Decision Trees

Decision trees are objects that help the understandement of the conditions leading to a particular outcome. In this section, several examples related to the construction of the decision trees are provided.

Ideas behind the building of decision trees are provided in scientific paper:

de Leoni, Massimiliano, Wil MP van der Aalst, and Marcus Dees. "A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs."

Expecially in Section 3 (extracting relevant process characteristics).

The general schema is the following:

- A representation of the log, on a given set of features, is obtained (for example, using one-hot encoding on string attributes and keeping numeric attributes as-they-are)
- A representation of the target classes is constructed

- The decision tree is calculated
- The decision tree is represented in some ways

Decision tree about the ending activity of a process

A process instance may potentially finish with different activities, signaling different outcomes of the process instance. A decision tree may help to understand the reasons behind each outcome.

First, a log could be loaded:

```
import os
from pm4py.objects.log.importer.xes import
factory as xes_importer

log = xes_importer.apply(os.path.join("tests",
"input_data", "roadtraffic50traces.xes"))
```

Then, a representation of a log on a given set of features could be obtained. Here:

- **str_trace_attributes** contains the attributes of type string, at trace level, that are one-hot encoded in the final matrix.
- **str_event_attributes** contains the attributes of type string, at event level, that are one-hot-encoded in the final matrix.
- **num_trace_attributes** contains the numeric

attributes, at trace level, that are inserted in the final matrix.

- **num_event_attributes** contains the numeric attributes, at event level, that are inserted in the final matrix.

```
from pm4py.objects.log.util import
get_log_representation

str_trace_attributes = []
str_event_attributes = ["concept:name"]
num_trace_attributes = []
num_event_attributes = ["amount"]

data, feature_names =
get_log_representation.get_representation(log,
str_trace_attributes, str_event_attributes,

num_trace_attributes, num_event_attributes)
```

Or an automatic representation (automatic selection of the attributes) could be obtained:

```
data, feature_names =
get_log_representation.get_default_representatio
```

Then, the target classes are formed. Each endpoint of the process belongs to a different class.

```
from pm4py.objects.log.util import
```

```
get_class_representation

target, classes =
get_class_representation.get_class_representatio
 "concept:name")
```

The decision tree could be then calculated:

