Waveform recognition with decision trees

Anton Antonov
Mathematica for Prediction blog
Mathematica for Prediction at GitHub
August 2013

Introduction

This document demonstrates the application of decision trees to the problem of waveform recognition taken from the book [1], "Classification and Regression Trees" by Breiman et al.

The *Mathematica* code for the decision tree building and classification can be downloaded from GitHub: see https://github.com/antononcube/MathematicaForPrediction .

In [1] the waveform recognition problem is given with three base waveforms. The code in this document can be easily extended to a larger number of waveforms.

A quick introduction into decision trees can found the Wikipedia article [3]; a quick and comprehensive introduction is given in [2]; a deep and insightful exposition is given in [1].

The waveform recognition problem

We have three waveforms h_1, h_2, h_3 that are piecewise linear functions defined as

$$
\begin{aligned}
h_1 &= \begin{cases} t-1 & 0 \le t \le 7 \\ 13-t & 7 \le t \le 13 \end{cases}, \\
h_2 &= \begin{cases} t-5 & 5 \le t \le 11 \\ 17-t & 11 \le t \le 17 \end{cases}, \\
h_3 &= \begin{cases} t-9 & 9 \le t \le 15 \\ 21-t & 15 \le t \le 21 \end{cases}.
\end{aligned}
\tag{1}
$$

Figure 1 shows these three base waveforms.

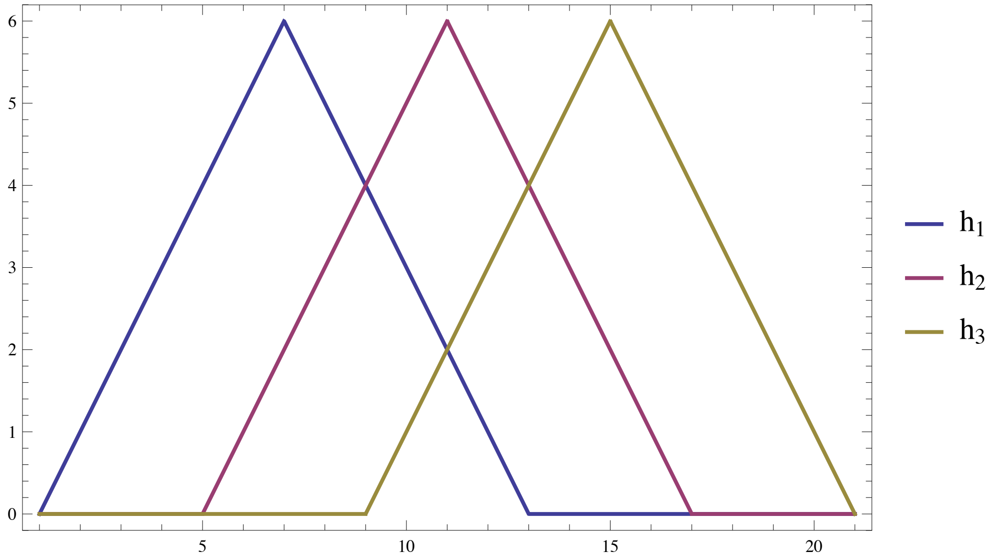


Figure 1: Three base waveforms

We have data array D with n rows and 21 columns. The rows of D are the linear combinations

$$\theta\, h_i(x) + (1-\theta)\, h_j(x),\;\; i,\, j \in [1, 2, 3],\, i \neq j,\;\; x \in [1, 21] \tag{2}$$

The data is overlaid with noise -- instead of (2) we have

$$\theta\, h_i(x) + (1-\theta)\, h_j(x) + \xi_x,\;\; i,\, j \in [1, 2, 3],\, i \neq j,\;\; x \in [1, 21], \tag{3}$$

where the random numbers vector ξ_x is generated with the normal distribution centered around 0 with standard deviation 1.

Problem: Given D and a vector v generated with (3) we want to determine which base waveforms have been used in (3) to generate v. ■

We have the following three ***class labels***

$$L := \{\text{class } 1+2,\;\; \text{class } 1+3,\;\; \text{class } 2+3\} \tag{4}$$

corresponding to all unique combinations of i and j in (3). We are looking into finding a classification function $F: \mathbb{R}^{21} \to L$. We are going to find such a classification function by building a decision tree using D.

We are going to call a row of D or a vector computed with (3) a ***wave sample***.

Figure 2 shows how the rows of D look and how they can be interpreted. The blue points represent the vectors generated with (3). The dashed red lines show the corresponding “clean” waves, with the noise vector ξ_x removed. The plot labels tell the corresponding waveform combination class labels.

