

# Importance of variables investigation guide

Importance of variables investigation using Classifiers, Mosaic plots, Decision trees, Association rules, and Dimension reduction

Anton Antonov  
MathematicaForPrediction blog at WordPress  
MathematicaForPrediction blog at GitHub  
December 2015  
January 2016  
Version 1.0

---

## Introduction

The document is meant to serve as a guide for variable importance finding and it is organized to have the flow and corresponding explanations in parallel for the two datasets, "Titanic" and "Mushroom".

One of the questions that would follow the formulation of a classification problem for a given data set is which variables are most important for the correct classification to the class labels.

For example, assume we have mushroom edibility data exemplified with this sample:

Out[174]=

#	cap-shape	cap-color	bruises?	odor	habitat	edibility of mushroom
1	convex	brown	True	pungent	urban	poisonous
2	convex	yellow	True	almond	grasses	edible
3	bell	white	True	anise	meadows	edible
4	convex	white	True	pungent	urban	poisonous
5	convex	gray	False	none	grasses	edible
6	convex	yellow	True	almond	grasses	edible
7	bell	white	True	almond	meadows	edible
8	bell	white	True	anise	meadows	edible

We can ask the following questions:

1. Which variables (cap shape, odor, etc.) are most decisive for determining the edibility of a mushroom?
2. Which values of which variables form conditions that would imply a poisonous mushroom?

This guide is mostly about answering the first question. The second question is answered in order to validate the answers for the first.

In the blog post "Classification and association rules for census income data", [2], I have described a procedure for determining variable importance that answers similar questions. The procedure builds a classifier with the training data, damages the test data in a systematic way for each variable, and compares the "damage effects", i.e. how much worse the classifier performs over the damaged test data. The variables that produce worst results are considered most important.

The procedure is taken from the book "Classification and regression trees", [1], largely written by Leo Breiman, the great advocate and inventor of random forests. The blog post [2] has examples and a

discussion of determining variable importance using two classifiers, Decision Trees, [3], and Naive Bayesian Classifier (NBC), [4].

The importance of variables determination has to be considered in the larger context of the data. The outcomes of the procedures used have to be evaluated do they make sense and do they explain observations from the data. For that in this document we are going to (i) see the application of Mosaic Plots [5,6], (ii) examine Decision trees [10], (iii) Association rules [11], and (iv) topics from dimension reduction, [13, 14].

In this document two datasets are used -- "Titanic" and "Mushroom" -- to make examples that illustrate the algorithms and procedures. Both datasets are available in *Mathematica* versions 10 and later.

"Titanic" consists of metadata and survival records of passengers in the wreck of the ship Titanic.

"Mushroom" has records for mushroom edibility (with much more records and variables than in the table above). The datasets are of mostly categorical data. For purely numerical data techniques like Principal Components Analysis (PCA) are applicable. (Not covered in this guide.) Nevertheless, numerical variables can be converted into categorical with suitable piecewise constant functions. (See the example below for "passenger age" of "Titanic".)

*Mathematica's* versions 10.0 and later have the function `Classify` that generates classification functions over a variety of data sets. The classifiers in the blog post [2] were developed before the introduction of `Classify` in *Mathematica*. This document discusses code, [8], and applications based on `Classify` and related functions for data partitioning and classifier function querying or invocation.

**Remark:** It should be pointed out that the considered datasets ("Titanic" and "Mushroom") are relatively small and give nice and clean results. Real applications of the described procedures might require preliminary (extensive) data cleaning and normalization before obtaining good classification and variable importance results.

This document has a version number associated with it since it is available at [MathematicaForPrediction](#) at GitHub and I plan to update and extend it. Another plan I have is to make a version of this document that includes commands in R, or it is entirely written with R and RStudio tools.

---

## Datasets

In this section we introduce the datasets with samples and data summaries.

### Titanic

The following command loads the "Titanic" dataset. The dataset is given as a list of rules, so in order to tabulate it we are going to flatten it first.

```
In[1]:= titanicDataset =
      Map[Flatten, List @@@ ExampleData[{"MachineLearning", "Titanic"}, "Data"]];
      Dimensions[titanicDataset]
```

```
Out[2]= {1309, 4}
```

Here are the variable names (the last one is the one with the class labels):

```
In[3]:= titanicVarNames = Flatten[
      List @@@ ExampleData[{"MachineLearning", "Titanic"}, "VariableDescriptions"]]
```

```
Out[3]= {passenger class, passenger age, passenger sex, passenger survival}
```

The following command tabulates a sample of the Titanic dataset rows. Each row represents a

passenger.

```
In[4]:= Magnify[#, 0.8] &@TableForm[RandomSample[titanicDataset, 12],
  TableHeadings -> {None, titanicVarNames}]
```

```
Out[4]=
```

passenger class	passenger age	passenger sex	passenger survival
2nd	32.	male	died
3rd	26.	male	died
2nd	46.	male	died
1st	39.	female	survived
1st	48.	female	survived
3rd	18.	male	died
3rd	21.	male	died
3rd	24.	female	died
3rd	27.	female	survived
1st	45.	female	survived
1st	42.	male	died
3rd	18.	male	survived

Here is a summary of different columns. (See [7] for the function RecordsSummary.)

```
In[5]:= Magnify[#, 0.8] &@
  Grid[List@RecordsSummary[titanicDataset /. _Missing -> 0, titanicVarNames],
  Dividers -> All, Alignment -> {Left, Top}]
```

```
Out[5]=
```

1 passenger class	2 passenger age	3 passenger sex	4 passenger survival
3rd 709	Min 0	male 843	died 809
1st 323	1st Qu 7.	female 466	survived 500
2nd 277	Mean 23.8775		
	Median 24.		
	3rd Qu 35.		
	Max 80.		

We can see that for 263 (or 20%) of the records the passenger age values are missing:

```
In[6]:= Count[#, _Missing] & /@
  AssociationThread[titanicVarNames, Transpose[titanicDataset]]
  % / Dimensions[titanicDataset][[1]] // N
```

```
Out[6]= <|passenger class -> 0, passenger age -> 263,
  passenger sex -> 0, passenger survival -> 0|>
```

```
Out[7]= <|passenger class -> 0., passenger age -> 0.200917,
  passenger sex -> 0., passenger survival -> 0.|>
```

## Mushroom

The following command loads the “Mushroom” dataset. The dataset is given as a list of rules, so in order to tabulate it we are going to flatten it first.

```
In[8]:= mushroomDataset = Map[Flatten,
  List@@@ ExampleData[{"MachineLearning", "Mushroom"}], "Data"]];
Dimensions[mushroomDataset]
```

```
Out[9]= {8124, 23}
```

Here are the variable names (the last one is the one with the class labels):

```
In[10]:= mushroomVarNames = Flatten[List @@
      ExampleData[{"MachineLearning", "Mushroom"}, "VariableDescriptions"]];
mushroomVarNames[[-1]] = StringReplace[mushroomVarNames[[-1]],
      " (" ~~ x___ ~~ ")" :> ""];
mushroomVarNames
```

```
Out[12]:= {cap-shape, cap-surface, cap-color, bruises?, odor, gill-attachment,
      gill-spacing, gill-size, gill-color, stalk-shape, stalk-root,
      stalk-surface-above-ring, stalk-surface-below-ring, stalk-color-above-ring,
      stalk-color-below-ring, veil-type, veil-color, ring-number, ring-type,
      spore-print-color, population, habitat, edibility of mushroom}
```

The following command tabulates a sample of the “Mushroom” dataset rows (in the PDF not all columns are seen):

```
In[13]:= Magnify[#, 0.8] &@TableForm[RandomSample[mushroomDataset, 12],
      TableHeadings -> {None, mushroomVarNames}]
```

cap-shape	cap-surface	cap-color	bruises?	odor	gill-attachment	gill-spacing	gi
flat	smooth	brown	False	none	attached	close	br
convex	smooth	red	False	foul	free	close	na
flat	fibrous	gray	False	foul	free	close	br
convex	fibrous	yellow	False	foul	free	close	br
flat	fibrous	gray	False	none	free	crowded	br
convex	scaly	brown	True	anise	free	close	br
convex	fibrous	gray	False	foul	free	close	br
flat	scaly	brown	True	none	free	close	br
flat	scaly	brown	False	foul	free	close	na
convex	fibrous	brown	True	none	free	close	br
knobbed	smooth	brown	True	none	free	close	br
flat	smooth	brown	False	foul	free	close	na

The following command gives summaries of the different columns. (See [7] for the function `RecordsSummary`.)

```
In[14]:= Magnify[#, 0.8] &@
Grid[ArrayReshape[RecordsSummary[mushroomDataset /. _Missing -> 0,
mushroomVarNames], {6, 4}, ""], Dividers -> All, Alignment -> {Left, Top}]
```

<b>1 cap-shape</b> convex 3656 flat 3152 knobbed 828 bell 452 sunken 32 conical 4	<b>2 cap-surface</b> scaly 3244 smooth 2556 fibrous 2320 grooves 4	<b>3 cap-color</b> brown 2284 gray 1840 red 1500 yellow 1072 white 1040 buff 168 (Other) 220	<b>4 bruises?</b> False 4748 True 3376
<b>5 odor</b> none 3528 foul 2160 fishy 576 spicy 576 almond 400 anise 400 (Other) 484	<b>6 gill-attachment</b> free 7914 attached 210	<b>7 gill-spacing</b> close 6812 crowded 1312	<b>8 gill-size</b> broad 5612 narrow 2512
<b>9 gill-color</b> buff 1728 pink 1492 white 1202 brown 1048 gray 752 chocolate 732 (Other) 1170	<b>10 stalk-shape</b> tapering 4608 enlarging 3516	<b>11 stalk-root</b> bulbous 3776 0 2480 equal 1120 club 556 rooted 192	<b>12 stalk-surface-above-ring</b> smooth 5176 silky 2372 fibrous 552 scaly 24
<b>13 stalk-surface-below-ring</b> smooth 4936 silky 2304 fibrous 600 scaly 284	<b>14 stalk-color-above-ring</b> white 4464 pink 1872 gray 576 brown 448 buff 432 orange 192 (Other) 140	<b>15 stalk-color-below-ring</b> white 4384 pink 1872 gray 576 brown 512 buff 432 orange 192 (Other) 156	<b>16 veil-type</b> partial 8124
<b>17 veil-color</b> white 7924 brown 96 orange 96 yellow 8	<b>18 ring-number</b> one 7488 two 600 none 36	<b>19 ring-type</b> pendant 3968 evanescent 2776 large 1296 flaring 48 none 36	<b>20 spore-print-color</b> white 2388 brown 1968 black 1872 chocolate 1632 green 72 buff 48 (Other) 144
<b>21 population</b> several 4040 solitary 1712 scattered 1248 numerous 400 abundant 384 clustered 340	<b>22 habitat</b> woods 3148 grasses 2148 paths 1144 leaves 832 urban 368 meadows 292 waste 192	<b>23 edibility of mushroom</b> edible 4208 poisonous 3916	

