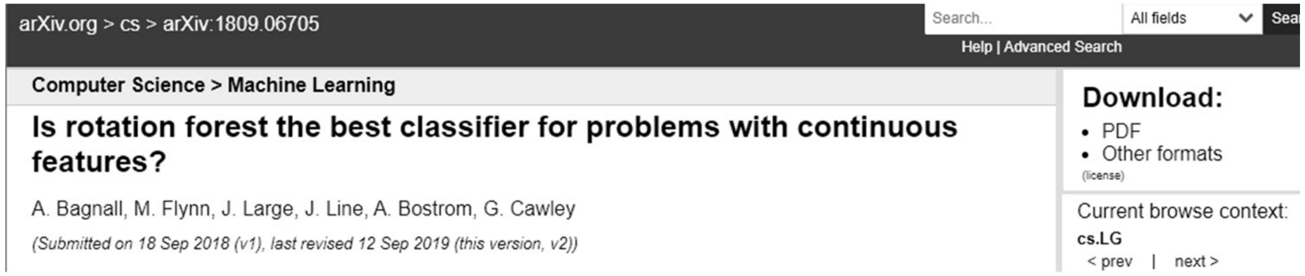
Machine Learning
Classifier Ensembles

- Ensemble types: homeogeneous vs heterogeneous
- Weighting schemes
- Ensemble schemes
  - Bagging
  - Boosting
  - Stacking
  - Classifiers:
    - Random Forest/Rotation Forest
    - Adaboost/XGBoost



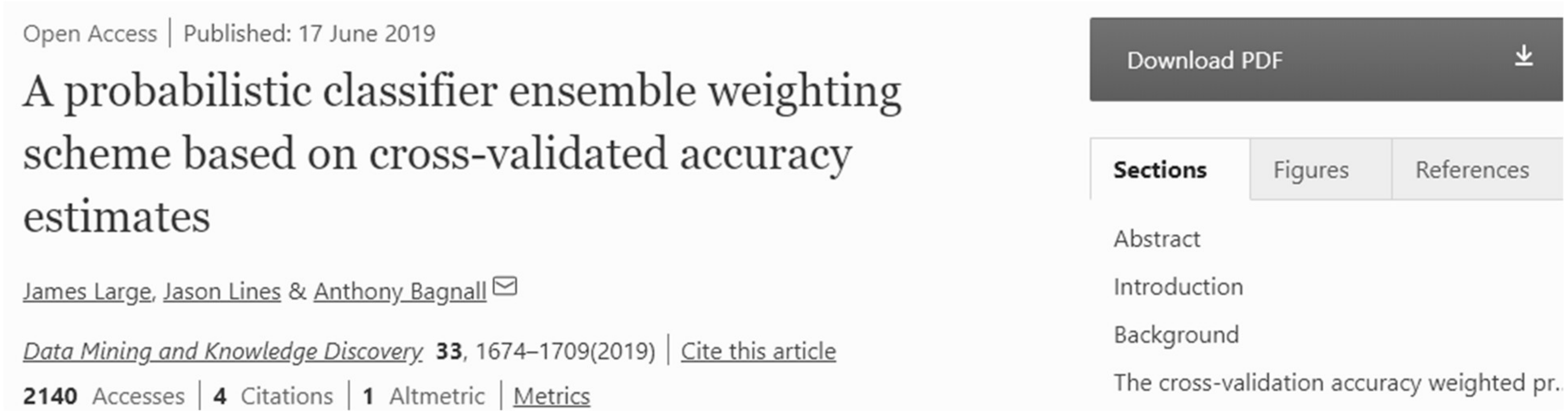
arXiv.org > cs > arXiv:1809.06705

Computer Science > Machine Learning

Is rotation forest the best classifier for problems with continuous features?

A. Bagnall, M. Flynn, J. Large, J. Line, A. Bostrom, G. Cawley

(Submitted on 18 Sep 2018 (v1), last revised 12 Sep 2019 (this version, v2))



Open Access | Published: 17 June 2019

A probabilistic classifier ensemble weighting scheme based on cross-validated accuracy estimates

James Large, Jason Lines & Anthony Bagnall

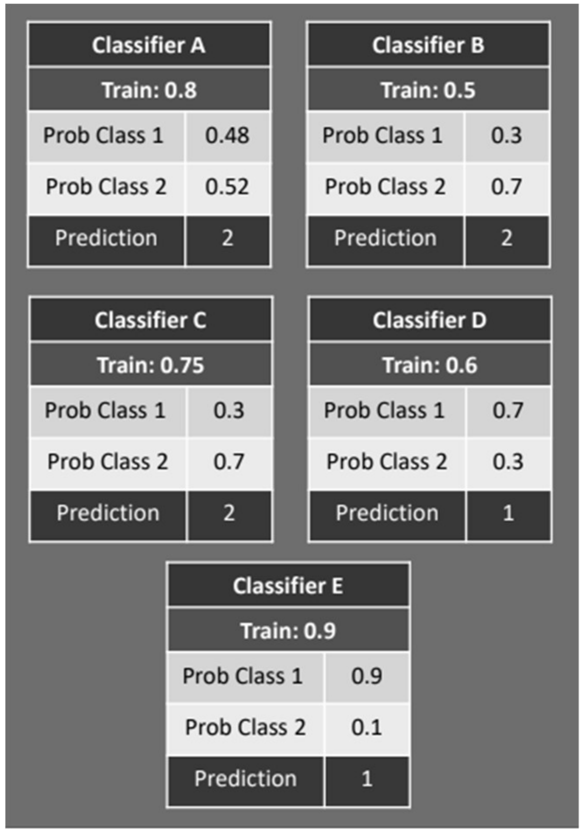
Data Mining and Knowledge Discovery **33**, 1674–1709(2019) | Cite this article

2140 Accesses | **4** Citations | **1** Altmetric | Metrics

Ensembles

Different classifiers make different kind of mistakes

Ensemble schemes are algorithms that combine the output of multiple base classifiers



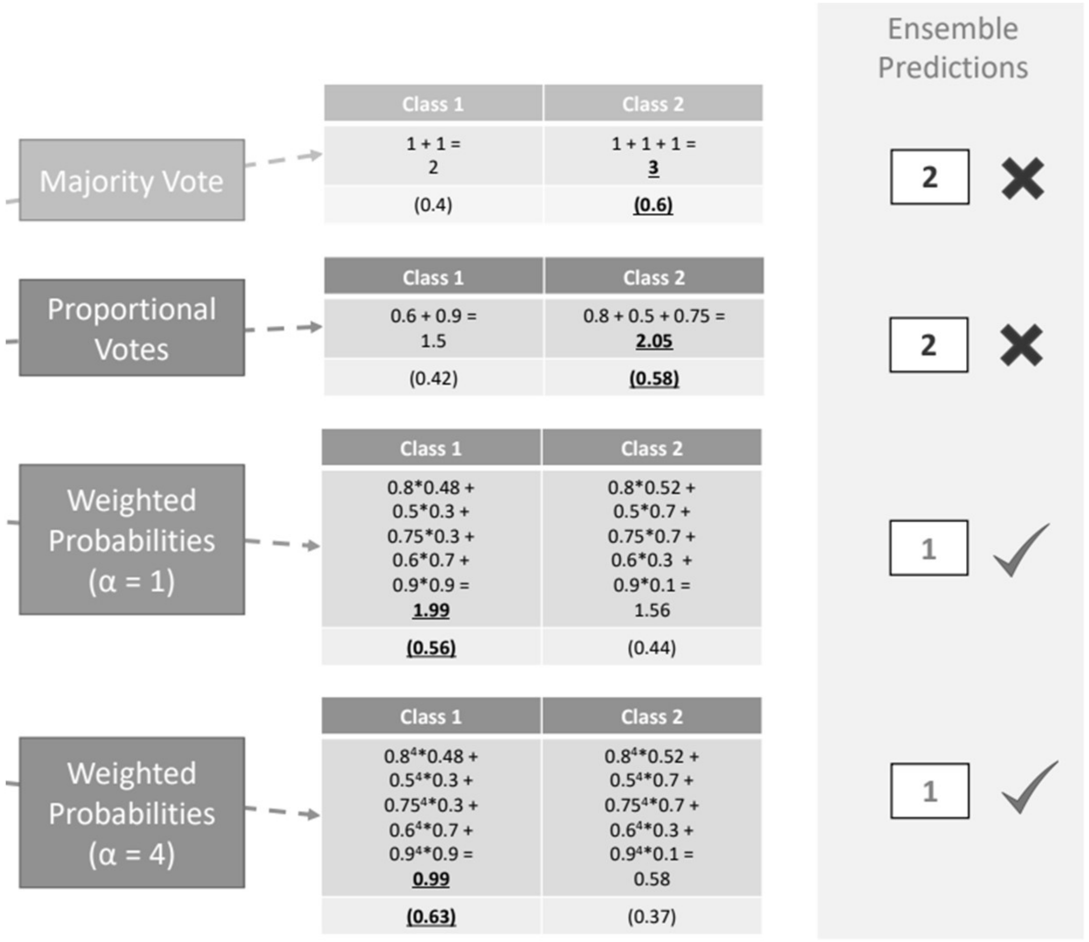
Classifier A	
Train: 0.8	
Prob Class 1	0.48
Prob Class 2	0.52
Prediction	2