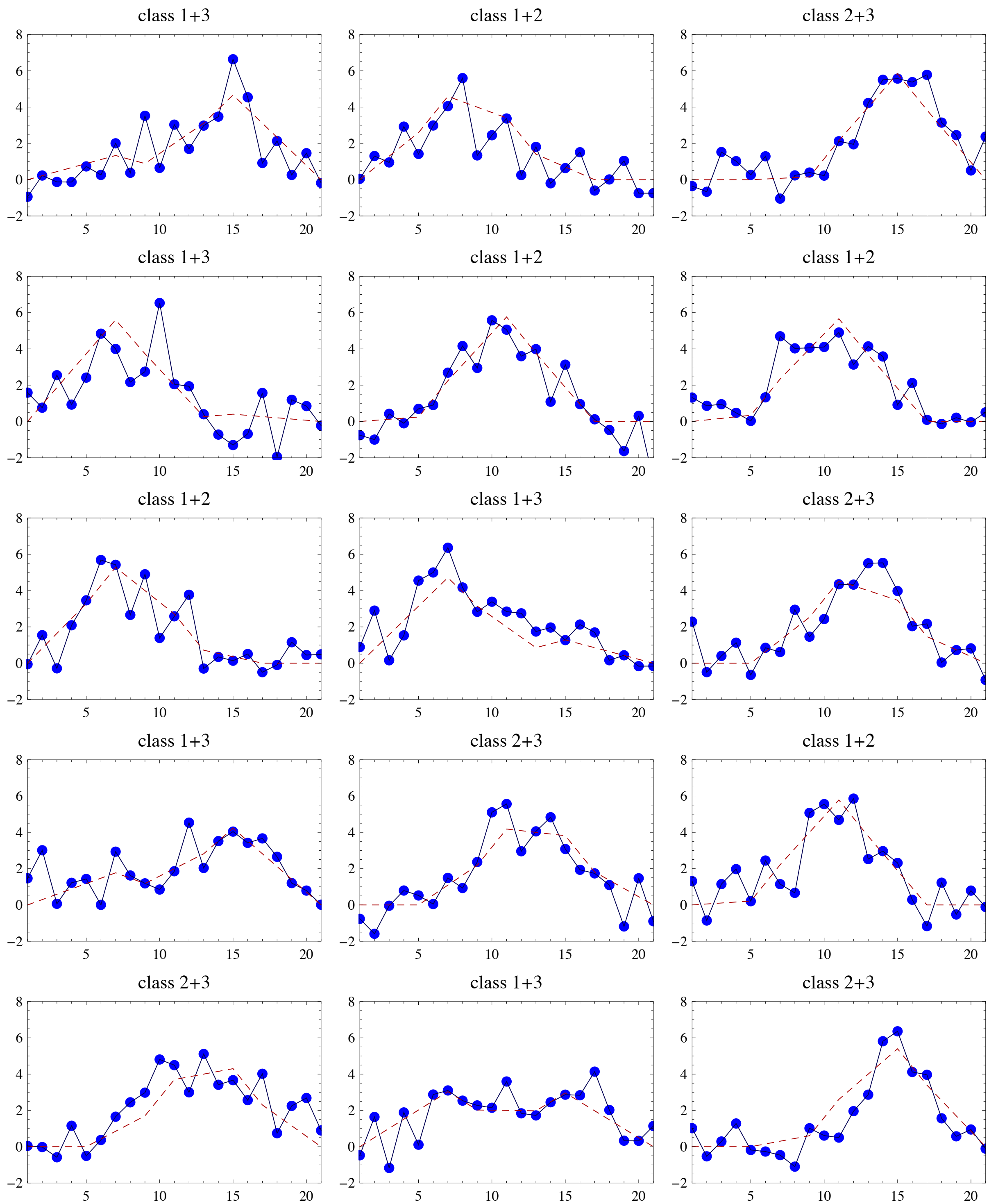


Figure 2. Examples of wave samples and their corresponding class labels.

With each integer $i \in [1, 21]$ we can associate a variable X_i. The domain of each variable X_i is $\text{dom}(X_i) \subset [-2, 10] \subset \mathbb{R}$. We can say that each row of D is a vector of values for the variables X_i, $i \in [1, 21]$, $i \in \mathbb{N}$ and we are looking for a classifier function

$$F : \text{dom}(X_1) \times \text{dom}(X_2) \times \ldots \times \text{dom}(X_{21}) \to L.$$

Data generation: three base waveforms

Here are the *Mathematica* definitions of the piecewise linear functions in (1):

```
In[8]:= h1[t_] := Piecewise[{{t - 1, 0 ≤ t <= 7}, {-t + 13, 7 <= t ≤ 13}}, 0];
h2[t_] := Piecewise[{{t - 5, 5 ≤ t ≤ 11}, {-t + 17, 11 <= t ≤ 17}}, 0];
h3[t_] := Piecewise[{{t - 9, 9 ≤ t ≤ 15}, {21 - t, 15 <= t ≤ 21}}, 0];
```

Here we map h_1, h_2, h_3 over [1, 21]:

```
In[11]:= ph1 = h1 /@ Range[1, 21];
ph2 = h2 /@ Range[1, 21];
ph3 = h3 /@ Range[1, 21];
```

The experimental data is generated in the following way.

1. Assign to n the total number of wave samples that are desired.

2. For each combination p from `{{ph1, ph2}, {ph1, ph3}, {ph2, ph3}}` and its corresponding class label c, start with and empty list $l_c = \{\}$ and do $n/3$ times the steps:

2.1. randomly generate θ, then use θ in (2) to obtain dc and in (3) to obtain d;
2.2. append the list $\{dc, d, c\}$ to the list l_c.

3. Join the lists computed with Step 2 in order to obtain the array t.

4. Shuffle the rows of t.

5. Form D by taking the second and third columns of t. Form the corresponding D_{clean} by taking the first and third columns of t.