We can see that for 2480 (or  $\approx 31\%$ ) of the records the values of the column “stalk-root” are missing:

```
In[15]:= t = Count[#, _Missing] & /@
AssociationThread[mushroomVarNames, Transpose[mushroomDataset]]];
Select[t, # > 0 &]
% / Dimensions[mushroomDataset][[1]] // N
```

```
Out[16]= <|stalk-root -> 2480|>
```

```
Out[17]= <|stalk-root -> 0.305268|>
```

## Procedure and implementation

This section has descriptions of the fundamental (main) procedure used in this guide for variable importance investigation.

### Procedure outline

1. Split the data into training and testing datasets.
2. Build a classifier with the training set.
3. Verify using the test set that good classification results are obtained. Find the baseline accuracy.
4. If the number of variables (attributes) is  $k$  for each  $i$ ,  $1 \leq i \leq k$ :
  - 4.1. Shuffle the values of the  $i$ -th column of the test data and find the classification success rates.
5. Compare the obtained  $k$  classification success rates between each other and with the success rates obtained by the un-shuffled test data.
6. The variables for which the classification success rates are the worst are the most decisive.

Note that instead of using the overall baseline accuracy we can make the comparison over the accuracies for selected, more important class labels. (See the examples below.)

The procedure is classifier agnostic. With certain classifiers, Naive Bayesian classifiers and Decision trees, the importance of variables can be directly concluded from their structure obtained after training.

The procedure can be enhanced by using dimension reduction before building the classifiers. (See the last section for an outline.)

### Implementation

The implementation of the procedure is straightforward in *Mathematica* -- see the package "VariableImportanceByClassifiers", [8].

The package can be imported with the command:

```
In[18]:= Import [
  "https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/
  master/VariableImportanceByClassifiers.m"]
```

At this point the package has only one function, `AccuracyByVariableShuffling`, that takes as arguments a `ClassifierFunction` object, a dataset, optional variable names, and the option "FScoreLabels" that allows the use of accuracies over a custom list of class labels instead of overall baseline accuracy.

Here is the function signature:

---

```
AccuracyByVariableShuffling[ clFunc_ClassifierFunction, testData_,
variableNames_:Automatic, opts:OptionsPattern[] ]
```

---

The returned result is an `Association` structure that contains the baseline accuracy and the accuracies corresponding to the shuffled versions of the dataset. I.e. steps 3 and 4 of the procedure are performed by `AccuracyByVariableShuffling`. Returning the result in the form `Association[___]` means we can treat the result as a list with named elements similar to the list structures in Lua and R.

Examples are given in the next section.

## Concrete applications

In this section we are going to split the datasets and find the accuracy decrease using the obtained with training data classifiers and the damaged versions of the test data.

### Titanic

First we partition the “Titanic” dataset into a training and testing sets.

```
In[20]:= titanicTrainingData =
  ExampleData[{"MachineLearning", "Titanic"}, "TrainingData"];
  Dimensions[titanicTrainingData][All, 1]

Out[21]= {916, 3}


In[22]:= titanicTestData = ExampleData[{"MachineLearning", "Titanic"}, "TestData"];
  Dimensions[titanicTestData][All, 1]

Out[23]= {393, 3}
```

**Remark:** We relied on `ExampleData` to produce dataset parts that are representative of the whole dataset. If the split is derived by other means it should be verified that the training and testing datasets have similar distributions of the observations. (For example, we can use `RecordsSummary`, [7], on each of them and compare the obtained values.)

We make a classifier with training data.

```
In[24]:= titanicCFunc = Classify[titanicTrainingData, Method → "SupportVectorMachine"]

Out[24]= ClassifierFunction[ Method SupportVectorMachine
  Number of classes 2]
```

Next we find the accuracies of the classifier function over the shuffled versions of the testing dataset.

```
In[25]:= AccuracyByVariableShuffling[titanicCFunc, titanicTestData, titanicVarNames]

Out[25]= <|None → 0.783715, passenger class → 0.709924,
  passenger age → 0.783715, passenger sex → 0.62341|>
```

The association value for the key `None` corresponds to the baseline accuracy. Since the accuracy corresponding to “passenger sex” is the worst we can conjecture that “passenger sex” is the most important variable. Confirmations of this conjecture are given in the next sections. The variables “passenger class” and “passenger age” *seem to be* on par.

**Remark:** With each run we would get (slightly) different results because of the random shuffling.

The default value of the argument `variableNames` is `Automatic`. If no variable names are specified then (currently) column indices are used in the result.

```
In[26]:= AccuracyByVariableShuffling[titanicCFunc, titanicTestData]

Out[26]= <|None → 0.783715, 1 → 0.70229, 2 → 0.778626, 3 → 0.603053|>
```

If we look at so called F-Scores produced by the classifier function we can see the class label “survived” is mis-classified more often “died”, and generally speaking a success rate of 0.7 (obtained with “SupportVectorMachine”) is not that great.

```
In[27]:= ClassifierMeasurements[titanicCFunc, titanicTestData, "FScore"]
```

```
Out[27]= <|died → 0.839925, survived → 0.666667|>
```

Since also the records summary table shows that records with the label “survived” are 38% (i.e. smaller than 50%) we might want to judge the importance of variables over the ability of the classifier to guess correctly the label “survived”. For this we use the option “FScoreLabels” of AccuracyByVariableShuffling.

```
In[28]:= accRes = AccuracyByVariableShuffling[titanicCFunc,
      titanicTestData, titanicVarNames, "FScoreLabels" → "survived"]
```

```
Out[28]= <|None → {0.666667}, passenger class → {0.558704},
      passenger age → {0.648438}, passenger sex → {0.409639}|>
```

We can see that “passenger sex” is still the most important, but now we also see that “passenger class” is significantly more important than “passenger age”. This can be made more obvious by looking into the relative damage effects:

```
In[29]:= (accRes[None] - accRes) / accRes[None]
```

```
Out[29]= <|None → {0.}, passenger class → {0.161943},
      passenger age → {0.0273437}, passenger sex → {0.385542}|>
```

Further investigation using mosaic plots and decision trees (see below) shows that female and first class passengers were much more likely to survive.

## Mushroom

Let us repeat the steps in the previous sub-section for the dataset “Mushroom”.

First let partition the “Mushroom” dataset into a training and testing sets.

```
In[30]:= mushroomTrainingData =
      ExampleData[{"MachineLearning", "Mushroom"}, "TrainingData"];
      Dimensions[mushroomTrainingData][All, 1]]
```

```
Out[31]= {5686, 22}
```

```
In[32]:= mushroomTestData = ExampleData[{"MachineLearning", "Mushroom"}, "TestData"];
      Dimensions[mushroomTestData][All, 1]]
```

```
Out[33]= {2438, 22}
```

We make a classifier with training data.

```
In[34]:= mushroomCFunc =
      Classify[mushroomTrainingData, Method -> "SupportVectorMachine"]
```

```
Out[34]= ClassifierFunction[  Method SupportVectorMachine  
Number of classes: 2]
```

Next we find and sort the accuracies of the classifier function over the shuffled versions of the testing dataset.



```
In[35]:= accRes = AccuracyByVariableShuffling[
  mushroomCFunc, mushroomTestData, mushroomVarNames];
Sort [
  accRes]
```

```
Out[36]= <|odor → 0.703856, spore-print-color → 0.986874, gill-size → 0.99918,
  bruises? → 1., cap-color → 1., cap-shape → 1., cap-surface → 1.,
  gill-attachment → 1., gill-color → 1., gill-spacing → 1.,
  habitat → 1., population → 1., ring-number → 1., ring-type → 1.,
  stalk-color-above-ring → 1., stalk-color-below-ring → 1.,
  stalk-root → 1., stalk-shape → 1., stalk-surface-above-ring → 1.,
  stalk-surface-below-ring → 1., veil-color → 1., veil-type → 1., None → 1. |>
```

We can see that “odor” seems most decisive and this makes sense. It is further clarified with mosaic plot below.

If we look at so called F-Scores produced by the classifier function we can see that for both class labels “edible” and “poisonous” we get perfect classification scores:

```
In[37]:= ClassifierMeasurements[mushroomCFunc, mushroomTestData, "FScore"]
```

```
Out[37]= <|edible → 1., poisonous → 1. |>
```

Hence there is no need to compute more specialized accuracies as it was done for “Titanic”.