Classifier B	
Train: 0.5	
Prob Class 1	0.3
Prob Class 2	0.7
Prediction	2

Classifier C	
Train: 0.75	
Prob Class 1	0.3
Prob Class 2	0.7
Prediction	2

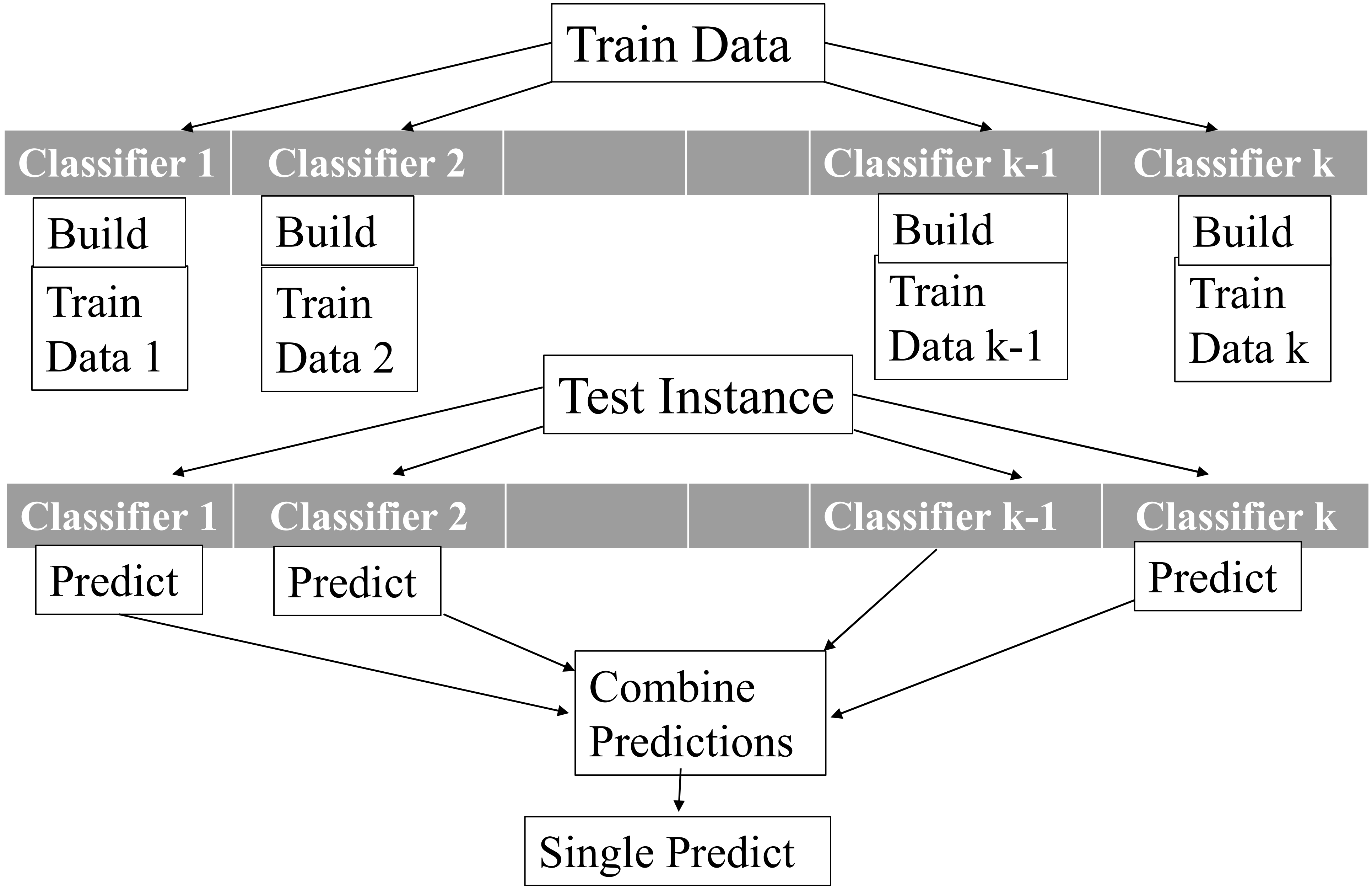
Classifier D	
Train: 0.6	
Prob Class 1	0.7
Prob Class 2	0.3
Prediction	1

Classifier E	
Train: 0.9	
Prob Class 1	0.9
Prob Class 2	0.1
Prediction	1



	Class 1	Class 2	Ensemble Predictions
Majority Vote	1 + 1 = 2 (0.4)	1 + 1 + 1 = **3** **(0.6)**	2 ✗
Proportional Votes	0.6 + 0.9 = 1.5 (0.42)	0.8 + 0.5 + 0.75 = **2.05** **(0.58)**	2 ✗
Weighted Probabilities (α = 1)	0.8*0.48 + 0.5*0.3 + 0.75*0.3 + 0.6*0.7 + 0.9*0.9 = **1.99** **(0.56)**	0.8*0.52 + 0.5*0.7 + 0.75*0.7 + 0.6*0.3 + 0.9*0.1 = 1.56 (0.44)	1 ✓
Weighted Probabilities (α = 4)	0.8^4*0.48 + 0.5^4*0.3 + 0.75^4*0.3 + 0.6^4*0.7 + 0.9^4*0.9 = **0.99** **(0.63)**	0.8^4*0.52 + 0.5^4*0.7 + 0.75^4*0.7 + 0.6^4*0.3 + 0.9^4*0.1 = 0.58 (0.37)	1 ✓

Ensemble Model



Structural Design Choices

1. Which classifiers to use and how many?
    - Copies of the same classifier (**homogenious**)
    - Different classification algorithms (**heterogeneous**)
2. Which data set to train each classifier on?
    - same data/sample cases/sample attributes
3. How to combine predictions?

- voting scheme/weighting scheme

Draw Overview Diagram HERE

Which classifiers to use and how many?

There is no point combining classifiers that all produce the same predictions

Conversely, if we add too many bad classifiers, performance overall will degrade

We seek **diversity** without compromising **quality**

COMMITTEE
THE MORE PEOPLE INVOLVED IN ANY PROJECT,
THE SLOWER AND STUPIDER THAT PROJECT TENDS TO BE.
WWW.ISHKUR.COM

Homogeneous ensembles

Bagging, random subspace and boosting

Random Forest, Rotation Forest, AdaBoost and XGBoost

Diversity through sampling

1. Bootstrap aggregation, or **bagging** [1]

- For each classifier, sample the training data with replacement (bootstrap sampling).
- This means there will be duplicates in each training data.
- Given a training set of size *n*, if we bootstrap sample *n* times, each new training set will have approximately 63% of the cases

2. **Random subspace methods**[2], or attribute sampling

- For each classifier, select a proportion of attributes to include

Random Forest and Rotation Forest

1. L. Breiman, "Bagging predictors". *Machine Learning*. **24** (2): 123–140, 1994

2. T. K. Ho, "The Random Subspace Method for Constructing Decision Forests". *IEEE Transactions on Pattern Analysis and Machine Intelligence*. **20** (8): 832–844,1998

3. L. Breiman, “Random Forests". *Machine Learning*. **45** (1): 5–32, 2001

Random Forest[3]

- Random Forest is probably the most popular machine learning classification algorithm (after deep learning)
- It combines bagging and random subspace sampling for each base classifier
- It uses a **`RandomTree`** decision tree base classifier.