Remark: The actual computations below are performed with the symbol `data` that corresponds to D. The symbol `dataCleanWaves` corresponds to D_{clean}.

```
In[14]:= n = 2000;
t = Flatten[#, 1] &@MapThread[
    Table[(θ = RandomReal[{0, 1}]; {θ #1[[1]] + (1 - θ) #1[[2]], θ #1[[1]] +
          (1 - θ) #1[[2]] + RandomReal[NormalDistribution[0, 1], {21}],
         #2}), {n / 3}] &, {{{ph1, ph2}, {ph1, ph3}, {ph2, ph3}},
     {"class 1+2", "class 1+3", "class 2+3"}}];
t = RandomSample[t];
data = Flatten /@ t[[All, {2, 3}]];
dataCleanWaves = Flatten /@ t[[All, {1, 3}]];
```

Decision tree classifier application

In this section we are going to build a decision tree over 300 randomly generated wave samples computed in the previous section. We are going to test the decision tree classification abilities over the rest of the wave samples.

To load the decision trees and forests package we use the command

```
(* load the decision tree package *)
Get["<directory specification>/AVCDecisionTreeForest.m"]
```

A short decision tree

First, for illustration purposes, we build a decision tree that is deliberately short (or shallow). Better classification results can be obtained with larger trees that using linear combinations of the variables X_i, $i \in [1, 21]$, $i \in \mathbb{N}$ -- see the next sub-sections.

With the following command we build a decision tree using the first 300 elements of `data` (that corresponds to *D* in the problem formulation) and assign it to the symbol `dtree` :

```
In[42]:= dtree = BuildDecisionTree[data⟦1 ;; 300⟧, {50, 0.06},
    "ImpurityFunction" → "Gini", "LinearCombinations" → {"Rank" → 0}];
```

The decision tree building command specifies that
1. the recursion process will stop if the data subset has less than 50 rows or if the impurity measure of the subset is less than 0.06;
2. the Gini index of heterogeneity is used to calculate the impurity of the subsets;
3. no linear combinations of the variables are used.

Figure 3 shows how the decision tree looks like. The non-leaf nodes have the form

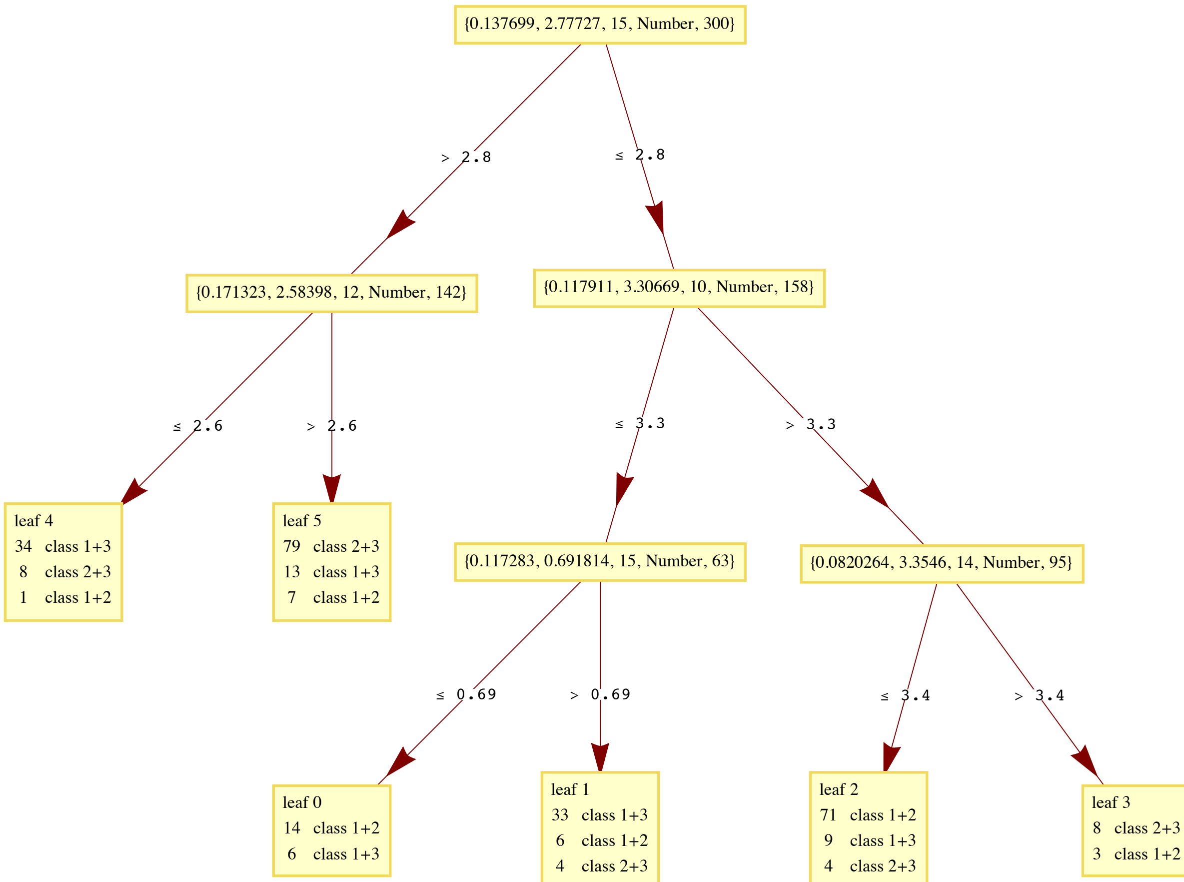
`{_?NumberQ, _?NumberQ, _Integer, Number, _Integer}` (5)

which is interpreted as

{*impurity*, *splitting value*, *splitting axis*, *variable type*, *data size*}.

