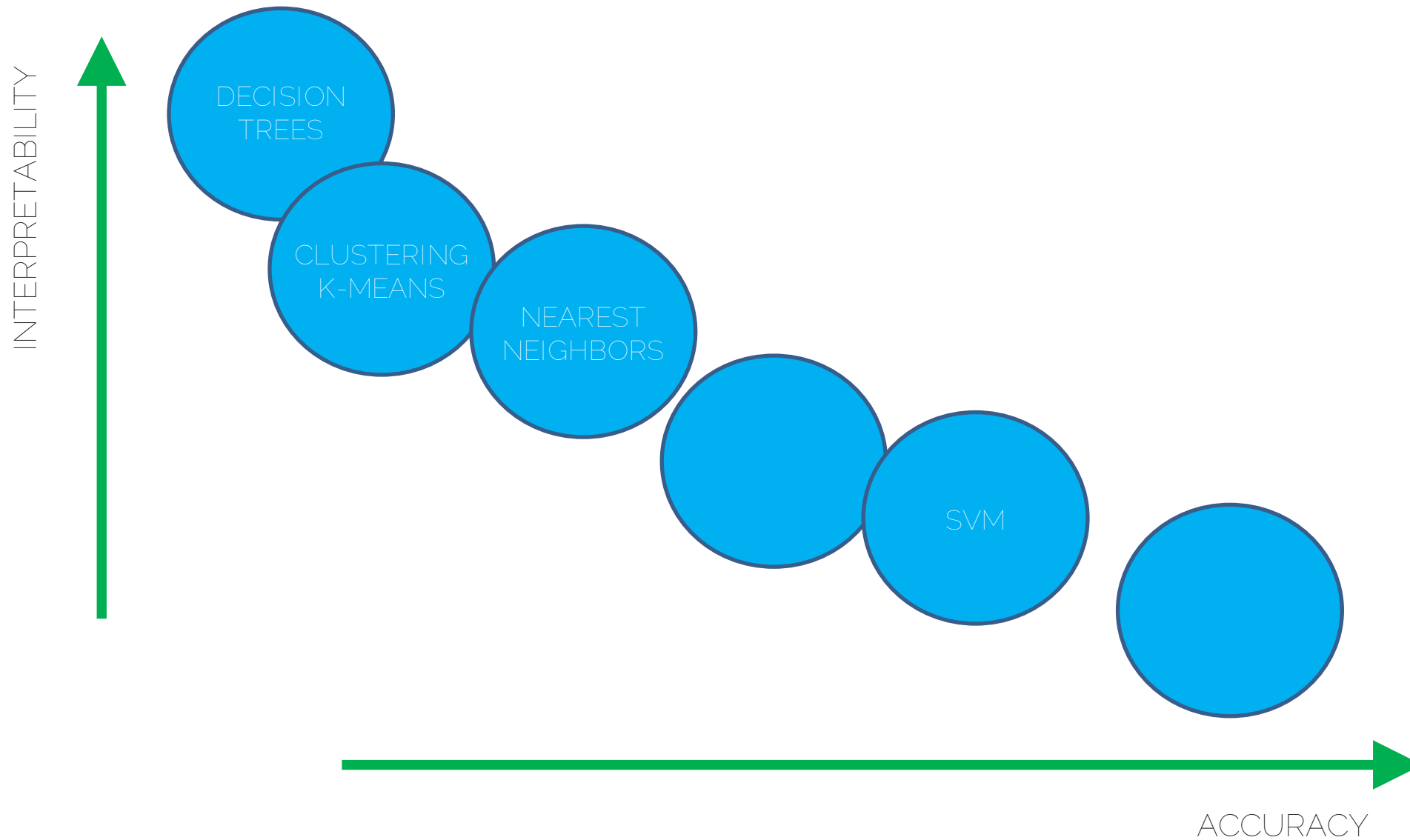
*Bagging* vs. *Random Forest*



CREDITS: PROF. GAVIN BROWN