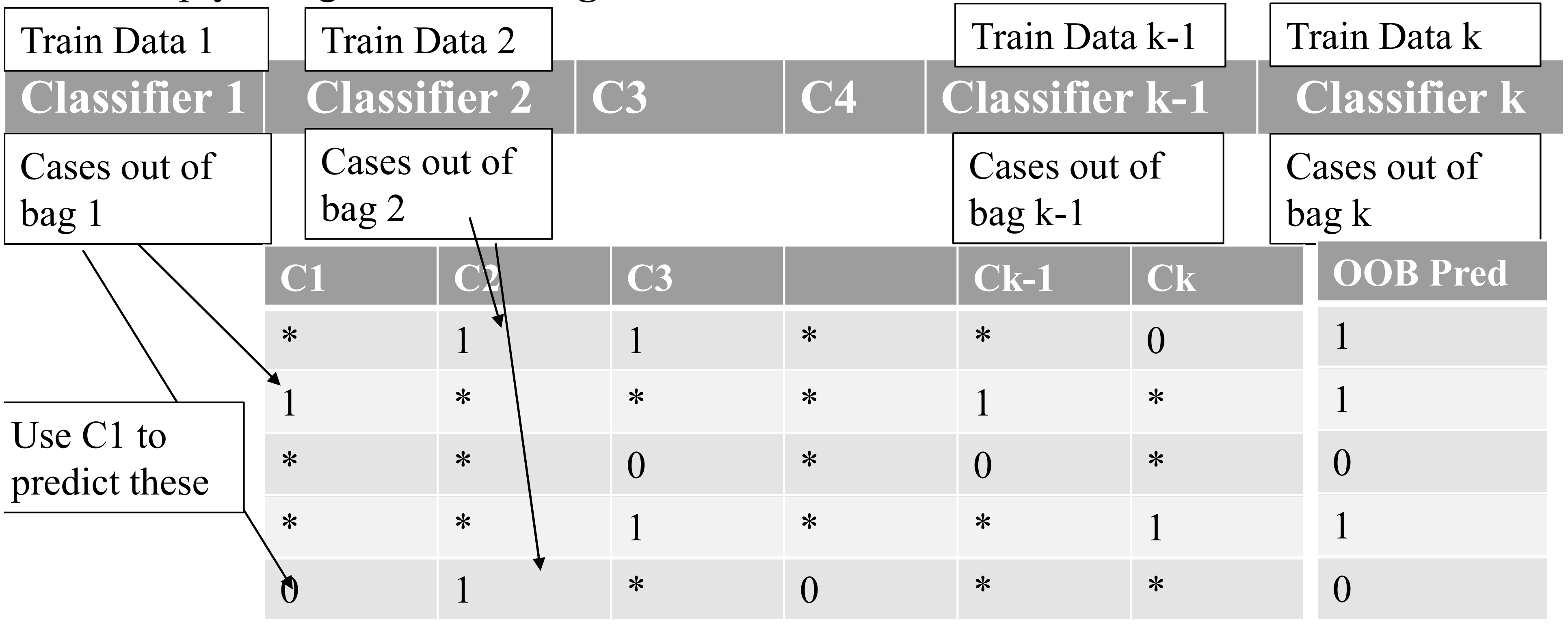
Stopping Criteria	Stops if a node is pure or max depth is reached, or no selected attribute gives an information gain
How to Split the Data	Different node for discrete attributes, binary split for continuous
Choosing an Attribute	**Randomly selects at most *k* attributes until one is found with positive information gain (not gain ratio)**

Random Forest Details

Random Forest has the following parameters

1. Bag percent size (number of samples for each): defaults to 100
2. Number of trees: usually defaults to 500
3. Number of features: usually defaults to sqrt(total number of features)
4. Maximum tree depth: usually defaults to number of features

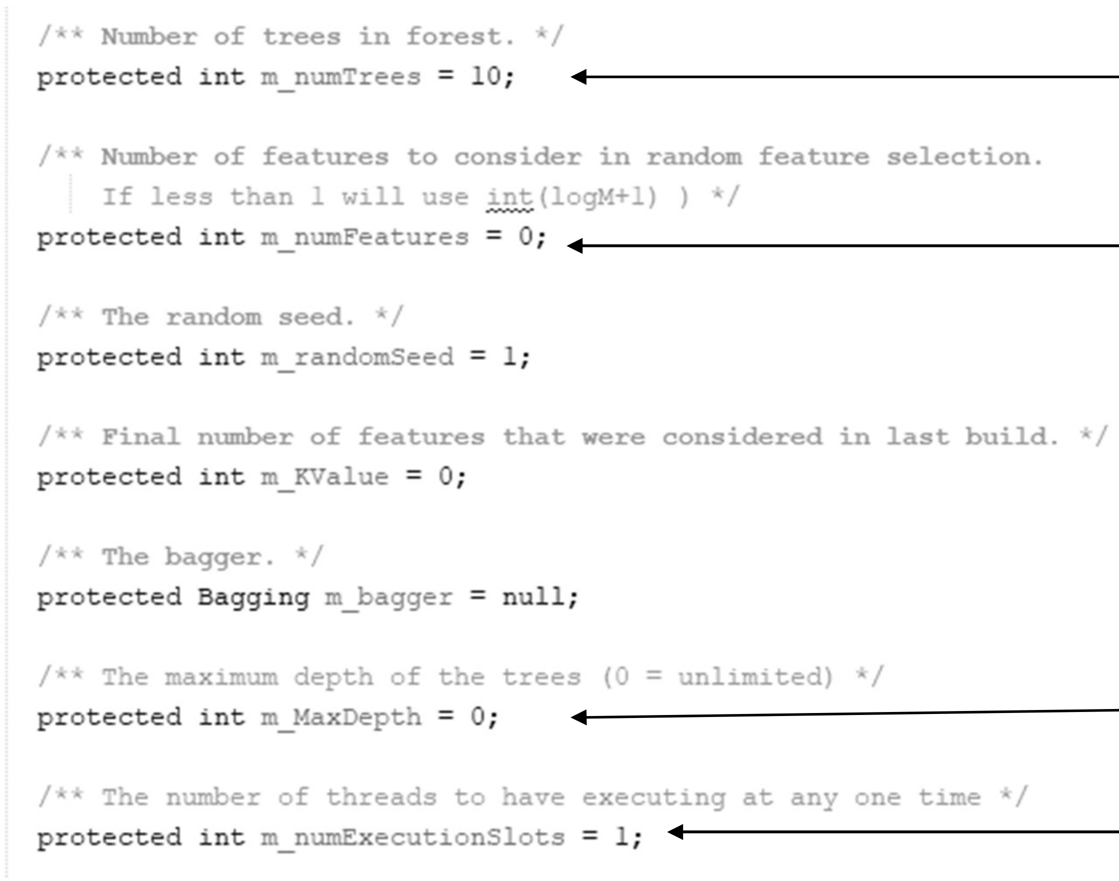
One benefit of Random Forests is we can assess the quality of the classifier cheaply using the **out of bag errror**



C1	C2	C3		Ck-1	Ck	OOB Pred
*	1	1	*	*	0	1
1	*	*	*	1	*	1
*	*	0	*	0	*	0
*	*	1	*	*	1	1
0	1	*	0	*	*	0

```
package weka.classifiers.trees;
```

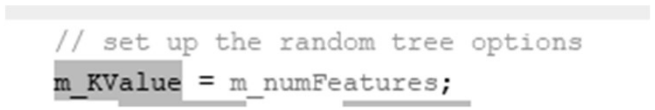
Random Forest in Weka



```
/** Number of trees in forest. */
protected int m_numTrees = 10;

/** Number of features to consider in random feature selection.
    If less than 1 will use int(logM+1) ) */
protected int m_numFeatures = 0;

/** The random seed. */
protected int m_randomSeed = 1;

/** Final number of features that were considered in last build. */
protected int m_KValue = 0;

/** The bagger. */
protected Bagging m_bagger = null;

/** The maximum depth of the trees (0 = unlimited) */
protected int m_MaxDepth = 0;

/** The number of threads to have executing at any one time */
protected int m_numExecutionSlots = 1;
```

Very small default number of trees

This is worse than sqrt(m)

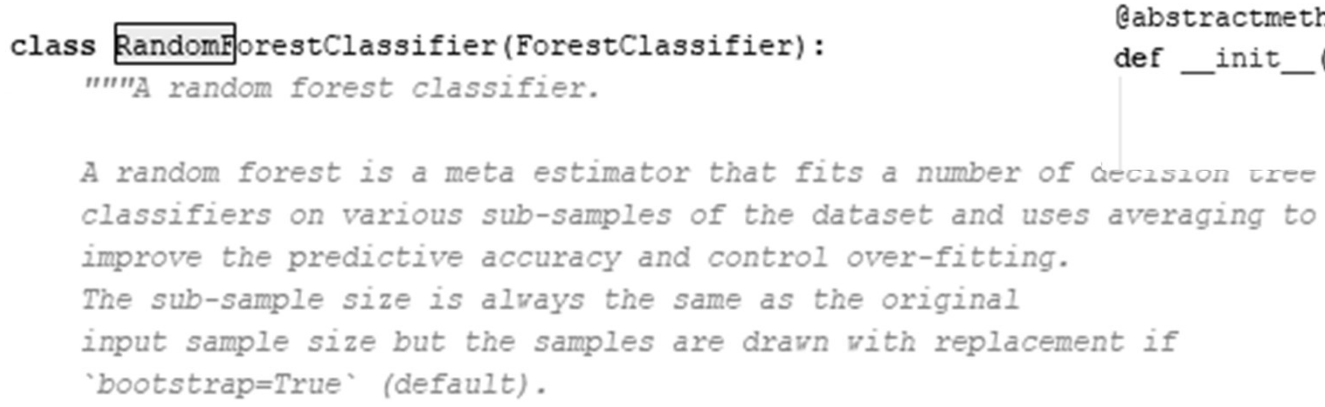


```
// set up the random tree options
m_KValue = m_numFeatures;
```