Remark: Because all variables in the data are numerical we have only one variable type in the tree, `Number`. The package functions also work with categorical variables and their type is designated with `Symbol`.

The leaf nodes are lists of pairs, each pair is comprised of a score and a classification label. Each leaf node is ordered according to the labels frequency or probability to appear in the training data that corresponds to the data subset formed by the path from the tree root to the leaf node.



"Figure 3. A short decision tree build over 300 wave samples."

We can see in the tree on Figure 3 that the most decisive variables are the ones that correspond to the columns 6, 10, and 12 in `data`, which makes sense since the functions h_1 and h_2 have their maximums near 6 and 10 and 12 respectively.

Given a wave sample we can "run it through" `dtree` using the function `DecisionTreeClassify`. Here is how the classification result looks like (which is a leaf of `dtree`):

```
In[45]:= DecisionTreeClassify[dtree, data⟦400⟧]
Out[45]= {{71, class 1+2}, {9, class 1+3}, {4, class 2+3}}
```

Usually we take the first element of the classification result, [1], but we can also use weighted random selection of the labels.

It is a good idea to see how the classifier `dtree` performs over the rest of the randomly generated data, `data⟦301 ;; -1⟧`. It is especially interesting to see how successful the classification is over the subsets of `data⟦301;;-1⟧` corresponding to the different class labels. The package provides a function for such tests called

DecisionTreeClassificationSuccess.

In[46]:=
```
classResRules =
  DecisionTreeClassificationSuccess[dtree, data[[301 ;; -1]]]
```

Out[46]=
```
{{class 1+2, True} → 0.758865, {class 1+2, False} → 0.241135,
 {class 1+3, True} → 0.544658, {class 1+3, False} → 0.455342,
 {class 2+3, True} → 0.751332, {class 2+3, False} → 0.248668,
 {All, True} → 0.684335, {All, False} → 0.315665}
```

The first rule of the result says that the fraction of correct guesses for “class 1+2” is ≈ 0.76. The second rules says that the fraction of incorrect guess for “class 1+2” is ≈ 0.24.

This table presents the result of `DecisionTreeClassificationSuccess` in a much easier to interpret manner:

Out[47]=

Label	Fraction of correct guesses	Fraction of incorrect guesses
class 1+2	0.758865	0.241135
class 1+3	0.544658	0.455342
class 2+3	0.751332	0.248668
All	0.684335	0.315665

Better classification results using a larger tree

Let us now construct a tree which goes deeper into the division of the subsets of `data` with the command

In[49]:=
```
dtree = BuildDecisionTree[data[[1 ;; 300]], {5, 0},
   "ImpurityFunction" → "Gini", "LinearCombinations" → {"Rank" → 0}];
```

The decision tree building command specifies that
1. the recursion process will stop if the data subset has less than 5 rows or if the impurity measure of the subset is 0;
2. the Gini index of heterogeneity is used to calculate the impurity of the subsets;
3. no linear combinations of the variables are used.

We can see that classification success is higher with this tree.

In[50]:=
```
classResRules =
  DecisionTreeClassificationSuccess[dtree, data[[301 ;; -1]]]
```

Out[50]=
```
{{class 1+2, True} → 0.751773, {class 1+2, False} → 0.248227,
 {class 1+3, True} → 0.62697, {class 1+3, False} → 0.37303,
 {class 2+3, True} → 0.698046, {class 2+3, False} → 0.301954,
 {All, True} → 0.691991, {All, False} → 0.308009}
```

The previous output tabulated:

Out[51]=

Label	Fraction of correct guesses	Fraction of incorrect guesses
class 1+2	0.751773	0.248227
class 1+3	0.62697	0.37303
class 2+3	0.698046	0.301954
All	0.691991	0.308009

Although the overall classification success is nearly the same, we can see that success rates are more evenly distributed over the labels.

Better classification results using a linear combinations of variables

Given a recursion step s of the decision tree building, it is possible that for step's subset of rows, D_s, of the data array D the best splitting is not along just one of the columns corresponding to the numerical variables X_i, $i \in [1, 21]$, but along a linear combination of them. In other words we are looking for splittings like

$$\sum_{i=1}^{21} \omega_i X_i \leq v, \tag{6}$$

for some ω_i, $v \in \mathbb{R}$, $i \in [1, 21]$. Instead of an implementation of a (local) extremum search procedure as the one described in [1], the decision tree building function `BuildDecisionTree` uses low-rank SVD to obtain directions (ω_i's) for the search of linear combinations splittings.

Let us now construct a tree which searches for splittings like (6).