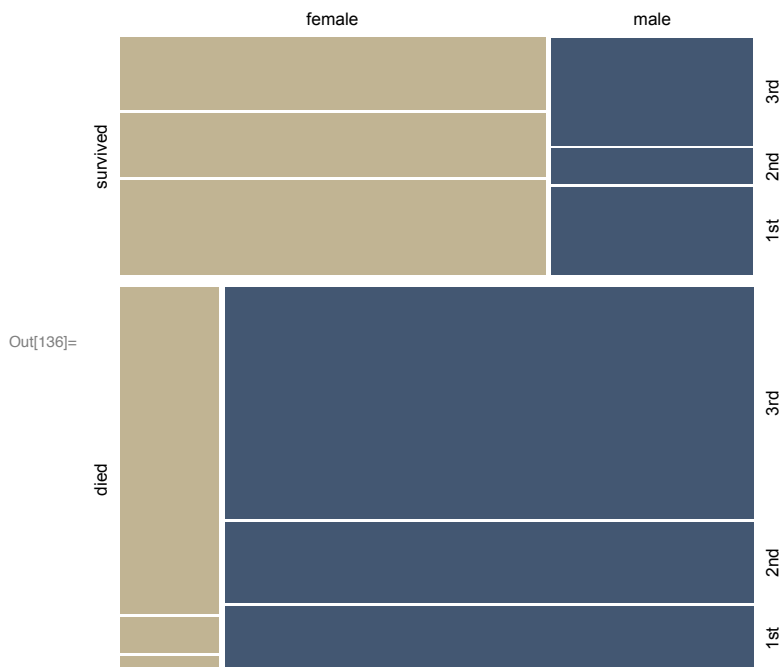
## Confirmations and explanations with Mosaic plots

Mosaic plots [5,6] give visual representation of conditional probabilities in categorical data. We can say that Mosaic plots correspond to NBC's, so it does make a lot of sense to use them for variable importance confirmation.

### Titanic

Here is a mosaic plot for the dataset "Titanic" without the column "passenger age" (which is not categorical):

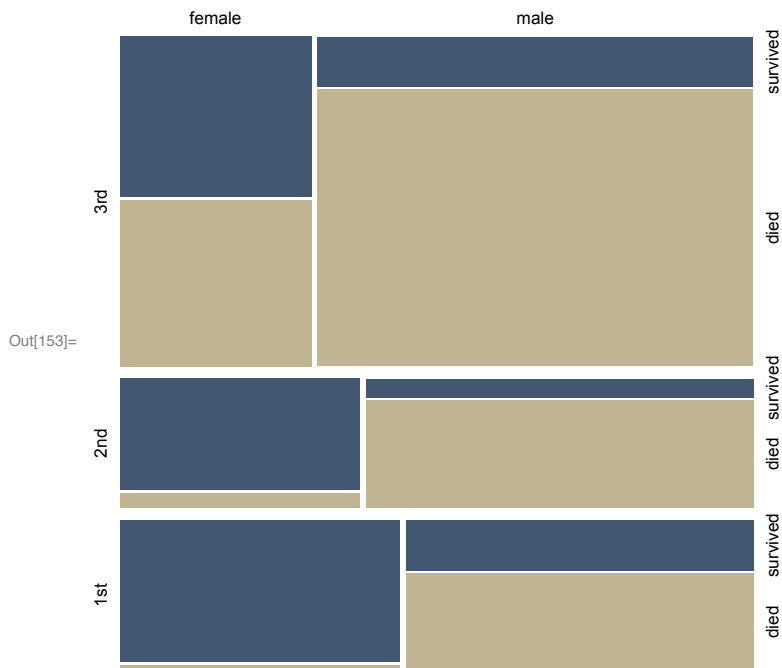
```
In[136]:= MosaicPlot[titanicDataset[All, {4, 3, 1}],  
  ColorRules -> {2 -> ColorData[7, "ColorList"]}]
```



From the previous plot it is immediately obvious why "passenger sex" is so decisive. The plot is made from the conditional probabilities direction starting from the class labels. For example, the plot answers questions like "given that a person survived, what is the probability of (i) being female, and (ii) being female and 1st class". The `Tooltip` tables show the answers are 0.678 and 0.278 respectively.

Let us make another mosaic plot that is structured and colored in such a way so we can follow which conditions led to death or survival.

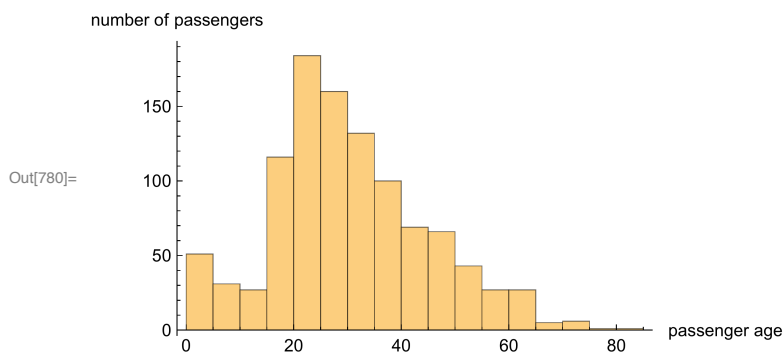
```
In[153]:= MosaicPlot[titanicDataset[All, {1, 3, 4}],
  ColorRules -> {3 -> ColorData[7, "ColorList"]}]]
```



We can see that the blue rectangles that correspond to survival are much larger for females and especially for females from 1st and 2nd class.

Here is the distribution of the passenger ages:

```
In[780]:= Histogram[titanicDataset[All, 2], Automatic,
  AxesLabel -> {"passenger age", "number of passengers"}]
```

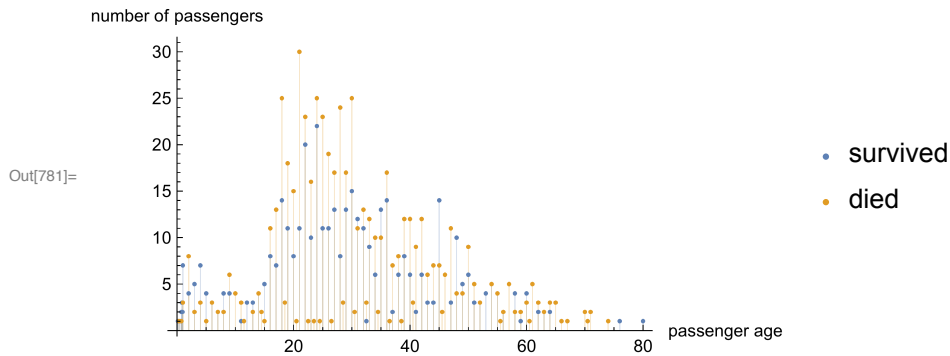


One way to visualize the survival dependence of age is the following.

```

In[781]:= ListPlot[{
  Tooltip[Tally[Pick[titanicDataset[All, 2]],
    # == "survived" & /@titanicDataset[All, 4]]], "survied"], Tooltip[
    Tally[Pick[titanicDataset[All, 2]], # == "died" & /@titanicDataset[All, 4]]],
    "died"]], Filling -> Axis,
  PlotRange -> All, PlotLegends -> {"survived", "died"},
  AxesLabel -> {"passenger age", "number of passengers"}]

```



From the plot we can see that age-wise more or less survival and death followed the distribution of the ages of all passengers.

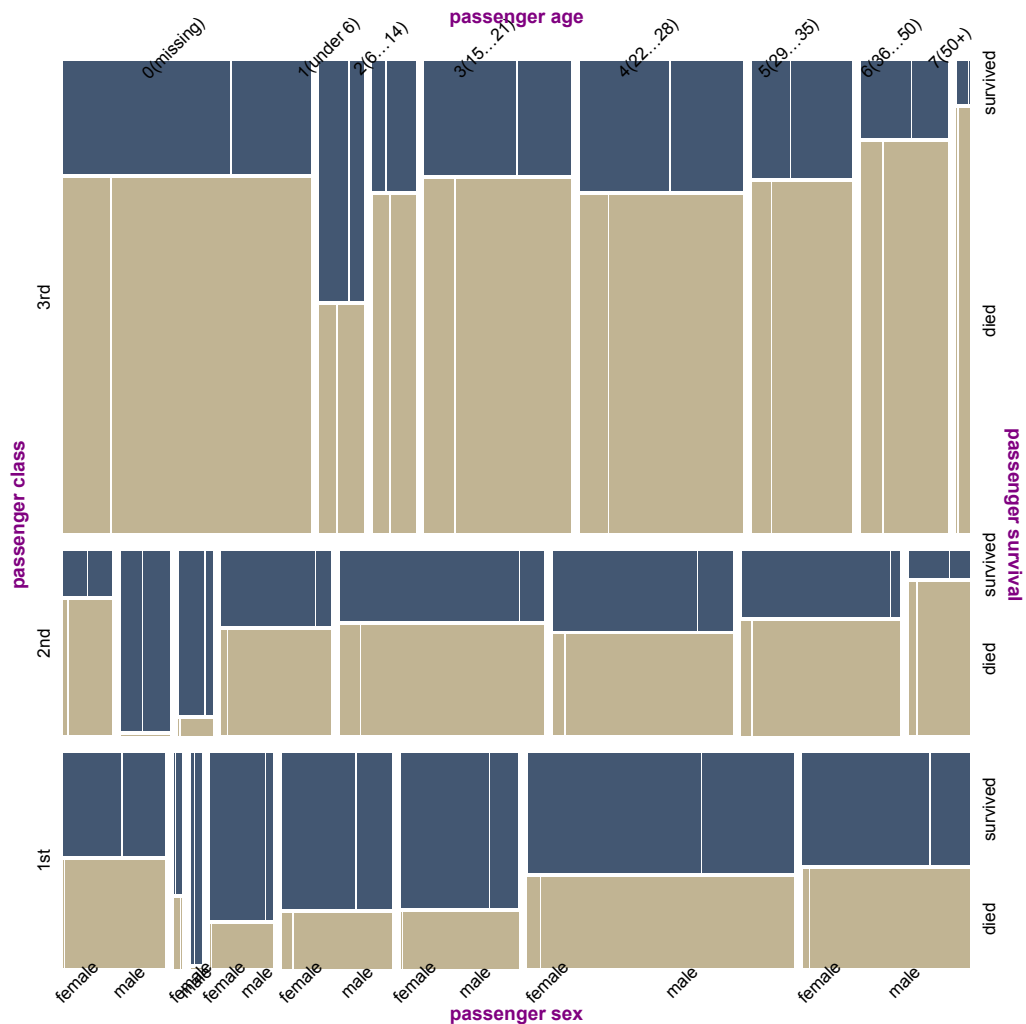
At this point it is better to turn age into a categorical variable and visualize with a mosaic plot. One way to do this is to use quantiles of ages; another is to use predefined age ranges. We are going to use the latter approach.

```
In[38]:= titanicDatasetCatAge = titanicDataset;
```

```
ageQF = {
  1 -∞ < #1 ≤ 5
  2 5 < #1 ≤ 14
  3 14 < #1 ≤ 21
  4 21 < #1 ≤ 28
  5 28 < #1 ≤ 35
  6 35 < #1 ≤ 50
  7 50 < #1 ≤ ∞
  0 True
  &;
```

```
titanicDatasetCatAge[All, 2] =
  Map[If[MissingQ[#], 0, ageQF[#]] &, titanicDatasetCatAge[All, 2]] /.
    {1 -> "1 (under 6)", 2 -> "2 (6...14)", 3 -> "3 (15...21)", 4 -> "4 (22...28)",
     5 -> "5 (29...35)", 6 -> "6 (36...50)", 7 -> "7 (50+)", 0 -> "0 (missing)"};
MosaicPlot[titanicDatasetCatAge[All, {1, 2, 4, 3}],
  ColorRules -> {3 -> ColorData[7, "ColorList"]},
  "LabelRotation" -> {{0.5, 0.5}, {0, 1}},
  "ColumnNames" ->
    Map[Style[#, Larger, Bold, Purple] &, titanicVarNames[{1, 2, 4, 3}]]]
```

```
Out[41]=
```