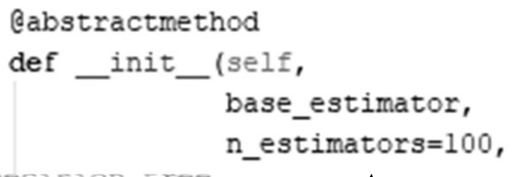
Normal to leave as 0

You can thread it

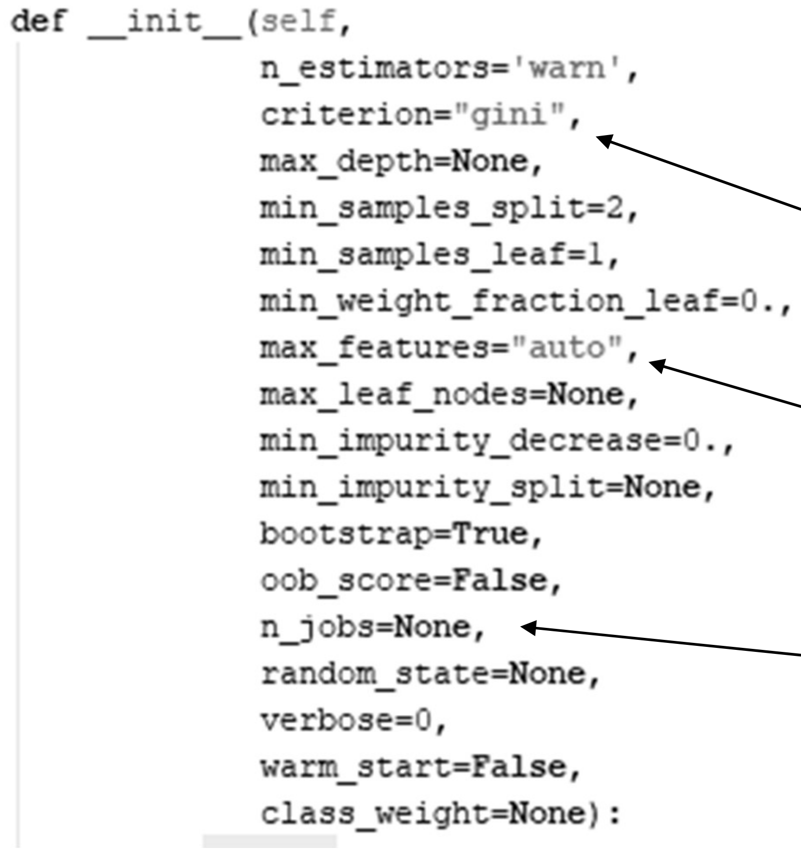
Random Forest in scikit



```
class RandomForestClassifier(ForestClassifier):
    """A random forest classifier.

    A random forest is a meta estimator that fits a number of decision tree
    classifiers on various sub-samples of the dataset and uses averaging to
    improve the predictive accuracy and control over-fitting.
    The sub-sample size is always the same as the original
    input sample size but the samples are drawn with replacement if
    `bootstrap=True` (default).
```



```
@abstractmethod
def __init__(self,
             base_estimator,
             n_estimators=100,
```



```
def __init__(self,
             n_estimators='warn',
             criterion="gini",
             max_depth=None,
             min_samples_split=2,
             min_samples_leaf=1,
             min_weight_fraction_leaf=0.,
             max_features="auto",
             max_leaf_nodes=None,
             min_impurity_decrease=0.,
             min_impurity_split=None,
             bootstrap=True,
             oob_score=False,
             n_jobs=None,
             random_state=None,
             verbose=0,
             warm_start=False,
             class_weight=None):
```

100 default number of trees

Uses gini by default (not entropy)

Defaults to sqrt(m)

You can thread it

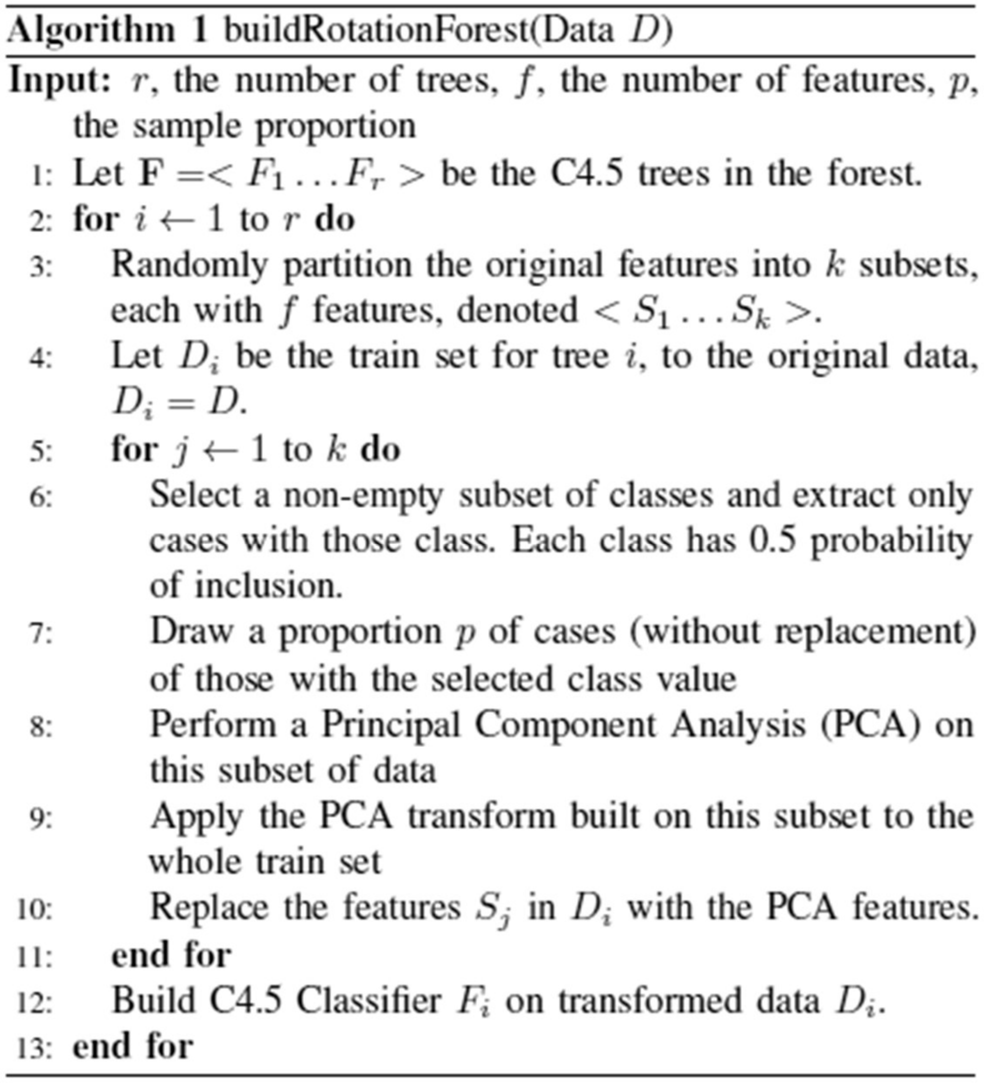
Rotation Forest [4]

Rotation forest is similar to random forest with the following key differences

1. It performs a different form of bagging and random subspace sampling, then transforms the samples using a **principle component analysis**

2. It uses C4.5 as a base classifier

You don't need to remember it in this detail



Algorithm 1 buildRotationForest(Data D)

```
Input: $r$, the number of trees, $f$, the number of features, $p$, the sample proportion
1: Let $\mathbf{F} = < F_1 \ldots F_r >$ be the C4.5 trees in the forest.
2: for $i \leftarrow 1$ to $r$ do
3:     Randomly partition the original features into $k$ subsets, each with $f$ features, denoted $< S_1 \ldots S_k >$.
4:     Let $D_i$ be the train set for tree $i$, to the original data, $D_i = D$.
5:     for $j \leftarrow 1$ to $k$ do
6:         Select a non-empty subset of classes and extract only cases with those class. Each class has 0.5 probability of inclusion.
7:         Draw a proportion $p$ of cases (without replacement) of those with the selected class value
8:         Perform a Principal Component Analysis (PCA) on this subset of data
9:         Apply the PCA transform built on this subset to the whole train set
10:        Replace the features $S_j$ in $D_i$ with the PCA features.
11:    end for
12:    Build C4.5 Classifier $F_i$ on transformed data $D_i$.
13: end for
```

4. Rodríguez JJ, Kuncheva LI, Alonso CJ, "**Rotation forest: A new classifier ensemble method**". *IEEE Transactions on Pattern Analysis and Machine Intelligence*. **28** (10): 1619–1630,2006

Diversity through sampling

3. **Boosting**[5]

- Train the first classifier on the full training data.
- Find out which cases the classifier got wrong
- For the next classifier, somehow give the wrong cases greater importance, i.e. **boost them**
- The boosting can involve resampling with increased probability, or it may be possible to embed the weights in the learning algorithm
- Boosting can also involve weighting the classifier voting
- There are many variants, these are the most popular