In[53]:=
```
dtree = BuildDecisionTree[data〚1 ;; 300〛, {5, 0},
    "ImpurityFunction" → "Gini", "LinearCombinations" → {"Rank" → 4}];
```

This decision tree building command specifies that

1. the recursion process will stop if the data subset has less than 5 rows or if the impurity measure of the subset is 0;
2. the Gini index of heterogeneity is used to calculate the impurity of the subsets;
3. use linear combinations generated by 4 orthogonal vectors obtained using low-rank SVD.

We can see that the classification success with this tree is higher than the ones of the earlier built decision trees.

In[54]:=
```
classResRules =
  DecisionTreeClassificationSuccess[dtree, data〚301 ;; -1〛]
```

Out[54]=
```
{{class 1+2, True} → 0.797872, {class 1+2, False} → 0.202128,
 {class 1+3, True} → 0.65324, {class 1+3, False} → 0.34676,
 {class 2+3, True} → 0.746004, {class 2+3, False} → 0.253996,
 {All, True} → 0.732038, {All, False} → 0.267962}
```

The previous output tabulated:

Out[55]=

Label	Fraction of correct guesses	Fraction of incorrect guesses
class 1+2	0.797872	0.202128
class 1+3	0.65324	0.34676
class 2+3	0.746004	0.253996
All	0.732038	0.267962

Classification rates wrt tuning parameters

It is interesting to see how the classification success evolves with respect to changing a parameter of the decision tree building. In this section we are going to look into the changes of three parameters: impurity threshold, rank of linear combinations of variables, and number of strata.

Tree size controlled by impurity threshold

The basic way to control the building process of a decision tree is to provide stopping criteria for the recursive process by specifying subset size threshold or by an impurity measure threshold. In this sub-section we compute decision trees over a range of impurity thresholds and plot their classification success statistics.

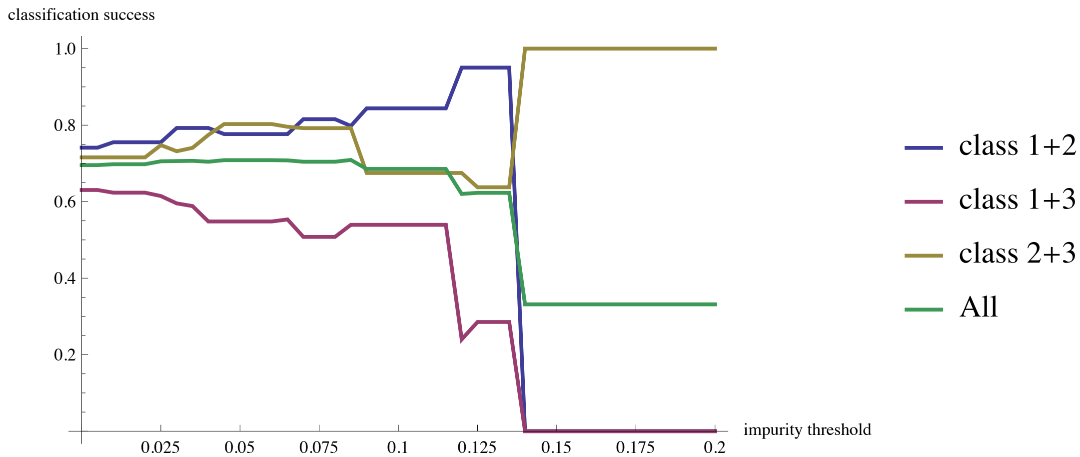
In[57]:=

```
AbsoluteTiming[
 dtrees = Table[BuildDecisionTree[data⟦1 ;; 300⟧, {5, μ},
    "ImpurityFunction" → "Gini", "LinearCombinations" → {"Rank" → 0},
    "PreStratify" → True, "Strata" → 100], {μ, 0, 0.2, 0.005}];
 cres = DecisionTreeClassificationSuccess[#, data⟦301 ;; -1⟧] & /@
   dtrees;
]
```

Out[57]=

```
{181.299365, Null}
```

Out[58]=



What we see on the plot is to be expected.

1. Requiring 0 impurity of the leaf subsets results in over-fitting, that is why the overall success rate (in green) slightly increases initially when the impurity is greater than 0.

2. When the trees become too short the overall classification rate deteriorates.

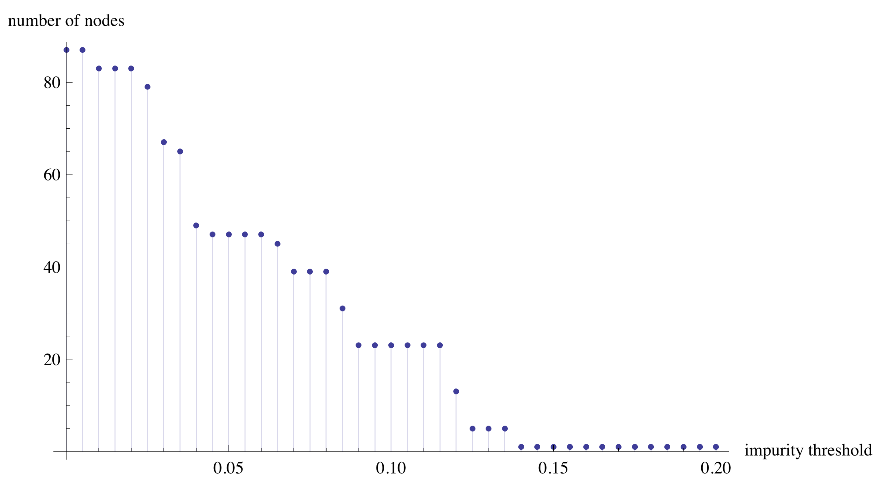
3. When the trees have only one leaf node for one of the class labels we have 100% classification success rate, and 0% for the rest.