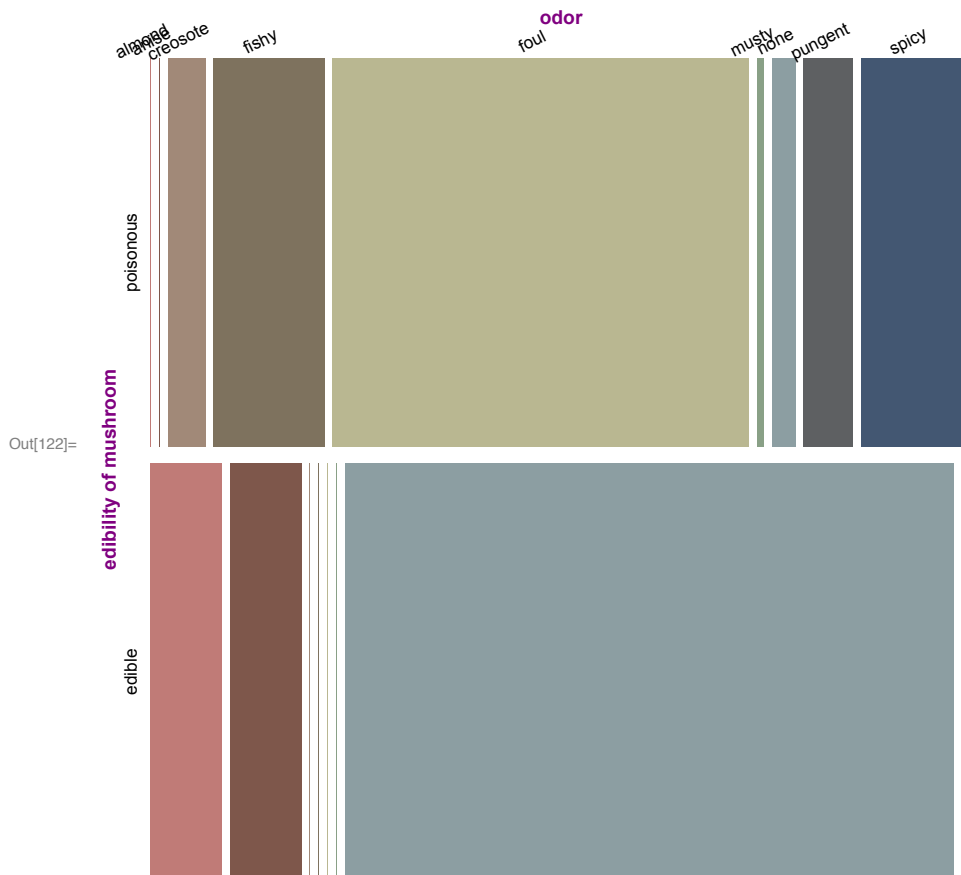
From the plot we can see that the very young were much more likely to survive in the 1st and 2nd class. Note the large amount of missing ages. The missing data might be one of the reasons “passenger age” is not that decisive.

The conversion of “passenger age” into a categorical variable is also useful for the application of Association rules and topic extraction. (See below.)

## Mushroom

Mosaic plots for the “Mushroom” dataset over the columns “edibility of mushroom” and “odor” immediately explain why “odor” is so decisive for the mushroom classification. For example, we can see that a large fraction,  $\approx 81\%$ , of the edible mushrooms have no odor, and only  $\approx 3\%$  of the poisonous mushrooms have no odor. (These fractions can be seen on the `Tooltip` tables when the mouse pointer is over the rectangles in the first plot below.)

```
In[122]:= MosaicPlot[mushroomDataset[All, {-1, 5}],
  "LabelRotation" -> {{1, 0.5}, {0, 1}}, "ColumnNames" ->
  Map[Style[#, Larger, Bold, Purple] &, mushroomVarNames[{-1, 5}]]]
```

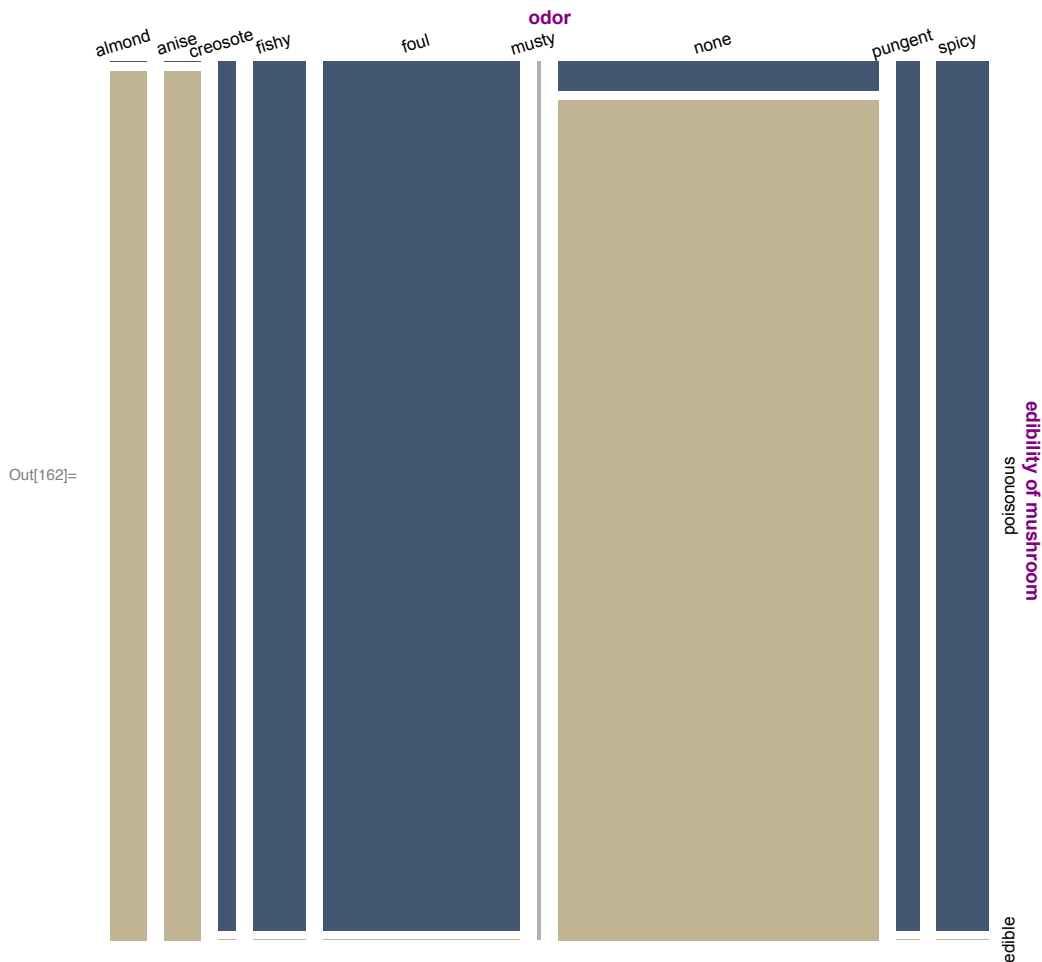


As with “Titanic” we can make a mosaic plot taking a different perspective of the conditional probabilities, showing which conditions lead to edible or poisonous. (Note the variable order and the specified coloring for the second axis in the value given to “ColorRules”.)

```

In[162]:= MosaicPlot[mushroomDataset[[All, {5, -1}]], "FirstAxis" → "Top",
  "LabelRotation" → {{1, 0.3}, {0, 1}}, "ColumnNames" →
    Map[Style[#, Larger, Bold, Purple] &, mushroomVarNames[[{5, -1}]],
  ColorRules → {2 -> ColorData[7, "ColorList"]}]]

```



In the second plot the blue rectangles correspond to “poisonous”. We can see that there is almost no overlap between the odors of edible with the odors of poisonous.

## Confirmations and explanations with Decision trees

Let us look into some decision trees built over the whole datasets in order to verify the found variable importance using the impurity functions of the decision tree building algorithm (based on Information entropy and Gini coefficient). For more details see [3,9].