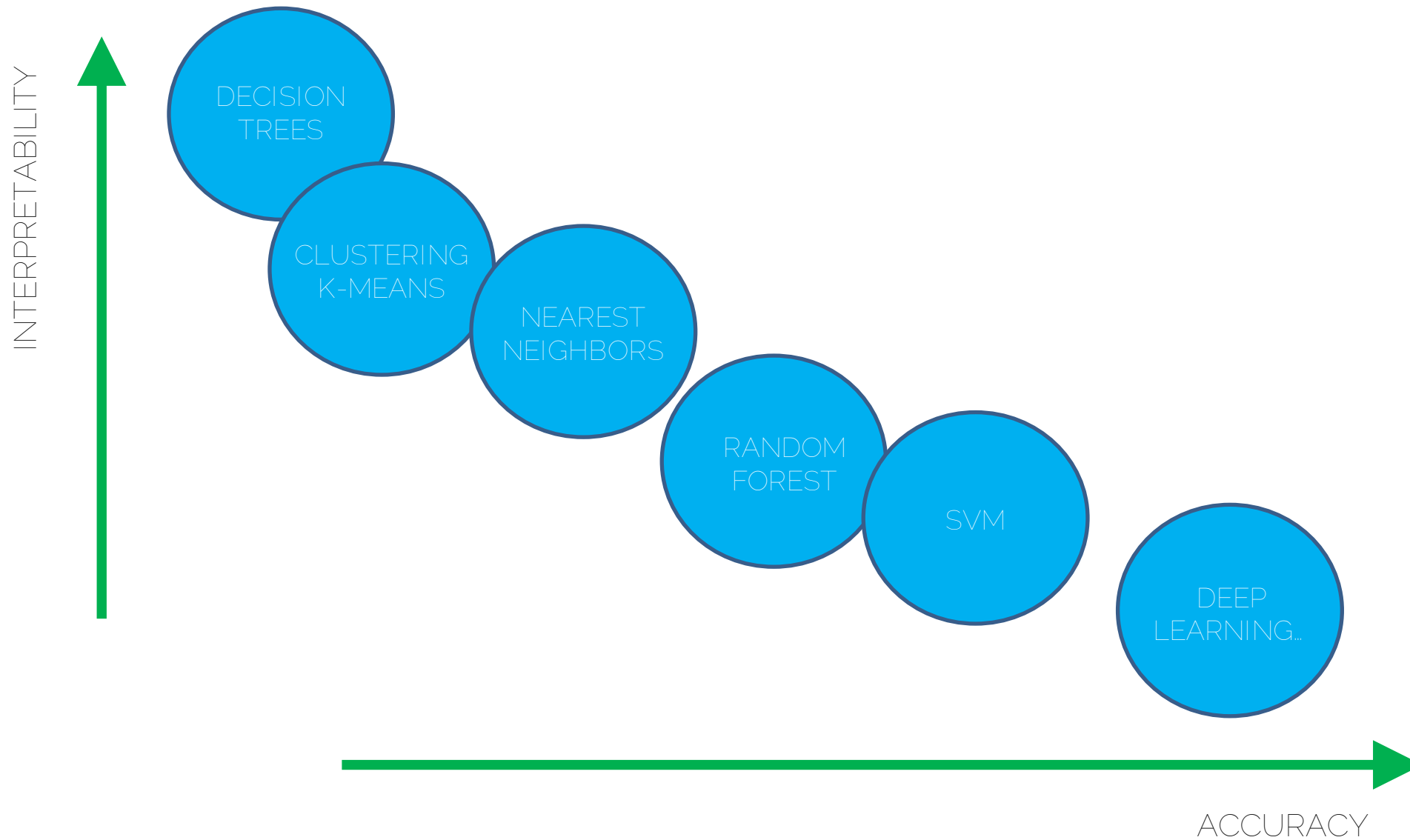
Figure 13: Bagging (LEFT) vs Random Forests (RIGHT) on the Splice dataset.