We can write a simple recursive function to count the nodes in a tree:

In[64]:=
```
Clear[NodeCount]
NodeCount[tree_] :=
  Which[
   Length[Rest[tree]] == 0, 1,
   Length[Rest[tree]] > 0, 1 + Total[NodeCount /@ Rest[tree]]
  ];
```

Using this function we can calculate the number of nodes in each decision tree with respect to the impurity threshold parameter.

Out[67]=



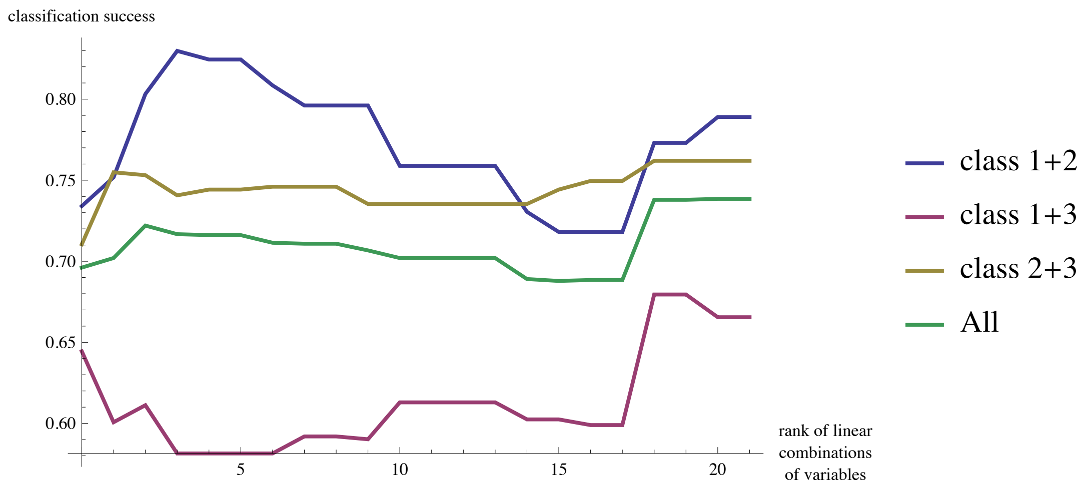
Rank of linear combinations of variables

In this sub-section we compute a list of decision trees, `dtrees`, over a range of low-ranks for SVD used for splittings like (6) and then plot the classification success statistics of `dtrees` with respect to the low-rank parameter.

In[68]:=
```
AbsoluteTiming[
 dtrees = Table[BuildDecisionTree[data⟦1 ;; 300⟧, {5, 0},
    "ImpurityFunction" → "Gini", "LinearCombinations" →
     {"Rank" → nd}, "Strata" → 10], {nd, Range[0, 21]}];
 cres = DecisionTreeClassificationSuccess[#, data⟦301 ;; -1⟧] & /@
   dtrees;
]
```

Out[68]= {98.106034, Null}

Out[69]=

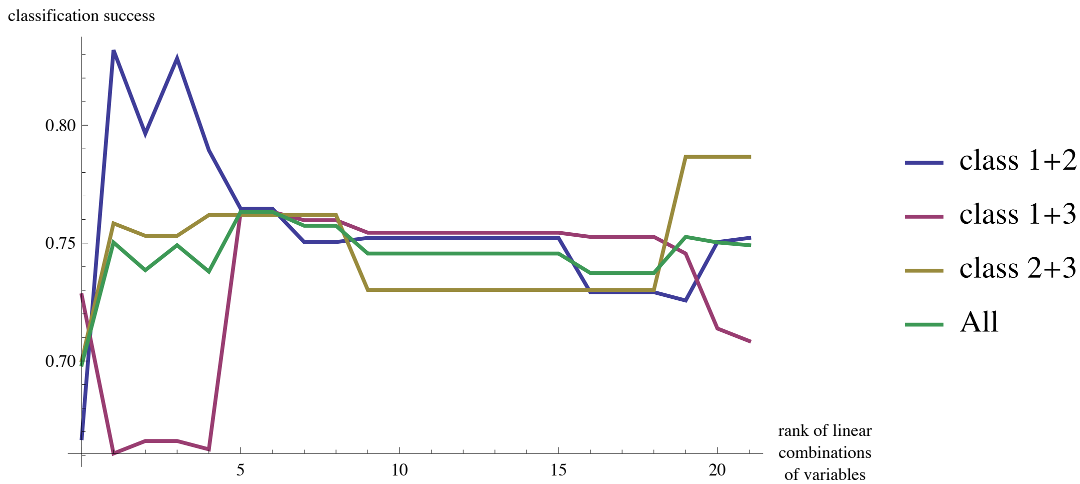


From the last plot we see that (i) using linear combinations of variables improves the overall recognition rate, and (ii) that the overall recognition rate changes smoothly for ranks in the interval [2, 17]. Only the first of these observation holds with different partitionings of *D* into training and test data.