The decision trees we make a deliberately short. It is expected that the most important variables are going to be used first for splitting.

This command loads the package [3]:

```
In[42]:= Import [
  "https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/
  master/AVCDecisionTreeForest.m"]
```

### Titanic

The following command builds a decision tree over the whole "Titanic" dataset. The leaf count shows that the tree is small (short).

```
dtree = BuildDecisionTree[titanicDataset /. {_Missing -> 0},
  {12, 0.02}, "ImpurityFunction" -> "Gini", "PreStratify" -> True];
LeafCount [
  dtree]
38
```

The package [3] provides functions to assess the classification rates of decision trees and forests. Since we used the whole dataset the invocation of these functions is just for sanity checks.

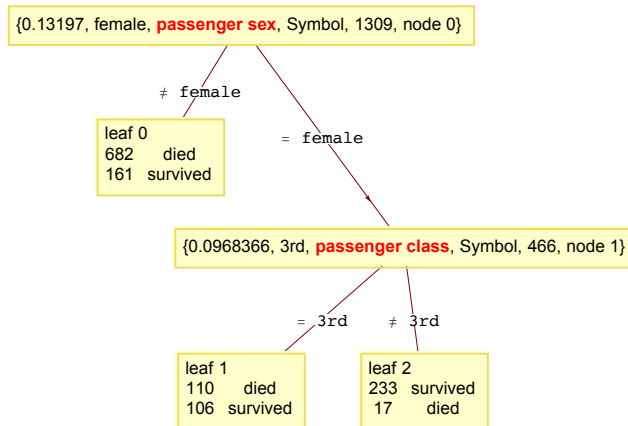
```
DecisionTreeClassificationSuccess [
  dtree, Map[Flatten, List @@@ titanicTestData]]
{{died, True} -> 0.979424, {died, False} -> 0.0205761,
 {survived, True} -> 0.453333, {survived, False} -> 0.546667,
 {All, True} -> 0.778626, {All, False} -> 0.221374}
```

Because of the rules {survived,True}→0.453333 and {survived,False}→0.546667 we see that the tree does not classify that well for the label "survived". The reasons are obvious by looking at the tree plot below.

The following command converts the obtained decision tree into a list of rules suitable for graph visualization and plots the corresponding graph.



```
LayeredGraphPlot[
  DecisionTreeToRules[dtree] /. {{imp_?NumberQ, x_, i_Integer, b__} :>
    {imp, x, Style[titanicVarNames[[i]], Red, Bold], b}}, VertexLabeling -> True]
```



In the plot:

1. Each non-leaf node of the tree has the format:

```
{impurity, splitting value, splitting variable,
  variable type, number of rows, node label}.
```

Where "variable type" is either Number or Symbol, and "number of rows" refers to the size of the part of the data that was observed (scanned) at that point of splitting.

2. The leaf nodes are numbered. The second and third rows of a leaf show the number of records and labels corresponding to the predicate used to obtain the leaf. For example, for the predicate

```
"passenger sex" = "female" ∧ "passenger class" = "3 rd"
```

the obtained subset of data has 110 records with "died" and 106 records with "survived".

We can see from the tree that (i) for males the probability to survive was  $\approx 19\%$  and that (ii) the females from 1st and 2nd class had much higher probability to survive. This confirms our findings with the classifier based procedure applied above.

## Mushroom

The following command builds a decision tree over the whole "Mushroom" dataset. The leaf count shows that the tree is small (short).

```
dtree = BuildDecisionTree[mushroomDataset /. {_Missing -> None},
  {3000, 0.03}, "ImpurityFunction" -> "Entropy"];
LeafCount[
  dtree]
```

74

Again, as in the previous sub-section, let us do a sanity check using a classification success rates computation function.

```

DecisionTreeClassificationSuccess[
  dtree, Map[Flatten, List@@mushroomDataset]]
{{edible, True} → 1., {edible, False} → 0.,
 {poisonous, True} → 0.850868, {poisonous, False} → 0.149132,
 {All, True} → 0.928114, {All, False} → 0.0718858}

```

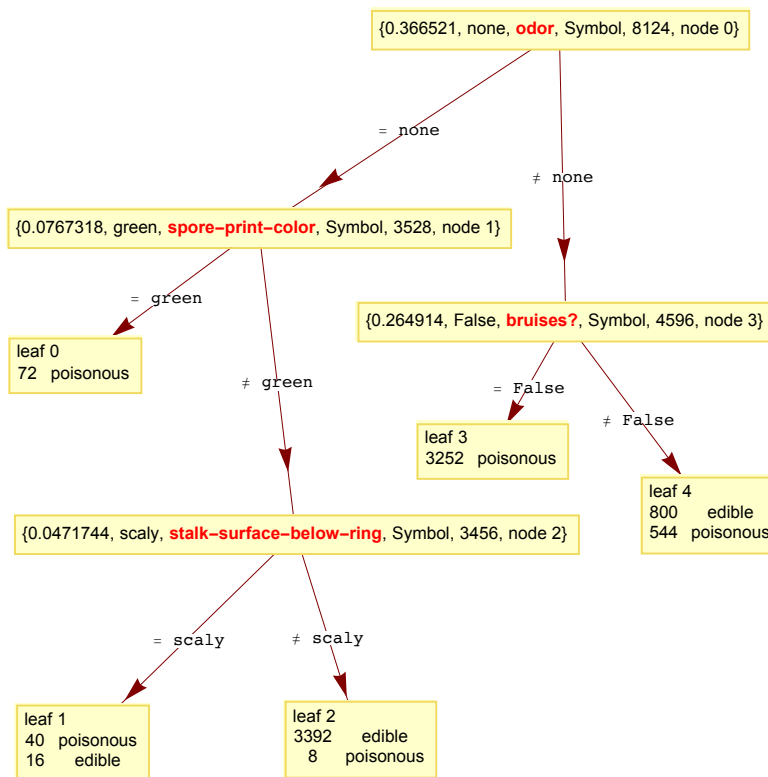
We see that the tree classifies very well for both labels. (We have high values for all rules of the form  $\{_, \text{True}\} \rightarrow _.$ )

The following command converts the obtained decision tree into a list of rules suitable for graph visualization and plots the corresponding graph.

```

LayeredGraphPlot[
  DecisionTreeToRules[dtree] /. {{imp_?NumberQ, x_, i_Integer, b__} :> {imp, x,
    Style[mushroomVarNames[i], Red, Bold], b}}, VertexLabeling → True]

```



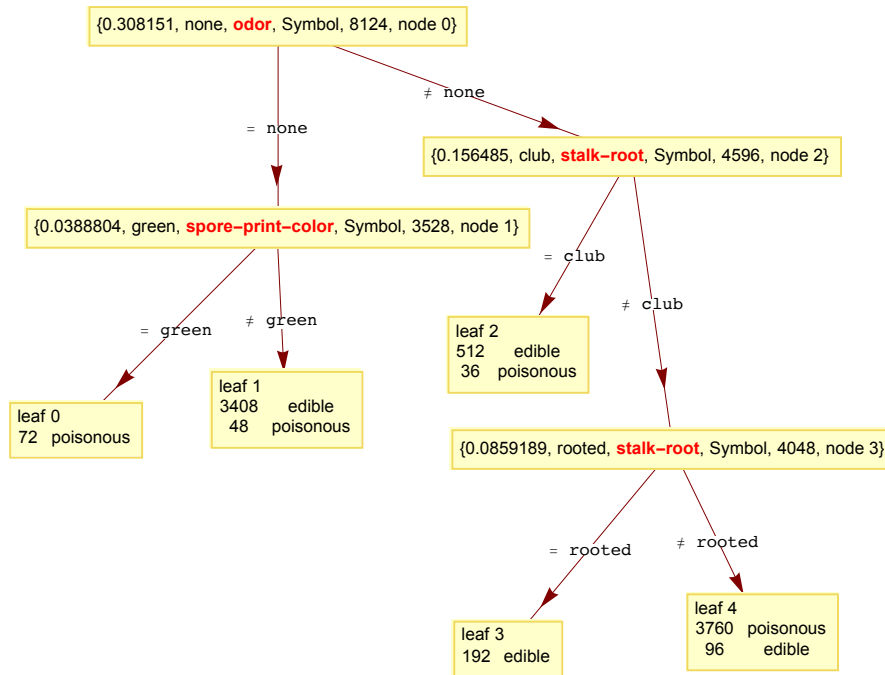
The plot of the obtained decision tree shows “odor” as the first, most significant variable having a large corresponding impurity value. (This should be expected having seen the “Mushroom” mosaic plot.)

We would get a different tree if we use “ImpurityFunction”→“Gini”. The variable “odor” is still the most decisive one.

```

dtree = BuildDecisionTree[mushroomDataset /. {_Missing -> 0},
  {3000, 0.03}, "ImpurityFunction" -> "Gini"];
LayeredGraphPlot[DecisionTreeToRules[dtree] /.
  {{imp_?NumberQ, x_, i_Integer, b__} :> {imp, x,
    Style[mushroomVarNames[[i]], Red, Bold], b}}, VertexLabeling -> True]

```



## Association rules

We can use Association rules learning, [10,11], to find out which values of which variables imply the different class labels with highest confidence. This is demonstrated also in [2]. This approach produces frequent sets and rules for concrete values of the variables. In order to derive (or imply) variable importance we do the following.

1. For each class label  $L_i$  select a subset of association rules with high confidence that have  $L_i$  as a consequent.
2. Compute statistics for the variables of the concrete values in the antecedents of the selected rules.
3. Variables that appear most often in the antecedents of the rules implying  $L_i$  are most decisive for that class label.

Instead of subsets of association rules we can use frequent sets and perform statistics on them. The frequent sets are the required step to find association rules, and computing the association rules might be time consuming. A similar procedure using frequent sets follows.