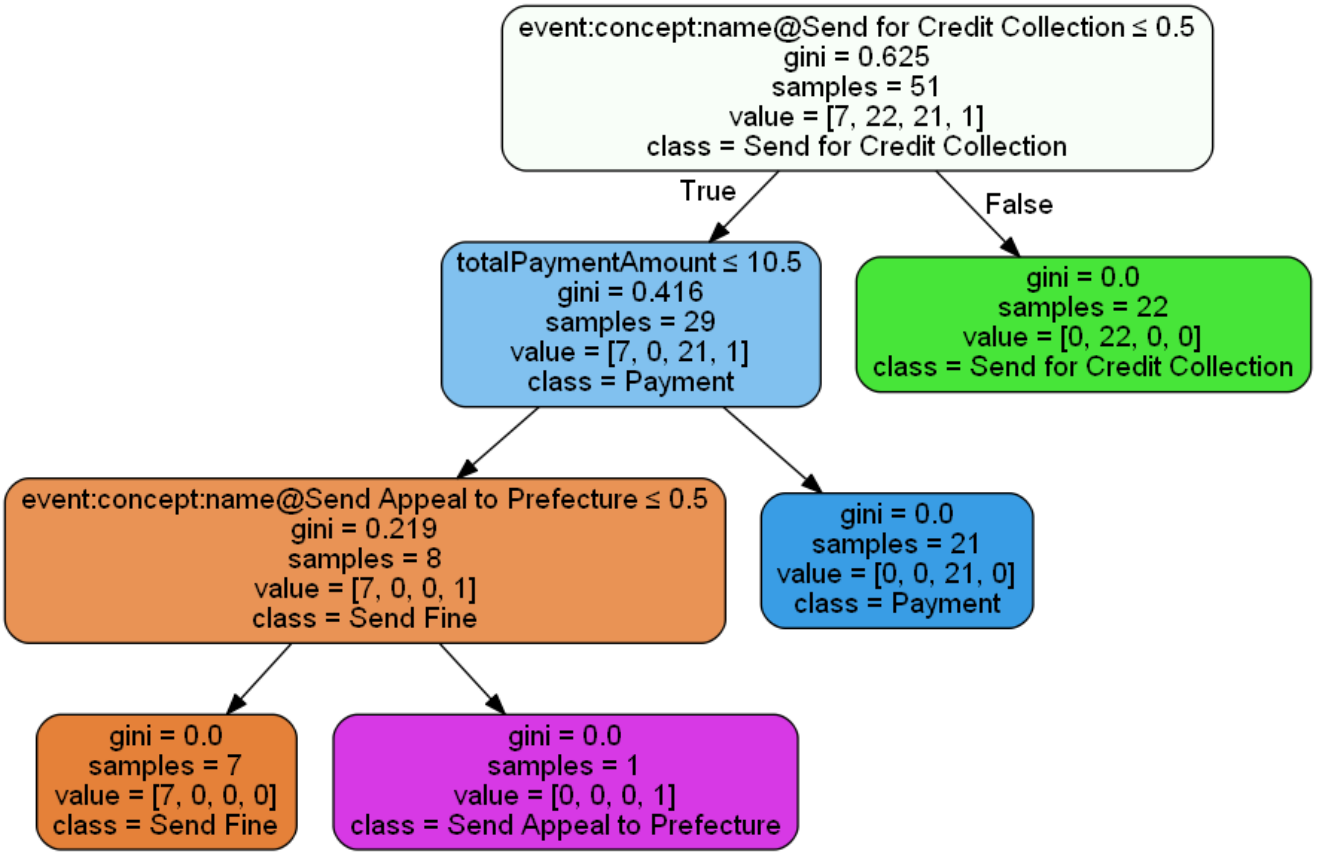
```
from sklearn import tree

clf = tree.DecisionTreeClassifier()
clf.fit(data, target)
```

and visualized:

```
from pm4py.visualization.decisiontree import
factory as dt_vis_factory

gviz = dt_vis_factory.apply(clf, feature_names,
classes)
```



Decision tree about the duration of a case (Root Cause Analysis)

A decision tree about the duration of a case helps to understand the reasons behind an high case duration (or, at least, a case duration that is above the threshold).

First, a log could be loaded:

```
import os
from pm4py.objects.log.importer.xes import
factory as xes_importer

log = xes_importer.apply(os.path.join("tests",
"input_data", "roadtraffic50traces.xes"))
```

Then, a representation of a log on a given set of features

could be obtained. Here:

- **str_trace_attributes** contains the attributes of type string, at trace level, that are one-hot encoded in the final matrix.
- **str_event_attributes** contains the attributes of type string, at event level, that are one-hot-encoded in the final matrix.
- **num_trace_attributes** contains the numeric attributes, at trace level, that are inserted in the final matrix.
- **num_event_attributes** contains the numeric attributes, at event level, that are inserted in the final matrix.

```
from pm4py.objects.log.util import
get_log_representation

str_trace_attributes = []
str_event_attributes = ["concept:name"]
num_trace_attributes = []
num_event_attributes = ["amount"]

data, feature_names =
get_log_representation.get_representation(log,
str_trace_attributes, str_event_attributes,

num_trace_attributes, num_event_attributes)
```

Or an automatic representation (automatic selection of the attributes) could be obtained:

```
data, feature_names =
get_log_representation.get_default_representatio
```

Then, the target classes are formed. There are two classes:

- Traces that are below the specified threshold (here, 200 days)
- Traces that are above the specified threshold

```
from pm4py.objects.log.util import
get_class_representation
```

```
target, classes =
get_class_representation.get_class_representatio
 2 * 8640000)
```

The decision tree could be then calculated:

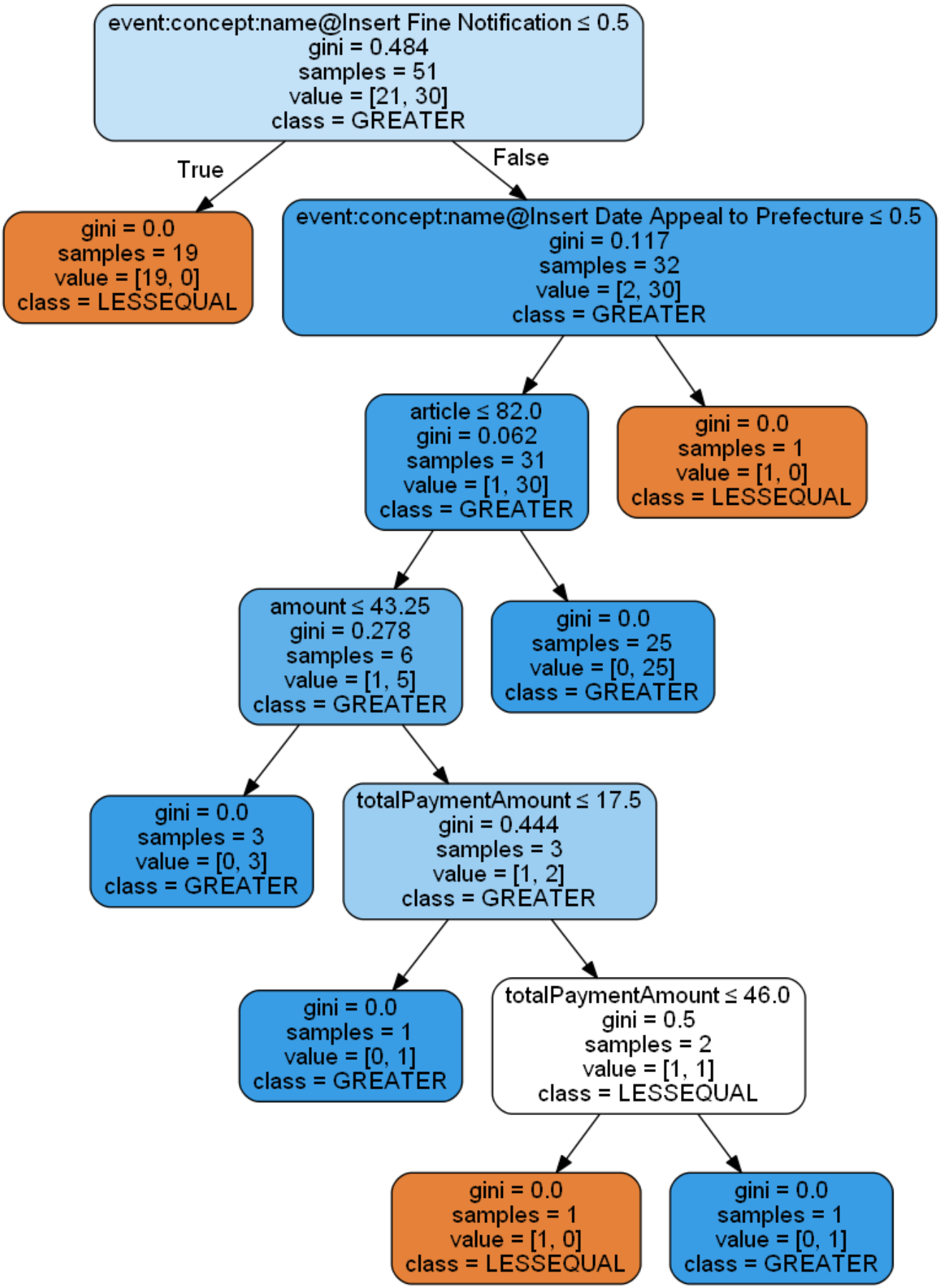
```
from sklearn import tree

clf = tree.DecisionTreeClassifier()
clf.fit(data, target)
```

and visualized:

```
from pm4py.visualization.decisiontree import
factory as dt_vis_factory
```

```
gviz = dt_vis_factory.apply(clf, feature_names, classes)
```



.