In[70]:=
```
(* another statistics calculation with a
 different partitioning into training and test data *)
AbsoluteTiming[
 Block[{data = RandomSample[data]},
  dtrees = Table[BuildDecisionTree[data⟦1 ;; 300⟧, {5, 0},
     "ImpurityFunction" → "Gini", "LinearCombinations" →
      {"Rank" → nd}, "Strata" → 10], {nd, Range[0, 21]}];
  cres = DecisionTreeClassificationSuccess[#, data⟦301 ;; -1⟧] & /@
    dtrees;
 ]
]
```

Out[70]= {99.452690, Null}

Out[71]=



From the last plot we see again that using linear combinations of variables improves the overall recognition rate.

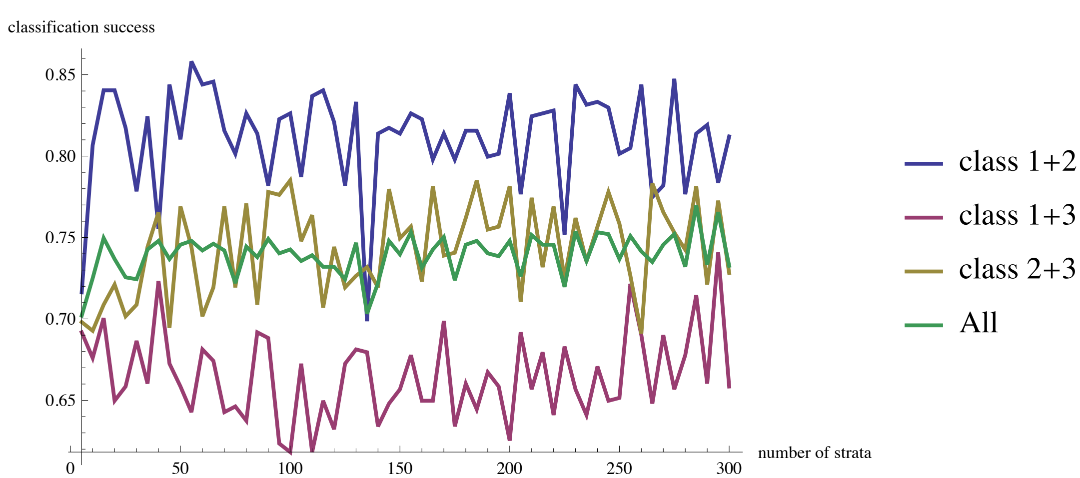
Number of strata with pre-stratification, no linear combinations

In this sub-section we compute a list of decision trees, `dtrees`, over a range of different number of strata. The number of strata option, “Strata”, is used to specify into how many intervals to discretize the numerical variables. The option “PreStratify” is used to specify when the stratification is done: (i) before the building of the decision tree, or (ii) at each recursive step, for each subset of the data.

In[72]:=
```
AbsoluteTiming[
 dtrees = Table[BuildDecisionTree[data⟦1 ;; 300⟧, {5, 0},
    "ImpurityFunction" → "Gini", "LinearCombinations" → {"Rank" → 3},
    "PreStratify" → True, "Strata" → ns], {ns, Range[5, 300, 5]}];
 cres = DecisionTreeClassificationSuccess[#, data⟦301 ;; -1⟧] & /@
   dtrees;]
```

Out[72]= {719.823092, Null}

Out[73]=



From the last plot we can see that the recognition rates vary non-smoothly with respect to the impurity threshold. This can be explained by the way the training and test data are generated. In formula (3) we add noise that can change the clean signal by 30%. The overall recognition rate varies within a smaller range because of compensation effects.

Using random forests

From the statistics and plots in the previous section we can see that it might be beneficial to use multiple decision trees instead of just one decision tree. This is the idea behind the so called Random forest classifier, [4].

In this section we make several decision forest building runs using different partitionings of the array of wave samples *D*. Since we consider all X_i, $i \in [1, 21]$, to be important for the recognition we do not random subsets of $\{X_i\}_{i=1}^{21}$ for each tree in the random forest. (As prescribed in Breiman's algorithm for random forests building, see [4].) Instead, for each tree we use tuning parameters that are random numbers from suitable ranges.

(In order to specify that random axes selection should be used, the decision tree building functions take the option "RandomAxes". The option takes expressions matching `All | True | _Integer`.)

In the three experiments below it can be seen that the random forests of 20 trees bring better classification results than a single decision tree. Since we use all variables X_i, a random decision tree forest of size *n* requires, on average, *n* times longer calculation time than the time needed for one tree. Note that each experiment partitions *D* into different training and testing subsets.