1. For each class label  $L_i$  select a subset of frequent sets that have  $L_i$  are that have large enough frequencies (support).
2. Compute statistics for the variables of the concrete values in the frequent sets.
3. Variables that appear most often in the frequent sets containing  $L_i$  are most decisive for that class label.

This command loads the package AprioriAlgorithm.m, [12]:

```
In[827]:= Import [
  "https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/
  master/AprioriAlgorithm.m"]
```

**Remark:** Frequent sets and association rules learning algorithms do not require array shaped data; they can work on general sets of metadata tags. (Lists of lists of strings.) That is why below the concrete values of the variables of the considered datasets ("Titanic" and "Mushroom") are called "tags".

Extended examples and explanations for the use of Association rules are given in [11].

## Titanic

The following computation finds frequent sets of tags that appear in at least 2% of the records. After the computation a table shows the tally of the lengths of the found frequent sets.

```
In[828]:=  $\mu$  = 0.02;
Print["Number of records corresponding to  $\mu$ =",
 $\mu$ , ":", Length[titanicDatasetCatAge] *  $\mu$ ]
Print["Computation time:", AbsoluteTiming[
  {aprioriRes, itemToIDRules, idToItemRules} =
    AprioriApplication[titanicDatasetCatAge,  $\mu$ ];
] [[1]]];
Grid[Prepend[Tally[Map[Length, Join@@aprioriRes]],
  {"frequent set\nlength", "number of\nfrequent sets"}],
  Dividers  $\rightarrow$  {None, {True, True}}]
```

Number of records corresponding to  $\mu=0.02$ : 26.18

Computation time:0.033345

frequent set length	number of frequent sets
1	15
2	61
3	57
4	13

Note that the command above uses the version of “Titanic” in which the numerical variable “passenger age” was replaced with a categorical representation.

Next we find association rules from the frequent sets that contain “survived”. We are asking for association rules that have confidence at least 70% and have support in at least 2% of the records.

```
In[832]:= items = {"survived"};
itemRules = ItemRules[titanicDatasetCatAge,
  aprioriRes, itemToIDRules, idToItemRules, items, 0.7, 0.02];
```

The following command tabulated those of the rules that have “survived” as a consequent. The rules are sorted according to their confidence.

```
In[834]:= Magnify[#, 0.7] &@Grid[
  Prepend[
    SortBy[Select[Join@@itemRules, MemberQ[items, #[-1, 1]]] &&
      #[[2]] > 0.7 && 2 ≤ Length[#[-2]] ≤ 10 &], -#[[2]] &],
    Map[Style[#, Blue, FontFamily → "Times"] &, {"Support", "Confidence",
      "Lift", "Leverage", "Conviction", "Antecedent", "Consequent"}]
  ], Alignment → Left]
```

Support	Confidence	Lift	Leverage	Conviction	Antecedent	Consequent
0.106188	0.965278	2.5271	0.0641682	17.7992	{1st, female}	{survived}
0.0290298	0.95	2.4871	0.0173576	12.3606	{1st, 6(36...50), female}	{survived}
0.0221543	0.90625	2.37256	0.0128166	6.59231	{7(50+), female}	{survived}
0.0718105	0.886792	2.32162	0.0408794	5.45926	{2nd, female}	{survived}
0.0412529	0.794118	2.079	0.0214102	3.00186	{5(29...35), female}	{survived}
0.0481283	0.768293	2.01139	0.0242004	2.66728	{6(36...50), female}	{survived}
0.0496562	0.747126	1.95598	0.0242693	2.44402	{4(22...28), female}	{survived}
0.0236822	0.738095	1.93233	0.0114264	2.35975	{1st, 4(22...28)}	{survived}
0.0252101	0.733333	1.91987	0.0120789	2.31761	{1st, 5(29...35)}	{survived}
0.0374332	0.710145	1.85916	0.0172987	2.1322	{3(15...21), female}	{survived}

We can see that the highest confidence of survival is associated with the tag “female”. The variable “passenger sex” that has values with that tag should be very decisive. We can confirm this by finding the high confidence association rules that contain “death”. We can see that almost all antecedents have the tag “male”.

```

In[835]:= items = {"died"};
itemRules = ItemRules[titanicDatasetCatAge,
  aprioriRes, itemToIDRules, idToItemRules, items, 0.7, 0.02];
Magnify[#, 0.7] &@Grid[
  Prepend[
    SortBy[Select[Join@@itemRules, MemberQ[items, #[-1, 1]] &&
      #[[2]] > 0.7 && 2 ≤ Length[#[-2]] ≤ 10 &], -#[[2]] &],
    Map[Style[#, Blue, FontFamily → "Times"] &, {"Support", "Confidence",
      "Lift", "Leverage", "Conviction", "Antecedent", "Consequent"}]
  ], Alignment → Left]

```

Support	Confidence	Lift	Leverage	Conviction	Antecedent	Consequent
0.0236822	0.96875	1.56748	0.00857377	12.2231	{2nd, 6(36...50), male}	{died}
0.0282659	0.925	1.49669	0.00938032	5.09295	{2nd, 4(22...28), male}	{died}
0.0351413	0.901961	1.45941	0.0110623	3.8961	{3rd, 6(36...50), male}	{died}
0.0977846	0.888889	1.43826	0.0297967	3.43774	{0(missing), 3rd, male}	{died}
0.0565317	0.870588	1.40865	0.0163999	2.95159	{3(15...21), 3rd, male}	{died}
0.0756303	0.868421	1.40515	0.0218065	2.90298	{3(15...21), male}	{died}
0.0236822	0.861111	1.39332	0.00668522	2.75019	{2nd, 5(29...35), male}	{died}
0.121467	0.859459	1.39065	0.0341212	2.71787	{0(missing), male}	{died}
0.0412529	0.857143	1.3869	0.0115082	2.6738	{7(50+), male}	{died}
0.111536	0.853801	1.38149	0.0307999	2.61268	{2nd, male}	{died}
0.319328	0.84787	1.37189	0.0865636	2.51082	{3rd, male}	{died}
0.0466005	0.835616	1.35207	0.0121344	2.32366	{3rd, 6(36...50)}	{died}
0.0626432	0.828283	1.3402	0.0159015	2.22442	{3rd, 4(22...28), male}	{died}
0.0909091	0.82069	1.32791	0.022449	2.13022	{6(36...50), male}	{died}
0.0985485	0.811321	1.31276	0.0234785	2.02445	{4(22...28), male}	{died}
0.038961	0.796875	1.28938	0.00874419	1.88047	{3rd, 5(29...35), male}	{died}
0.0221543	0.783784	1.2682	0.0046852	1.76662	{1st, 7(50+), male}	{died}
0.0718105	0.783333	1.26747	0.0151539	1.76294	{5(29...35), male}	{died}
0.120703	0.759615	1.22909	0.0224981	1.589	{0(missing), 3rd}	{died}
0.0710466	0.756098	1.2234	0.0129736	1.56608	{3(15...21), 3rd}	{died}
0.0481283	0.75	1.21354	0.00846873	1.52788	{3rd, 5(29...35)}	{died}
0.0756303	0.722628	1.16925	0.0109473	1.37711	{3rd, 4(22...28)}	{died}

## Mushroom

Before applying the Apriori algorithm to the “Mushroom” dataset let us modify its values to include the variable (column) names.

```

In[838]:= mushroomDatasetWithVarNames =
  Map[ToString, mushroomDataset /. _Missing → "NA", {-1}];
mushroomDatasetWithVarNames =
  Transpose[MapThread[Function[{col, vn}, Map[vn <> ":" <> # &, col]],
    {Transpose[mushroomDatasetWithVarNames], mushroomVarNames}]];
mushroomDatasetWithVarNames = mushroomDatasetWithVarNames /.
  {"edibility of mushroom:edible" → "edible",
    "edibility of mushroom:poisonous" → "poisonous"};

```

A row of this data looks like this:

```

In[841]:= mushroomDatasetWithVarNames[[12]]
Out[841]= {cap-shape:convex, cap-surface:scaly, cap-color:yellow, bruises?:True,
  odor:almond, gill-attachment:free, gill-spacing:close, gill-size:broad,
  gill-color:brown, stalk-shape:enlarging, stalk-root:club,
  stalk-surface-above-ring:smooth, stalk-surface-below-ring:smooth,
  stalk-color-above-ring:white, stalk-color-below-ring:white,
  veil-type:partial, veil-color:white, ring-number:one, ring-type:pendant,
  spore-print-color:black, population:scattered, habitat:meadows, edible}

```

The following computation finds frequent sets of 3 or less tags that appear in at least 5% of the records.

After the computation a table shows the tally of the lengths of the found frequent sets.

```
In[842]:=  $\mu = 0.05$ ; mt = 3;
Print["Number of records corresponding to  $\mu =$ ",
 $\mu$ , ": ", Length[mushroomDataset] *  $\mu$ ]
Print["Computation time:", AbsoluteTiming[
  {aprioriRes, itemToIDRules, idToItemRules} = AprioriApplication[
    mushroomDatasetWithVarNames,  $\mu$ , "MaxNumberOfItems" → mt];
  ]][1]];
Grid[Prepend[Tally[Map[Length, Join@@aprioriRes]],
  {"frequent set\nlength", "number of\nfrequent sets"}],
  Dividers → {None, {True, True}}]
```

Number of records corresponding to  $\mu=0.05$ : 406.2

Computation time:9.24365

frequent set length	number of frequent sets
1	73
2	1329
3	10 618

Next we find association rules from the frequent sets that contain “edible”. We are asking for association rules that have confidence at least 70% and have support in at least 20% of the records.

```
In[846]:= AbsoluteTiming[
  items = {"edible"};
  itemRules = ItemRules[mushroomDatasetWithVarNames,
    aprioriRes, itemToIDRules, idToItemRules, items, 0.7, 0.2];
]
```

Out[846]= {1.28314, Null}

Next we pick the associations rules with antecedent length 1.

```
In[847]:= Magnify[#, 0.7] &@Grid[
  Prepend[
    SortBy[Select[Join@@itemRules, MemberQ[items, #[-1, 1]] && #[-2]] > 0.7 &&
      1 ≤ Length[#[-2]] ≤ 1 && Length[#[-1]] == 1 &], -#[-2] &],
    Map[Style[#, Blue, FontFamily → "Times"] &, {"Support", "Confidence",
      "Lift", "Leverage", "Conviction", "Antecedent", "Consequent"}]
  ], Alignment → Left]
```

Support	Confidence	Lift	Leverage	Conviction	Antecedent	Consequent
0.419498	0.965986	1.86494	0.194559	14.1716	{odor:none}	{edible}
0.214673	0.886179	1.71086	0.0891965	4.23497	{spore-print-color:brown}	{edible}
0.202856	0.880342	1.6996	0.0835004	4.02838	{spore-print-color:black}	{edible}
0.338749	0.815166	1.57377	0.123502	2.6079	{bruises?:True}	{edible}
0.387986	0.794355	1.53359	0.134994	2.34398	{ring-type:pendant}	{edible}
0.448055	0.703246	1.35769	0.118043	1.62434	{stalk-surface-above-ring:smooth}	{edible}

What is in the table above is similar to what we observed with mosaic plots for “Mushroom” for the value “none” of the variable “odor”. Using the code above we can examine rules with larger number of tags in their antecedents or consequents.