AdaBoost, LogitBoost and XGBoost

5. Schapire, Robert E. (1990). "The Strength of Weak Learnability". *Machine Learning*. **5** (2): 197–227

Basic Boosting

```
Algorithm 3 buildBooster(Data $D =< (\mathbf{x_i}, y_1)...(\mathbf{x_n}, y_n))$
Require: $k$, the number of classifiers
1: Let $\mathbf{H} \leftarrow < h_1 \ldots h_k >$ be the base classifiers
2: Let $\mathbf{W} \leftarrow < w_1 \ldots w_n >$ be the weights
3: for $i \leftarrow 1$ to $n$ do
4:     $w_i \leftarrow 1/n$
5: for $i \leftarrow 1$ to $k$ do
6:     buildClassifier($D, h_i, \mathbf{W}$)
7:     updateWeights($D, h_i, \mathbf{W}$)
```

- The principle is that the classifier can use the weights to alter the model it builds and that the weights are updated to give greater weight to cases the previous classifier misclassified
- One way of doing this is using the weights to sample the data

Adaptive Boosting: AdaBoost[6]

```
Algorithm 4 buildAdaBoost(Data $D =< (\mathbf{x_i}, y_1)...(\mathbf{x_n}, y_n))$
Require: $k$, the number of classifiers
 1: Let $\mathbf{H} \leftarrow < h_1 \dots h_k >$ be the base classifiers
 2: Let $\mathbf{W} \leftarrow < w_1 \dots w_n >$ be the weights
 3: for $i \leftarrow 1$ to $n$ do
 4:     $w_i \leftarrow 1/n$
 5: for $i \leftarrow 1$ to $k$ do
 6:     buildClassifier$(D, h_i, \mathbf{W})$
 7:     Let $\mathbf{p} \leftarrow < p_1 \dots p_n >$ be the predicted classes for the train data for $h_i$.
 8:     $e_i \leftarrow$ trainError$(\mathbf{y}, \mathbf{p})$
 9:     $\alpha_i \leftarrow \frac{1}{2} \log \frac{1-e_i}{e_i}$
10:     for $i \leftarrow 1$ to $n$ do
11:         $w_i = w_i \times \begin{cases} e^{-\alpha_i} & \text{if } p_i = y_i \\ e^{\alpha_i}, & \text{otherwise} \end{cases}$
12:     $s_0 = 0$
13:     for $i \leftarrow 1$ to $n$ do
14:         $s_0 = s_0 + w_i$
15:     $s_0 = s_0/n$
16:     for $i \leftarrow 1$ to $n$ do
17:         $w_i = \frac{w_i}{s}$
```

Initialise weights equally

Build base classifier using weights and get predictions on train data

Get the overall classifier weight

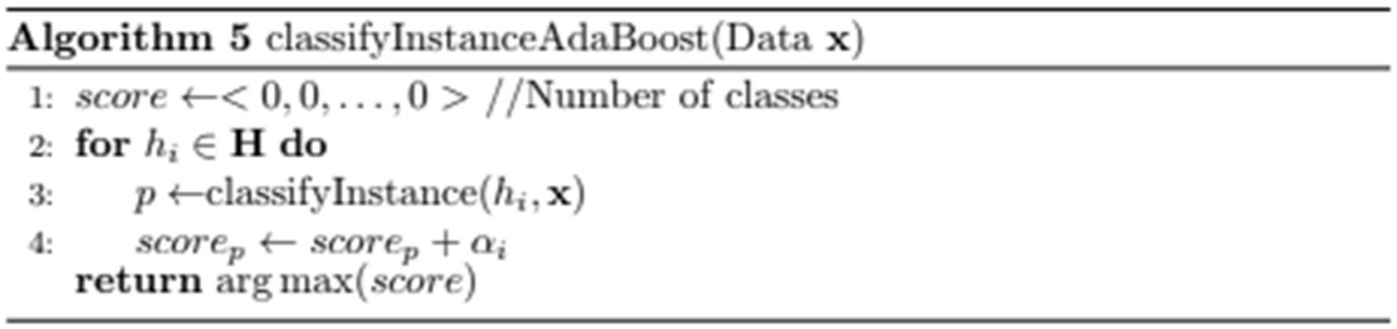
Tweak case weight down if correctly classified or up if misclassified

Normalise so weights add up to one

6. Freund and Schapire: Experiments with a new boosting algorithm. In: Thirteenth International Conference on Machine Learning, San Francisco, 148-156, 1996.

AdaBoost Predictions

New cases are classified by a **weighted majority vote**



Algorithm 5 classifyInstanceAdaBoost(Data $\mathbf{x}$)

```
1: $score \leftarrow < 0, 0, \ldots, 0 >$ //Number of classes
2: for $h_i \in \mathbf{H}$ do
3:     $p \leftarrow$ classifyInstance($h_i, \mathbf{x}$)
4:     $score_p \leftarrow score_p + \alpha_i$
   return $\arg\max(score)$
```

- Each ensemble member is given weight equal to alpha.

Weights can be used by the base classifier in two ways

1. Internally, to adjust the way the classifier builds the model
2. Externally, by using them to sample through bagging

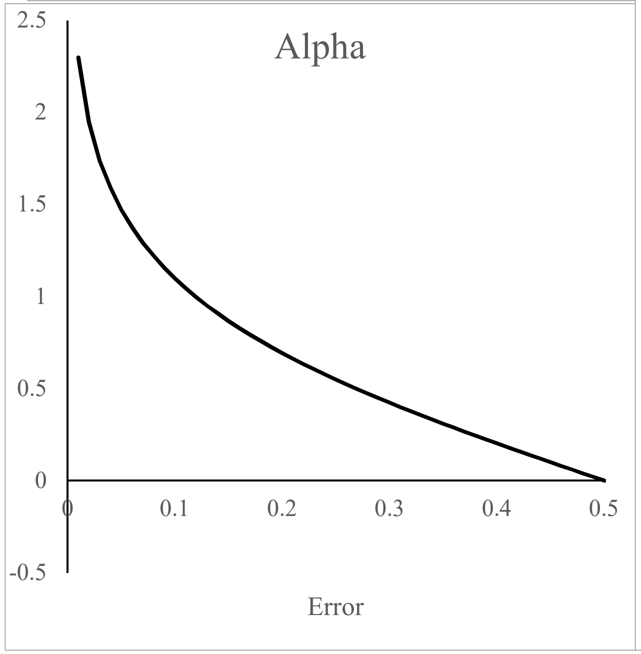
AdaBoost Details

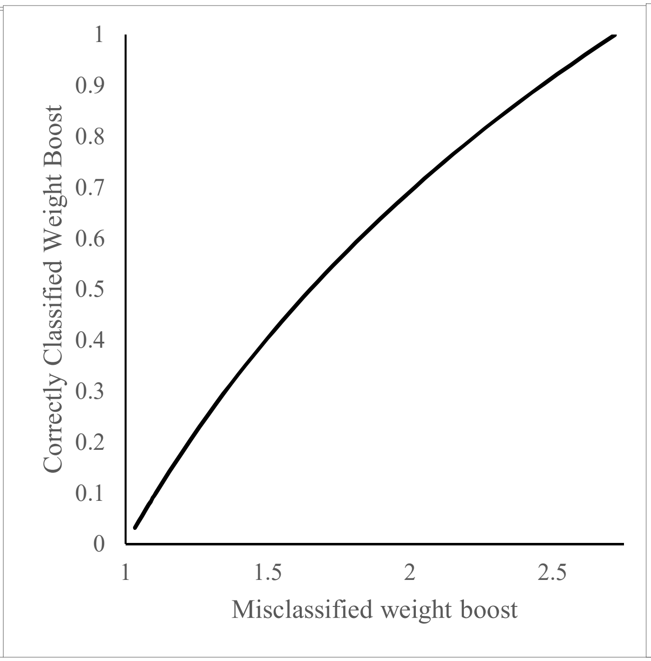
- The base classifier in the original work and the Weka implementation is a **Decision Stump**

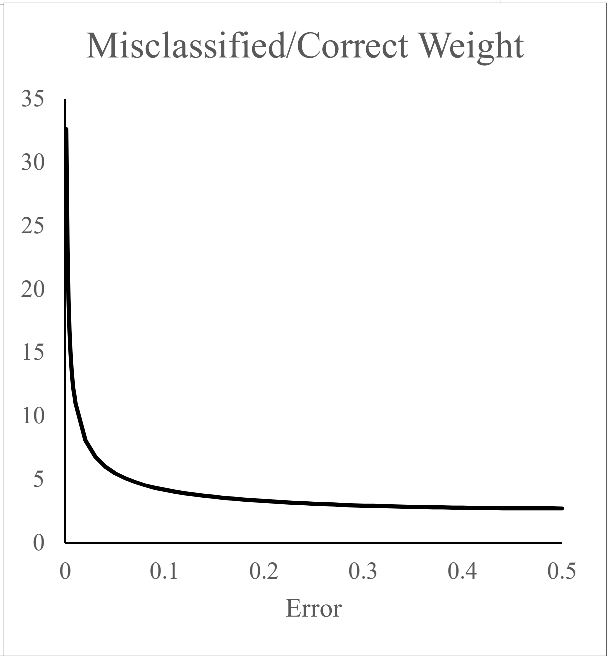
- A decision stump is a 1 level classification tree that uses information gain