# Interpretability-Accuracy TRADEOFF

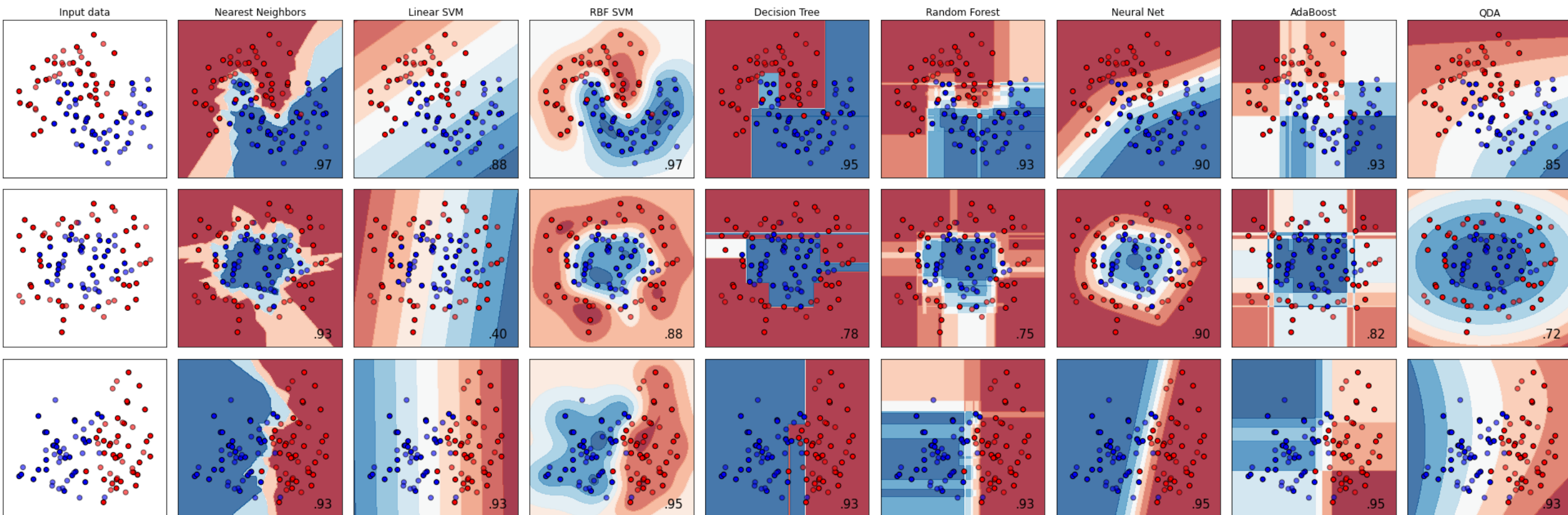




# Interpretability-Accuracy TRADEOFF



# Which is which | Decision Function





<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

## sklearn.ensemble.RandomForestClassifier

```
class sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None,
min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False,
class_weight=None, ccp_alpha=0.0, max_samples=None)
```

[source]

**class\_weight** : {"balanced", "balanced\_subsample"}, dict or list of dicts, default=None

Weights associated with classes in the form `{class_label: weight}`. If not given, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.

Note that for multioutput (including multilabel) weights should be defined for each class of every column in its own dict. For example, for four-class multilabel classification weights should be `[(0: 1, 1: 1), {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}]` instead of `[[{1:1}, {2:5}, {3:1}, {4:1}]]`.

The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`

The "balanced\_subsample" mode is the same as "balanced" except that weights are computed based on the bootstrap sample for every tree grown.

For multi-output, the weights of each column of y will be multiplied.

Note that these weights will be multiplied with sample\_weight (passed through the fit method) if sample\_weight is specified.

## Methods

<code>apply(X)</code>	Apply trees in the forest to X, return leaf indices.
<code>decision_path(X)</code>	Return the decision path in the forest.
<code>fit(X, y[, sample_weight])</code>	Build a forest of trees from the training set (X, y).
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>predict(X)</code>	Predict class for X.
<code>predict_log_proba(X)</code>	Predict class log-probabilities for X.
<code>predict_proba(X)</code>	Predict class probabilities for X.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.



<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html>  
<https://scikit-learn.org/stable/modules/ensemble.html#adaboost>

## sklearn.ensemble.AdaBoostClassifier

```
class sklearn.ensemble.AdaBoostClassifier(estimator=None, *, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R',
random_state=None, base_estimator='deprecated')
```

[source]

**class\_weight** : {"balanced", "balanced\_subsample", dict or list of dicts, default=None

Weights associated with classes in the form `{class_label: weight}`. If not given, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.

Note that for multioutput (including multilabel) weights should be defined for each class of every column in its own dict. For example, for four-class multilabel classification weights should be `[(0: 1, 1: 1), {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}]` instead of `[[{1:1}, {2:5}, {3:1}, {4:1}]]`.

The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as `n_samples / (n_classes * np.bincount(y))`

The "balanced\_subsample" mode is the same as "balanced" except that weights are computed based on the bootstrap sample for every tree grown.

For multi-output, the weights of each column of y will be multiplied.

Note that these weights will be multiplied with sample\_weight (passed through the fit method) if sample\_weight is specified.

### Methods

<code>decision_function(X)</code>	Compute the decision function of x.
<code>fit(X, y[, sample_weight])</code>	Build a boosted classifier/regressor from the training set (X, y).
<code>get_params([deep])</code>	Get parameters for this estimator.
<code>predict(X)</code>	Predict classes for X.
<code>predict_log_proba(X)</code>	Predict class log-probabilities for X.
<code>predict_proba(X)</code>	Predict class probabilities for X.
<code>score(X, y[, sample_weight])</code>	Return the mean accuracy on the given test data and labels.
<code>set_params(**params)</code>	Set the parameters of this estimator.
<code>staged_decision_function(X)</code>	Compute decision function of x for each boosting iteration.
<code>staged_predict(X)</code>	Return staged predictions for X.
<code>staged_predict_proba(X)</code>	Predict class probabilities for X.
<code>staged_score(X, y[, sample_weight])</code>	Return staged scores for X, y.