Let us find association rules for the class label “poisonous”.

```

In[848]:= AbsoluteTiming[
  items = {"poisonous"};
  itemRules = ItemRules[mushroomDatasetWithVarNames,
    aprioriRes, itemToIDRules, idToItemRules, items, 0.7, 0.20];
]

Out[848]= {1.22516, Null}

In[849]:= Magnify[#, 0.7] &@Grid[
  Prepend[
    SortBy[Select[Join@@itemRules, MemberQ[items, #[-1, 1]] && #[-2] > 0.8 &&
      1 ≤ Length[#[-2]] ≤ 1 && Length[#[-1]] == 1 &], -#[-2] &],
    Map[Style[#, Blue, FontFamily → "Times"] &, {"Support", "Confidence",
      "Lift", "Leverage", "Conviction", "Antecedent", "Consequent"}]
  ], Alignment → Left]

Out[849]=


| Support  | Confidence | Lift    | Leverage | Conviction | Antecedent                       | Consequent  |
|----------|------------|---------|----------|------------|----------------------------------|-------------|
| 0.212703 | 1.         | 2.07457 | 0.110174 | 1000.      | {gill-color:buff}                | {poisonous} |
| 0.265879 | 1.         | 2.07457 | 0.137718 | 1000.      | {odor:foul}                      | {poisonous} |
| 0.274249 | 0.939292   | 1.94862 | 0.133509 | 8.53214    | {stalk-surface-above-ring:silky} | {poisonous} |
| 0.265879 | 0.9375     | 1.94491 | 0.129174 | 8.28754    | {stalk-surface-below-ring:silky} | {poisonous} |
| 0.273757 | 0.88535    | 1.83672 | 0.12471  | 4.51786    | {gill-size:narrow}               | {poisonous} |


```

Association rules that have confidence 1 are logical rules -- see the summary table in the next sub-section. For example, from the table above we can see that mushrooms with foul odor are 100% poisonous. Again this is something we observed with the mosaic plots for “Mushroom”.

## Associations rules measures summary

The following tables are also given in [11] with more detailed corresponding explanations.

	measure	definition	interpretation	
	support	$\text{supp}_T(A \Rightarrow B)$	$P(A \cap B)$	
	confidence	$\text{conf}_T[A \Rightarrow B] := \text{supp}_T[A \Rightarrow B] / \text{supp}_T[A]$	$P(B / A)$	
Out[853]=	lift	$\frac{\text{conf}_T[A \Rightarrow B]}{\text{supp}_T[B]}$	$\frac{P(B / A)}{P(B)}$	
	leverage	$\text{supp}_T(A \Rightarrow B) - \text{supp}_T(A) \text{supp}_T(B)$	$P(A \cap B) - P(A) P(B)$	
	conviction	$\frac{1 - \text{supp}_T(B)}{1 - \text{conf}_T(A \Rightarrow B)}$	$\frac{1 - P(B)}{1 - P(B / A)}$	
	measure	min value, incompatibility	value at independance	max value, logical rule
	support	0	$\text{supp}_T(A) \text{supp}_T(B)$	$\text{supp}_T(A)$
Out[854]=	confidence	0	$\text{supp}_T(B)$	1
	lift	0	1	$\frac{1}{\text{supp}_T(B)}$
	leverage	$-\text{supp}_T(A) \text{supp}_T(B)$	0	$\text{supp}_T(A) (1 - \text{supp}_T(B))$
	conviction	$1 - \text{supp}_T(B)$	1	$\infty$



## Confirmations and explanations with Dimension reduction

Somewhat similar -- but orthogonal in character -- to the application of Association rules learning we can apply matrix factorization algorithms in a manner used in Principal Components Analysis (PCA) and Latent Semantic Analysis (LSA) for natural language texts. In LSA with matrix factorizations we obtain topics of words and statistical thesaurus entries; see [13,14] for matrix factorization applied to podcast transcripts.

See the last section for using dimension reduction before the building the classifiers.

### Topic extraction of linear vector space representations

Applying Non-Negative Matrix Factorization (NNMF) to the “Titanic” dataset produces easy to interpret topics (change of basis vectors). With larger datasets the interpretation of the results from the variable importance perspective is not that straightforward.

The main difference between NNMF application shown here and PCA is that PCA uses orthogonal transformations of the linear vector space representation of the data, like Singular Value Decomposition (SVD). PCA is good for numerical data, but not that good for categorical data (like the datasets considered in this document). We can apply PCA using the built-in functions `SingularValueDecomposition` or `DimensionReduction`. (See the next sub-section.)

First, as we did for the application of Association rules learning, we concatenate the variable names with the dataset values.

```
In[855]:= data = Map[ToString, titanicDatasetCatAge /. _Missing -> "NA", {-1}];
data = Transpose[MapThread[Function[{col, vn}, Map[vn <> ":" <> # &, col]],
  {Transpose[data], titanicVarNames}]];
```

We can see each row of the dataset as a “bag of words”. Next we represent the dataset with a matrix that can be seen as a linear operator that maps passengers to a space of tags (in this case passenger class, sex, age, and survival).

This commands loads a package useful for LSA vector space models [15].


```
In[757]:= Import [
  "~/MathFiles/MathematicaForPrediction/DocumentTermMatrixConstruction.m"]
```

The package [15] has a function, `DocumentTermMatrix`, that can be used to convert a list of lists of tags (“bags of words”) into a matrix in which every column corresponds to an unique tag (“word”).

```
In[857]:= {TM, words} = DocumentTermMatrix[data, {}, {}];
```

Here is the matrix summary:

```
In[858]:= TM
```

```
Out[858]:= SparseArray[ Specified elements 5236  
Dimensions {1309, 15}]
```

Here is table that shows the matrix columns tag interpretation and their corresponding number of non-zero elements:

```
In[859]:= Magnify[#, 0.8] &@GridTableForm[
  Transpose[{words, Total[TM]}], TableHeadings -> {"word", "count"}]
```

```
Out[859]=
```

#	word	count
1	passenger age:0 (missing)	263
2	passenger age:1 (under 6)	56
3	passenger age:2 (6...14)	51
4	passenger age:3 (15...21)	183
5	passenger age:4 (22...28)	246
6	passenger age:5 (29...35)	188
7	passenger age:6 (36...50)	227
8	passenger age:7 (50+)	95
9	passenger class:1st	323
10	passenger class:2nd	277
11	passenger class:3rd	709
12	passenger sex:female	466
13	passenger sex:male	843
14	passenger survival:died	809
15	passenger survival:survived	500

The matrix  $TM \in \mathbb{R}^{1309 \times 15}$  that is the vector space representation of the dataset “Titanic”. The  $i$ -th row of the matrix  $TM$  corresponds to the  $i$ -th Titanic passenger and the values of the row (which are either 0 or 1) are the coefficients for the basis vectors corresponding to the variable values shown in the table above.

The following command loads the package [15] for performing NMF:

```
In[761]:= Import [
  "~/MathFiles/MathematicaForPrediction/NonNegativeMatrixFactorization.m"]
```

With the following command we apply an NMF algorithm to reduce the dimension of the original matrix  $TM$  from having 15 columns to having 2.

```
In[762]:= {W, H} = GDCLS[TM, 2];
```

At this point the matrix  $TM \in \mathbb{R}^{1309 \times 15}$  is approximated with the matrix-matrix product  $WH$ ,  $W \in \mathbb{R}^{1309 \times 2}$ ,  $H \in \mathbb{R}^{2 \times 15}$ . The entries of the matrices  $W$  and  $H$  are non-negative. The applied algorithm finds such  $W$  and  $H$  that (roughly speaking) a local minimum of  $\|TM - WH\|_2$  is obtained. The rows of the matrix  $H$  represent the two topics to which the basis of 15 “words” of the dataset was reduced.