- Weights can be used by the base classifier in two ways

1. Internally, to adjust the way the classifier builds the model
2. Externally, by using them to sample through bagging (default with stump)







Other Popular Boosting Algorithms

Logistic Boosting: LogitBoost

- Use a decision stump like AdaBoost
- Adjusts the weights after each new model using and adaption of Logistic Regression
- Always does 1 vs All models

Extreme Gradient Boost: XGBoost

- Currently popular because of Kaggle success
- Uses CART trees as base models
- Uses a greedy algorithm to create an additive model, each iteration tries to correct the errors of the previous
- Uses a penalty function (**regularisation**) to avoid overfitting the train data

Heterogeneous ensembles

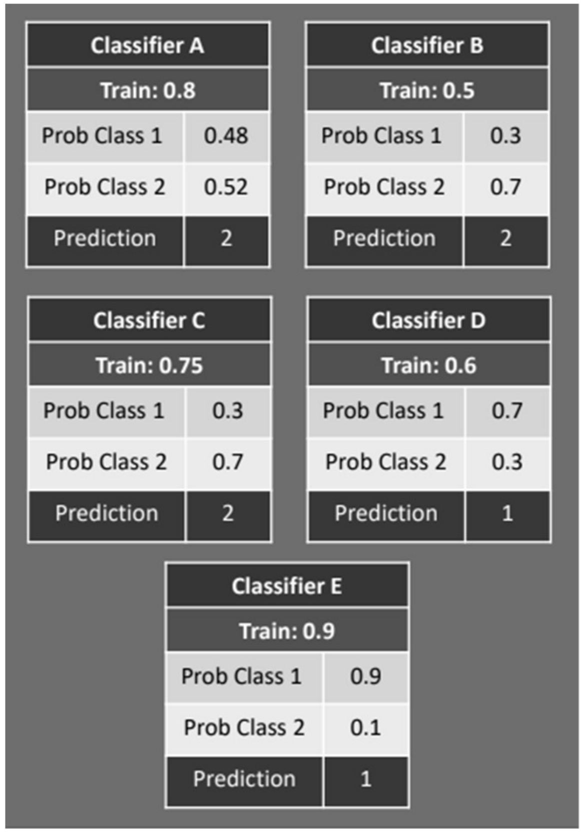
Weighting Schemes: majority vote, weighted majority vote, weighted probabilities (CAWPE)
Selection Schemes: overproduce and select
Stacking: build a model on the output of the base models

Ignore the base classifiers, address the issue of how to combine the classifier outputs to improve performance

Assume we have *k* classifiers

Weighting Schemes

Assume *k* base classifiers. For a new case they produce *k* predictions and *k* probability distributions



Classifier A	
Train: 0.8	
Prob Class 1	0.48
Prob Class 2	0.52
Prediction	2

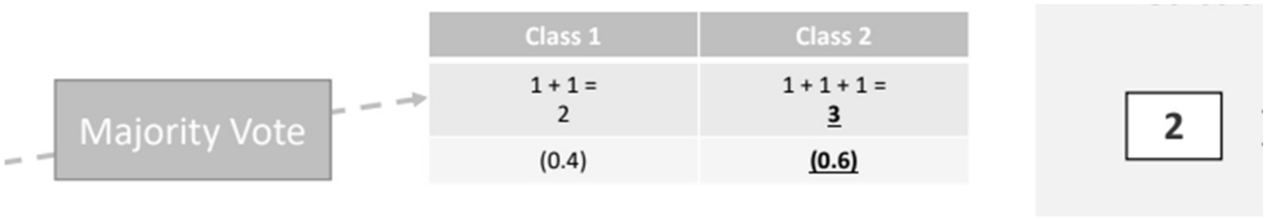
Classifier B	
Train: 0.5	
Prob Class 1	0.3
Prob Class 2	0.7
Prediction	2

Classifier C	
Train: 0.75	
Prob Class 1	0.3
Prob Class 2	0.7
Prediction	2

Classifier D	
Train: 0.6	
Prob Class 1	0.7
Prob Class 2	0.3
Prediction	1

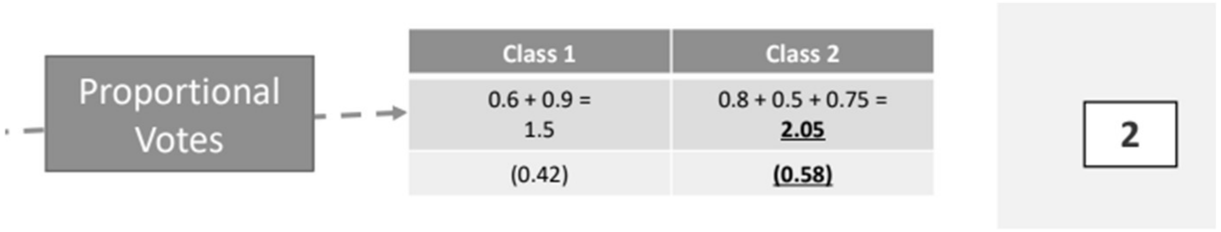
Classifier E	
Train: 0.9	
Prob Class 1	0.9
Prob Class 2	0.1
Prediction	1

Majority voting simply counts how many vote for each class



Class 1	Class 2
1 + 1 = 2	1 + 1 + 1 = **3**
(0.4)	**(0.6)**

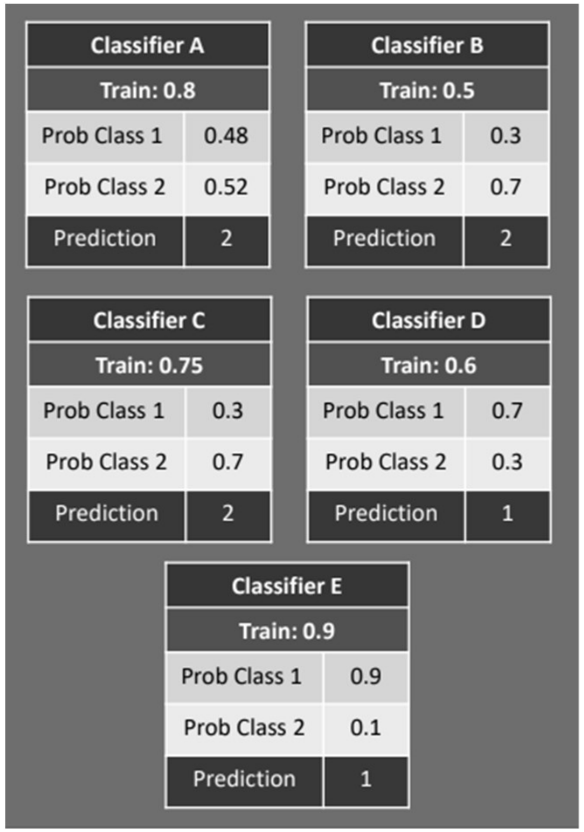
Weighted majority voting gives each classifier a weight formed on the train data (we estimate accuracy with 10-fold cross validation



Class 1	Class 2
0.6 + 0.9 = 1.5	0.8 + 0.5 + 0.75 = **2.05**
(0.42)	**(0.58)**

Weighting Schemes

Weights based on an estimate of the test accuracy found on the train data through cross validation or bagging



Classifier A	
Train: 0.8	
Prob Class 1	0.48
Prob Class 2	0.52
Prediction	2

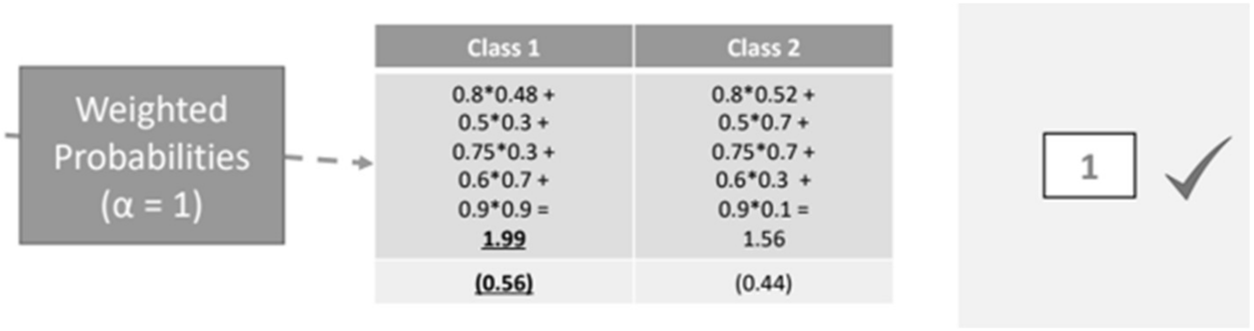
Classifier B	
Train: 0.5	
Prob Class 1	0.3
Prob Class 2	0.7
Prediction	2