Experiment 1

```
AbsoluteTiming[
 Block[{data = RandomSample[data]},
  dtree =
   BuildDecisionTree[data[[1 ;; 300]], {5, 0.01}, "RandomAxes" → All,
    "ImpurityFunction" → "Gini", "LinearCombinations" → {"Rank" → 3},
    "Strata" → 100, "PreStratify" → False];
  dforest =
   Table[BuildDecisionTree[data[[1 ;; 300]],
     {RandomChoice[{5, 10}], RandomReal[{0.01, 0.06}]},
     "RandomAxes" → All, "ImpurityFunction" → "Gini",
     "LinearCombinations" → {"Rank" → RandomInteger[{1, 5}]},
     "Strata" → RandomInteger[{40, 150}],
     "PreStratify" → True], {20}];
  treeClassResRules = DecisionTreeClassificationSuccess[
    dtree, data[[301 ;; -1]]];
  forestClassResRules = DecisionForestClassificationSuccess[
    dforest, data[[301 ;; -1]]];
 ]]
```

Out[163]= {170.480853, Null}

The classification rates of the decision tree in the previous calculations tabulated:

Out[173]=

Label	Fraction of correct guesses	Fraction of incorrect guesses
class 1+2	0.815603	0.184397
class 1+3	0.775832	0.224168
class 2+3	0.840142	0.159858
All	0.810365	0.189635

The classification rates of the decision forest in the previous calculations tabulated:

Out[170]=

Label	Fraction of correct guesses	Fraction of incorrect guesses
class 1+2	0.86087	0.13913
class 1+3	0.813743	0.186257
class 2+3	0.861404	0.138596
All	0.845701	0.154299

Experiment 2

In[175]:=
```
AbsoluteTiming[
 Block[{data = RandomSample[data]},
  dtree =
   BuildDecisionTree[data[[1 ;; 300]], {5, 0.01}, "RandomAxes" → All,
    "ImpurityFunction" → "Gini", "LinearCombinations" → {"Rank" → 3},
    "Strata" → 100, "PreStratify" → False];
  dforest =
   Table[BuildDecisionTree[data[[1 ;; 300]],
     {RandomChoice[{5, 10}], RandomReal[{0.01, 0.06}]},
     "RandomAxes" → All, "ImpurityFunction" → "Gini",
     "LinearCombinations" → {"Rank" → RandomInteger[{1, 5}]},
     "Strata" → RandomInteger[{40, 150}],
     "PreStratify" → True], {20}];
  treeClassResRules = DecisionTreeClassificationSuccess[
    dtree, data[[301 ;; -1]]];
  forestClassResRules = DecisionForestClassificationSuccess[
    dforest, data[[301 ;; -1]]];
 ]]
```

Out[175]= {178.177245, Null}

The classification rates of the decision tree in the previous calculations tabulated:

Out[176]=

Label	Fraction of correct guesses	Fraction of incorrect guesses
class 1+2	0.805893	0.194107
class 1+3	0.724868	0.275132
class 2+3	0.788809	0.211191
All	0.773263	0.226737

The classification rates of the decision forest in the previous calculations tabulated:

Out[178]=

Label	Fraction of correct guesses	Fraction of incorrect guesses
class 1+2	0.840555	0.159445
class 1+3	0.664903	0.335097
class 2+3	0.893502	0.106498
All	0.799176	0.200824

Experiment 3

In[180]:=
```
AbsoluteTiming[
 Block[{data = RandomSample[data]},
  dtree =
   BuildDecisionTree[data[[1 ;; 300]], {5, 0.01}, "RandomAxes" → All,
    "ImpurityFunction" → "Gini", "LinearCombinations" → {"Rank" → 3},
    "Strata" → 100, "PreStratify" → False];
  dforest =
   Table[BuildDecisionTree[data[[1 ;; 300]],
     {RandomChoice[{5, 10}], RandomReal[{0.01, 0.06}]},
     "RandomAxes" → All, "ImpurityFunction" → "Gini",
     "LinearCombinations" → {"Rank" → RandomInteger[{1, 5}]},
     "Strata" → RandomInteger[{40, 150}],
     "PreStratify" → True], {20}];
  treeClassResRules = DecisionTreeClassificationSuccess[
    dtree, data[[301 ;; -1]]];
  forestClassResRules = DecisionForestClassificationSuccess[
    dforest, data[[301 ;; -1]]];
 ]]
```

Out[180]= {197.486547, Null}

The classification rates of the decision tree in the previous calculations tabulated:

Out[181]=

Label	Fraction of correct guesses	Fraction of incorrect guesses
class 1+2	0.774138	0.225862
class 1+3	0.734403	0.265597
class 2+3	0.748654	0.251346
All	0.75265	0.24735

The classification rates of the decision forest in the previous calculations tabulated:

Out[183]=

Label	Fraction of correct guesses	Fraction of incorrect guesses
class 1+2	0.82069	0.17931
class 1+3	0.71836	0.28164
class 2+3	0.813285	0.186715
All	0.784452	0.215548

References

[1] Leo Breiman et al., Classification and regression trees, Chapman & Hall, 1984.

[2] Nong Ye, editor, The Handbook of Data Mining, Lawrence Erlbaum Associates, 2004.

[3] Decision tree, Wikipedia, http://en.wikipedia.org/wiki/Decision_tree.

[4] Random forest, Wikipedia, http://en.wikipedia.org/wiki/Random_forest.

[5] Lars Eldén, Matrix Methods in Data Mining and Pattern Recognition, SIAM, 2007.

[6] Singular value decomposition, http://en.wikipedia.org/wiki/Singular_value_decomposition.