In order to do the topic interpretation we have to normalize the product  $WH$  in such a way that the rows of  $H$  have norms 1. Then each row of the matrix  $W$  (i.e. each passenger) has the coefficients for the new basis of two topics.

```
In[763]:= {W, H} = RightNormalizeMatrixProduct[W, H];
Norm /@ H
```

```
Out[764]= {1., 1.}
```

Finally, we tabulate the extracted two topics:

```
In[765]:= Magnify[#, 0.8] & /@Map[GridTableForm[BasisVectorInterpretation[#, 15, words],
TableHeadings -> {"score", "tag"}] &, Normal[H]]
```

#	score	tag	#	score	tag
1	0.657989	passenger survival:survived	1	0.604266	passenger survival:died
2	0.61337	passenger sex:female	2	0.599553	passenger sex:male
3	0.298565	passenger class:1st	3	0.450227	passenger class:3rd
4	0.157363	passenger class:3rd	4	0.161146	passenger age:0 (missing)
5	0.149261	passenger class:2nd	5	0.113584	passenger age:4 (22...28)
6	0.128667	passenger age:6 (36...50)	6	0.0894026	passenger class:2nd
7	0.114136	passenger age:4 (22...28)	7	0.0891074	passenger age:3 (15...21)
8	0.0980145	passenger age:5 (29...35)	8	0.0873511	passenger age:6 (36...50)
9	0.0752975	passenger age:3 (15...21)	9	0.0759732	passenger age:5 (29...35)
10	0.0672325	passenger age:0 (missing)	10	0.0535455	passenger class:1st
11	0.0518915	passenger age:7 (50+)	11	0.0321496	passenger age:7 (50+)
12	0.0429124	passenger age:1 (under 6)	12	0.0198067	passenger age:2 (6...14)
13	0.0270378	passenger age:2 (6...14)	13	0.0140569	passenger age:1 (under 6)
14	0.	passenger survival:died	14	0.	passenger survival:survived
15	0.	passenger sex:male	15	0.	passenger sex:female

We can see that top “words” in the extracted two topics confirm that the passenger survival correlates with the passengers being first class and female. We also see that the passenger death correlates with the passengers being male and third class.

## Principal components analysis

PCA is a standard statistical procedure. Because of its utilization of an orthogonal linear space transformation it has limited use when working with categorical data, but here those related PCA properties can be useful to verify the correlations between the values of variables and the class labels.

By applying SVD to the matrix  $TM \in \mathbb{R}^{1309 \times 15}$  we are going to obtain an approximation of  $TM$  with the matrix product  $W S H$ ,  $W \in \mathbb{R}^{1309 \times 2}$ ,  $S \in \mathbb{R}^{2 \times 2}$ ,  $H \in \mathbb{R}^{2 \times 15}$ .

```
In[766]:= {W, S, H} = SingularValueDecomposition[N@TM, 2];
```

The following command gives the interpretation of the basis vectors:

```
In[767]:= Magnify[#, 0.8] & /@Map[GridTableForm[BasisVectorInterpretation[#, 15, words],
TableHeadings -> {"score", "tag"}] &, Transpose[H]]
```

#	score	tag	#	score	tag
1	-0.0261459	passenger age:1 (under 6)	1	0.294571	passenger survival:died
2	-0.0262607	passenger age:2 (6...14)	2	0.241316	passenger sex:male
3	-0.0440082	passenger age:7 (50+)	3	0.0340162	passenger class:3rd
4	-0.0991959	passenger age:5 (29...35)	4	0.00533461	passenger age:0 (missing)
5	-0.10484	passenger age:3 (15...21)	5	-0.0171344	passenger age:2 (6...14)
6	-0.118895	passenger age:6 (36...50)	6	-0.0349295	passenger age:1 (under 6)
7	-0.127011	passenger class:2nd	7	-0.0353647	passenger age:3 (15...21)
8	-0.13663	passenger class:1st	8	-0.0379867	passenger age:7 (50+)
9	-0.140121	passenger age:4 (22...28)	9	-0.0656465	passenger age:5 (29...35)
10	-0.169412	passenger age:0 (missing)	10	-0.066688	passenger age:4 (22...28)
11	-0.178492	passenger sex:female	11	-0.0908569	passenger age:6 (36...50)
12	-0.18876	passenger survival:survived	12	-0.108554	passenger class:2nd
13	-0.465237	passenger class:3rd	13	-0.268735	passenger class:1st
14	-0.540118	passenger survival:died	14	-0.584589	passenger sex:female
15	-0.550387	passenger sex:male	15	-0.637843	passenger survival:survived

We can see that because of the orthogonality of the new basis vectors they have coordinates with mixed signs. If the data matrix given to SVD is not normalized and centralized, the first vector, with the largest singular value, points to the center of the data. The second vector is oriented in such a way that the data is most diverse (spread out) in its direction. Now let us note that in the obtained second SVD vector (i) “died” and “male” have positive coordinates with close absolute values and (ii) “female” and

“survived” have negative coordinates with close absolute values. We can use this observation to confirm that “passenger sex” is the important variable in “Titanic”.

## Further investigations

### Repeating the results with several classifiers

It is good idea to verify that we get the same results using different classifiers. Below is given code that computes the shuffled accuracies and returns the relative damage scores for a set of methods of `Classify`.

```
In[782]:= mres = Association@Map[
  Function[{clMethod},
    cf = Classify[titanicTrainingData, Method -> clMethod];
    accRes = AccuracyByVariableShuffling[cf,
      titanicTestData, titanicVarNames, "FScoreLabels" -> "survived"];
    clMethod -> (accRes[None] - Rest[accRes]) / accRes[None]
  ], {"LogisticRegression", "NearestNeighbors",
    "NeuralNetwork", "RandomForest", "SupportVectorMachine"}];
```

```
In[783]:= Magnify[#, 0.8] &@Dataset[mres]
```

Out[783]=

	passengerclass	passengerage	passengersex
LogisticRegression	{0.1411}	{0.07227}	{0.4077}
NearestNeighbors	{0.1686}	{0.1071}	{0.4396}
NeuralNetwork	{0.2201}	{0.03633}	{0.3486}
RandomForest	{0.1887}	{0.01578}	{0.4026}
SupportVectorMachine	{0.196}	{-0.01575}	{0.449}

3 levels | 5 rows

### Pearson coefficient

Pearson coefficient can be used in the coloring of the mosaic plots. This functionality is not implemented in the package [6], but it is provided by R’s mosaic plots functions and packages. The interpretation is analogous to the impurity function values in the decision trees.

### Parallel implementation

A natural extension for the function `AccuracyByVariableShuffling` is to speed it up by parallel execution. Its parallel implementation is almost trivial using `ParallelMap` or `ParallelTable`.

### Dimension reduction before building the classifiers

We can apply the topic extraction described above first and build a classifier based on the matrix factor  $W$ . Using dimension reduction is one possible way to improve classifiers performance. Dimension reduction already allows direct variable importance interpretation by examining the norms of the vectors in the new vector bases. Using the classifier based data columns shuffling procedure is for further explanations and verification.

Note that in general with dimension reduction some experiments would be required to determine which variants of LSA, PCA, or NNMF to apply for best results and what is the best number of topics to which

to reduce the dimension. Also, because of the mapping of categorical data into a linear vector space, certain suitable numerical thresholds for the coordinates in the new reduced dimension basis have to be determined before building a classifier.

## DimensionReduction

One of the examples of the (experimental) built-in function `DimensionReduction` is making a recommender. We can turn such a recommender into a classifier and apply the main procedure described in this document.

## References

- [1] Leo Breiman et al., Classification and regression trees, Chapman & Hall, 1984, ISBN-13: 978-0412048418.
- [2] Anton Antonov, "Classification and association rules for census income data", (2014), MathematicaForPrediction at WordPress.com , URL: <https://mathematicaforprediction.wordpress.com/2014/03/30/classification-and-association-rules-for-census-income-data/> .
- [3] Anton Antonov, Decision tree and random forest implementations in Mathematica, (2013), source code MathematicaForPrediction at GitHub, <https://github.com/antononcube/MathematicaForPrediction>, package AVCDDecisionTreeForest.m.
- [4] Anton Antonov, Implementation of naive Bayesian classifier generation in Mathematica, (2013), source code at MathematicaForPrediction at GitHub, , <https://github.com/antononcube/MathematicaForPrediction>, package NaiveBayesianClassifier.m.
- [5] Anton Antonov, Mosaic plot for data visualization implementation in Mathematica, (2014), MathematicaForPrediction at GitHub, package MosaicPlot.m.
- [6] Anton Antonov, "Mosaic plots for data visualization", (2014), MathematicaForPrediction at Word-Press blog. URL: <https://mathematicaforprediction.wordpress.com/2014/03/17/mosaic-plots-for-data-visualization/> .
- [7] Anton Antonov, "MathematicaForPrediction utilities", (2014), source code MathematicaForPrediction at GitHub, <https://github.com/antononcube/MathematicaForPrediction>, package MathematicaForPredictionUtilities.m.
- [8] Anton Antonov, Variable importance determination by classifiers implementation in Mathematica, (2014), source code at MathematicaForPrediction at GitHub, <https://github.com/antononcube/MathematicaForPrediction>, package VariableImportanceByClassifiers.m.
- [9] Anton Antonov, "Waveform recognition with decision trees", (2013), MathematicaForPrediction at GitHub, <https://github.com/antononcube/MathematicaForPrediction>, folder Documentation.
- [10] Amiya Nayak (Editor), Ivan Stojmenovic (Editor), Handbook of Applied Algorithms: Solving Scientific, Engineering, and Practical Problems, Wiley-IEEE Press, 2008, ISBN: 978-0-470-04492-6.
- [11] Anton Antonov, "MovieLens genre associations" (2013), MathematicaForPrediction at GitHub, <https://github.com/antononcube/MathematicaForPrediction>, folder Documentation.
- [12] Anton Antonov, Implementation of the Apriori algorithm in Mathematica, (2014), source code at MathematicaForPrediction at GitHub, package AprioriAlgorithm.m.
- [13] Anton Antonov, "Statistical thesaurus from NPR podcasts", (2013), MathematicaForPrediction at WordPress blog. URL: <https://mathematicaforprediction.wordpress.com/2013/10/15/statistical-thesaurus-from-npr-podcasts/> .
- [14] Anton Antonov, "Topic and thesaurus extraction from a document collection" (2013), MathematicaForPrediction at GitHub, <https://github.com/antononcube/MathematicaForPrediction>, folder Documentation.
- [15] Anton Antonov, Implementation of the Non-Negative Matrix Factorization algorithm in Mathematica, (2013), source code at MathematicaForPrediction at GitHub, package NonNegativeMatrixFactorization.m.