Classifier C	
Train: 0.75	
Prob Class 1	0.3
Prob Class 2	0.7
Prediction	2

Classifier D	
Train: 0.6	
Prob Class 1	0.7
Prob Class 2	0.3
Prediction	1

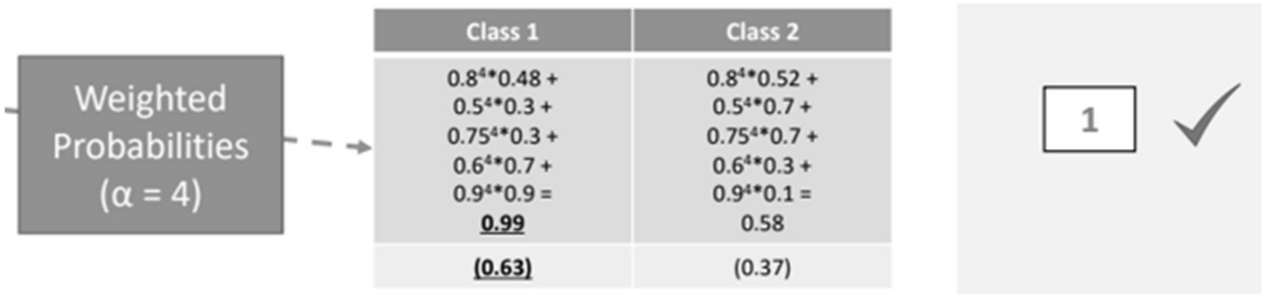
Classifier E	
Train: 0.9	
Prob Class 1	0.9
Prob Class 2	0.1
Prediction	1

Probability weighting applies the weights to the probabilities, not the predictions



Class 1	Class 2
0.8*0.48 + 0.5*0.3 + 0.75*0.3 + 0.6*0.7 + 0.9*0.9 = **1.99**	0.8*0.52 + 0.5*0.7 + 0.75*0.7 + 0.6*0.3 + 0.9*0.1 = 1.56
(0.56)	(0.44)

CAWPE raises the weights to the power alpha. It uses a tilted distribution and inverse temperature



Class 1	Class 2
$0.8^4*0.48+$ $0.5^4*0.3+$ $0.75^4*0.3+$ $0.6^4*0.7+$ $0.9^4*0.9=$ **0.99**	$0.8^4*0.52+$ $0.5^4*0.7+$ $0.75^4*0.7+$ $0.6^4*0.3+$ $0.9^4*0.1=$ 0.58
(0.63)	(0.37)

Selection Schemes

- A currently popular approach is to build a large set of classifiers, then select a subset to actually use.
- This is often called an "overproduce and choose" scheme.
- Often they use a greedy algorithm to construct a model

Classifier 1	Classifier 2			Classifier T-1	Classifier T
Acc = 0.87	Acc=0.67			Acc=0.60	Acc=0.91

Classifier T		
Classifier T	Classifier 8	
Classifier T	Classifier 8	Classifier 3

Stacking

- Stacking involves constructing a "meta" model on top of the training data output.
- We perform a cross validation on the train data, which gives us predictions and probabilities for each train case
- Suppose we have three base classifiers and we cross validate to get predictions and probabilities on the train data

Random Forest

0	1	0.166	0.834
0	0	0.802	0.198
0	0	0.638	0.362
0	0	0.54	0.46
1	1	0.096	0.904
1	0	0.852	0.148
1	0	0.574	0.426

MLP

0	1	0.021619	0.978381
0	0	0.995253	0.004747
0	0	0.778656	0.221344
0	1	0.073482	0.926518
1	1	0.036168	0.963832
1	0	0.990263	0.009737
1	0	0.971657	0.028343

C4.5

0	1	0	1
0	0	0.666667	0.333333
0	1	0.25	0.75
0	1	0.25	0.75
1	0	0.666667	0.333333
1	0	1	0
1	0	1	0

Stacking

- We use the output to form a new classification problem. Assume we use the probabilities

0.166	0.834	0.022	0.978	0	1	0
0.802	0.198	0.995	0.005	0.667	0.333	0
0.638	0.362	0.779	0.221	0.25	0.75	0
0.54	0.46	0.073	0.927	0.25	0.75	0
0.096	0.904	0.036	0.964	0.667	0.333	1
0.852	0.148	0.99	0.01	1	0	1
0.574	0.426	0.972	0.028	1	0	1

$$\downarrow$$

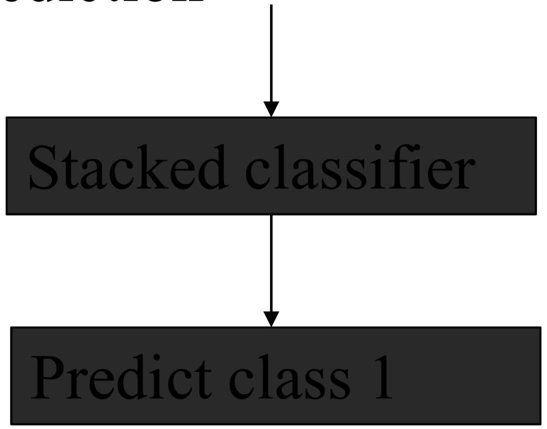
Train stacked classifier

Stacking

- New cases are classified by first finding the output of the base classifiers

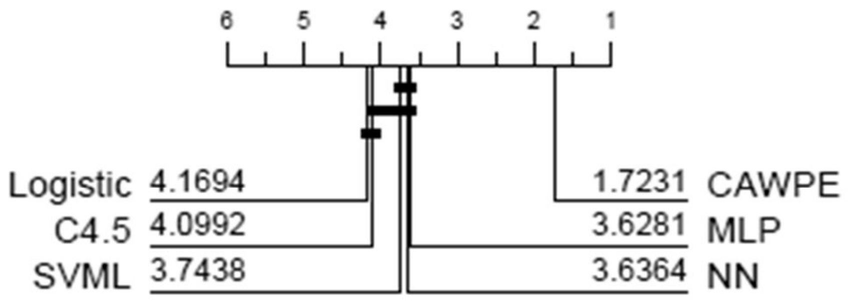
0.11	0.89	0.44	0.56	0.76	0.24

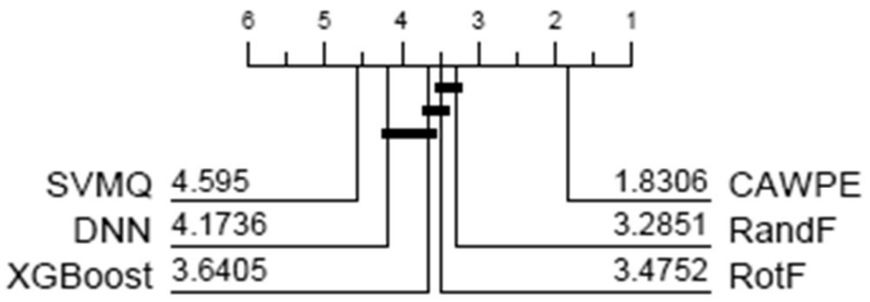
- This output is passed to the "meta" classifier, which then produces a prediction



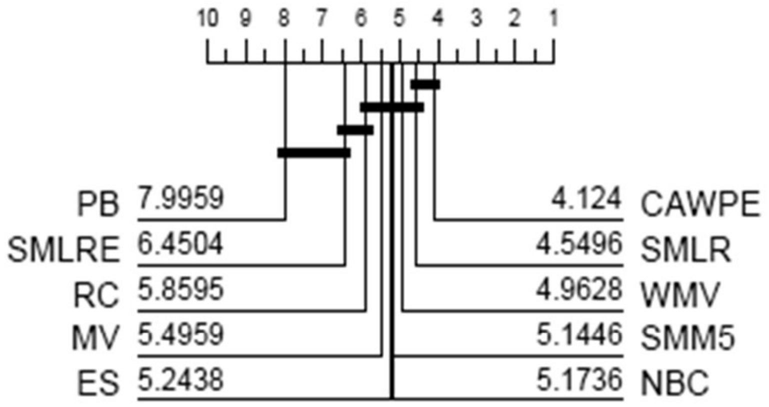
Algorithm 1 CAWPE classify(A test case $\boldsymbol{x}$)

```
Output: prediction for case $x$)
1: Given a set of classifiers $< M_1, \ldots, M_k >$, an exponent $\alpha$, a set of weights, $w_i$ and the
   number of classes $c$
2: $\{\hat{p}_1, \ldots, \hat{p}_c\} = \{0, \ldots, 0\}$
3: for $i \leftarrow 1$ to $k$ do
4:     for $j \leftarrow 1$ to $c$ do
5:         $\hat{q}_j \leftarrow \hat{p}((y = j|M_i, x)$
6:         $\hat{p}_j \leftarrow \hat{p}_j + w_i^{\alpha} \cdot \hat{q}_j$
       return $\arg\max_{j=1\ldots c} \hat{p}_j$
```

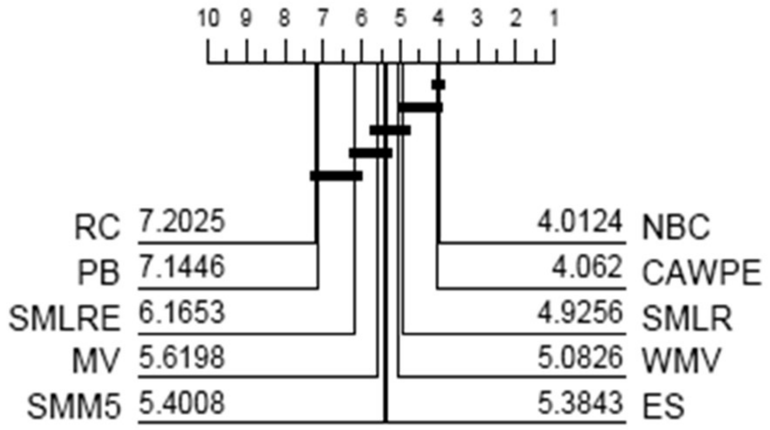




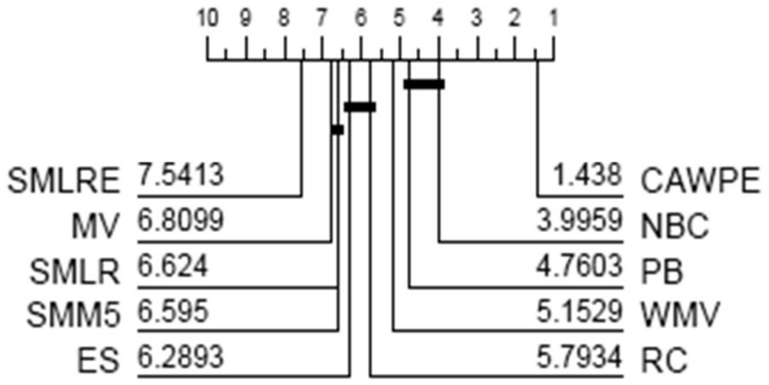
Comparison of Weighting Schemes for Heterogeneous Classifiers



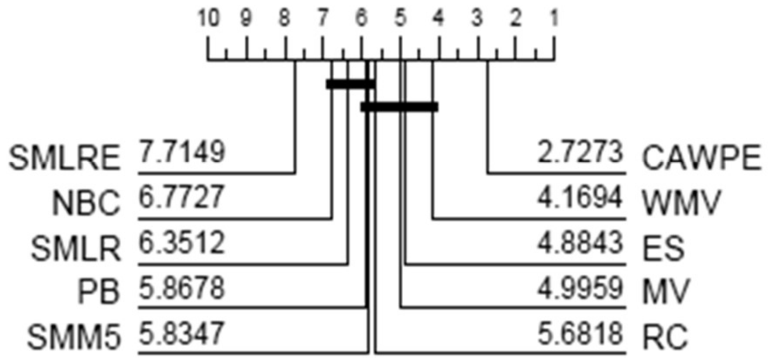
(a) Error



(b) Balanced Error

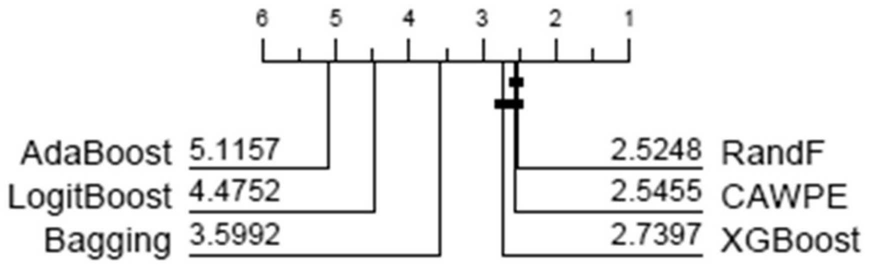


(c) AUC

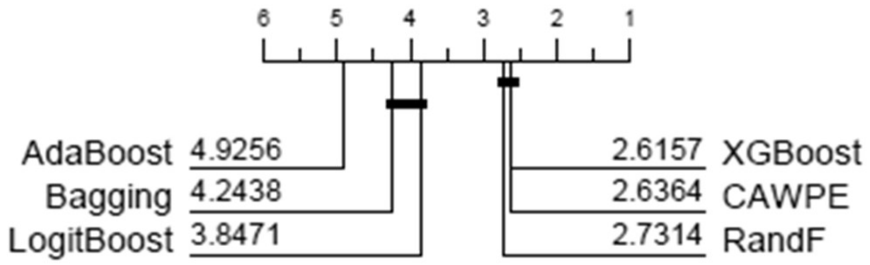


(b) NLL

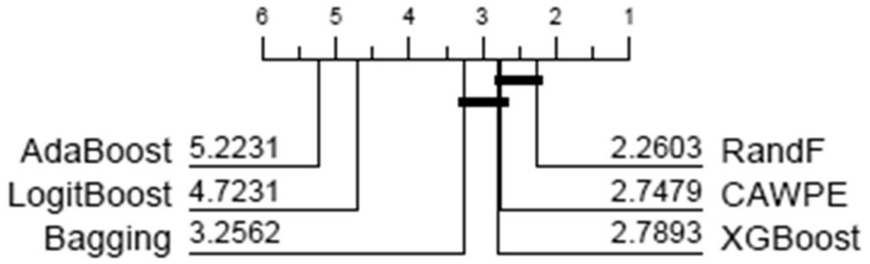
Comparison of Homogeneous classifiers (and basic CAWPE)

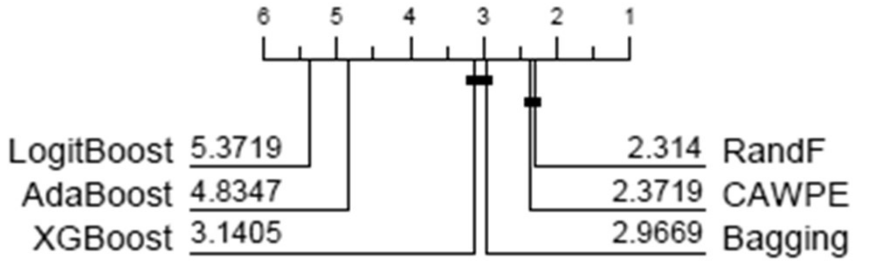


(a) Error



(b) Balanced Error





Comparison of tuned classifiers

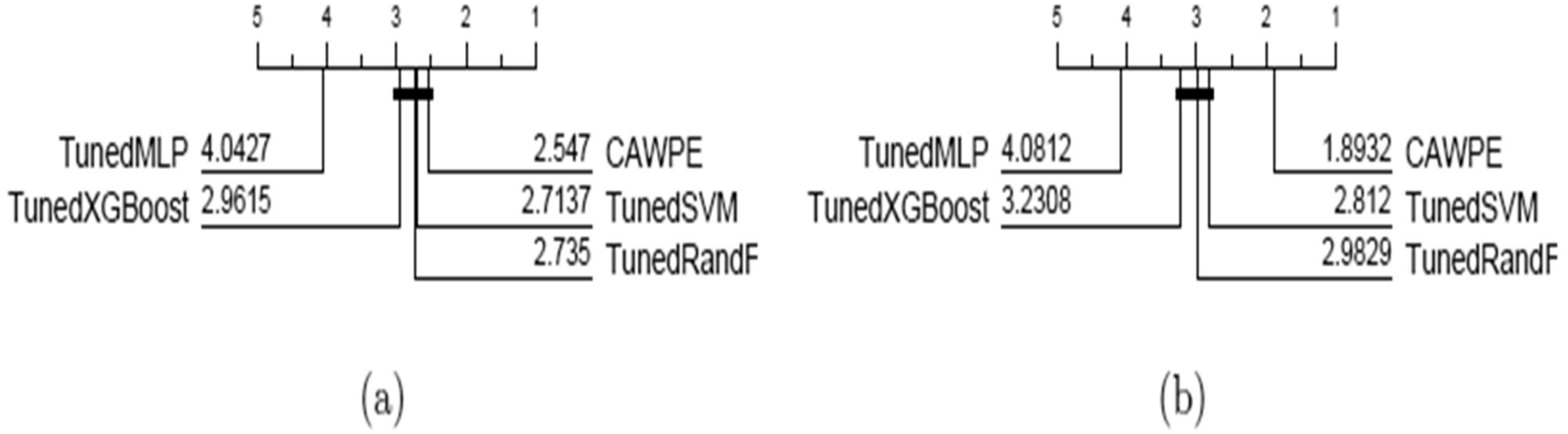


Fig. 6 Average ranked errors for CAWPE with (a) simpler and (b) advanced components against four tuned classifiers on 117 datasets in the UCI archive. The datasets adult, chess-krvk, miniboone and magic are omitted due to computational restraints.

Ensembles Summary

The design goal is diversity without loss of accuracy

Ensembles main purpose is to reduce the variability in estimates, but they are usually more accurate too.

Homogeneous ensembles use a large number of “weak” learners and inject diversity through sampling the data.

Samplers: ***Random Forest, Rotation Forest***

Boosters: ***AdaBoost, LogitBoost, XGBoost***

Heterogeneous classifiers use a smaller number of “strong” classifiers

Weighting Scheme: CAWPE

Overproduce and Select

Stacking