My Project

Generated by Doxygen 1.8.17

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

AuFileProcessor . . . . . ??
LinearClassifier . . . . . ??
log_struct . . . . . ??
LOGGER . . . . . ??
MachineLearningModel . . . . . ??
    ArtificialNeuralNetwork . . . . . ??
    DecisionTree . . . . . ??
    OneVsOneSVM . . . . . ??
    RandomForest . . . . . ??
Neuron . . . . . ??
TreeNode . . . . . ??

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

ArtificialNeuralNetwork . . . . . ??
AuFileProcessor . . . . . ??
DecisionTree . . . . . ??
LinearClassifier . . . . . ??
log_struct
    Hold structure for log config . . . . . ??
LOGGER . . . . . ??
MachineLearningModel . . . . . ??
Neuron . . . . . ??
OneVsOneSVM . . . . . ??
RandomForest . . . . . ??
TreeNode . . . . . ??

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

demo/artificial_neural_network_demo.cpp . . . ??
demo/decision_tree_demo.cpp . . . ??
demo/extractor_demo.cpp . . . ??
demo/one_vs_one_svm_demo.cpp . . . ??
demo/random_forest_demo.cpp . . . ??
extraction/au_file_processor.cpp . . . ??
extraction/au_file_processor.h . . . ??
helpers/classification_helpers.h . . . ??
helpers/file_helpers.h . . . ??
helpers/globals.h . . . ??
helpers/log.h . . . ??
helpers/music_style_helpers.h . . . ??
helpers/print_helpers.h . . . ??
helpers/signal.h . . . ??
ml_algorithms/artificial_neural_network.cpp . . . ??
ml_algorithms/artificial_neural_network.h . . . ??
ml_algorithms/decision_tree.cpp . . . ??
ml_algorithms/decision_tree.h . . . ??
ml_algorithms/machine_learning_model.h . . . ??
ml_algorithms/one_vs_one_svm.cpp . . . ??
ml_algorithms/one_vs_one_svm.h . . . ??
ml_algorithms/random_forest.cpp . . . ??
ml_algorithms/random_forest.h . . . ??

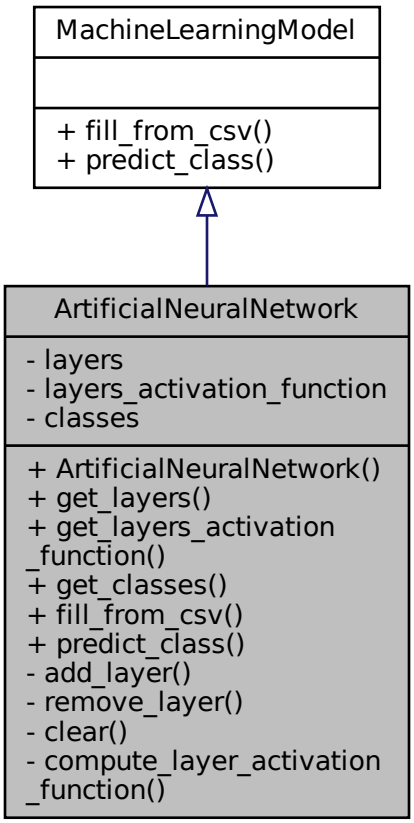
Chapter 4

Class Documentation

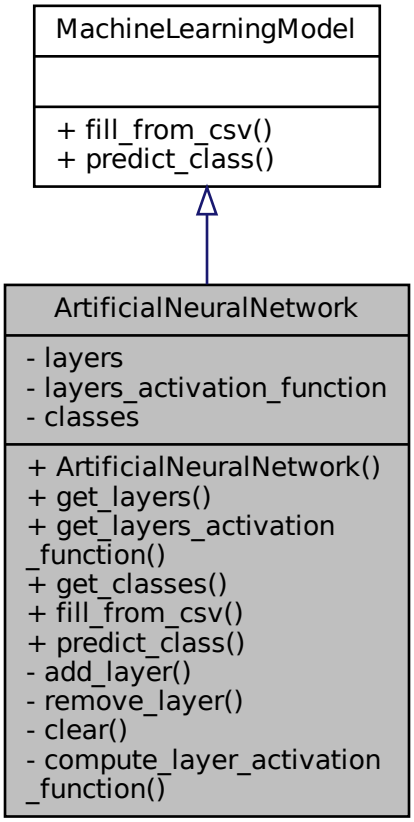
4.1 ArtificialNeuralNetwork Class Reference

`#include <artificial_neural_network.h>`

Inheritance diagram for ArtificialNeuralNetwork:



Collaboration diagram for ArtificialNeuralNetwork:



Public Member Functions

- ArtificialNeuralNetwork ()
- const std::map< std::size_t, std::vector< Neuron > > & get_layers () const
- const std::map< std::size_t, ActivationFunction > & get_layers_activation_function () const
- const std::vector< std::string > & get_classes () const
- void fill_from_csv (const std::filesystem::path &csv_file_path) override
- std::string predict_class (const real_vector_t &features_vector) override

Private Member Functions

- void add_layer (std::size_t layer_id, const std::vector< Neuron > &neurons, ActivationFunction activation_↩ function)
- void remove_layer (std::size_t layer_id)
- void clear ()
- real_t compute_layer_activation_function (std::size_t layer_ind, real_vector_t &weighted_sums, std::size_↩ t weighted_sum_ind)

Private Attributes

- std::map< std::size_t, std::vector< Neuron > > layers
- std::map< std::size_t, ActivationFunction > layers_activation_function
- std::vector< std::string > classes

Friends

- std::ostream & operator<< (std::ostream &os, const ArtificialNeuralNetwork &artificial_neural_network)

4.1.1 Detailed Description

Definition at line 40 of file artificial_neural_network.h.

4.1.2 Constructor & Destructor Documentation

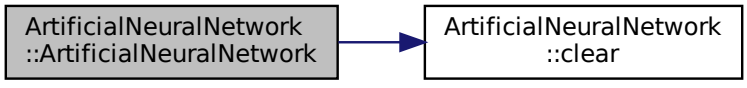
4.1.2.1 ArtificialNeuralNetwork()

```
ArtificialNeuralNetwork::ArtificialNeuralNetwork ( )
```

Definition at line 52 of file artificial_neural_network.cpp.

```
52                                                {
53     this->clear();
54 }
```

Here is the call graph for this function:



4.1.3 Member Function Documentation

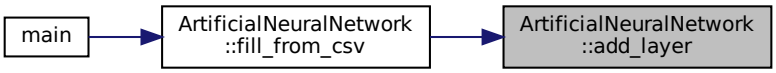
4.1.3.1 add_layer()

```
void ArtificialNeuralNetwork::add_layer (
            std::size_t layer_id,
            const std::vector< Neuron > & neurons,
            ActivationFunction activation_function )  [private]
```

Definition at line 201 of file artificial_neural_network.cpp.

```
201
                                {
202     this->layers.insert(std::make_pair(layer_id, neurons));
203     this->layers_activation_function.insert(std::make_pair(layer_id, activation_function));
204 }
```

Here is the caller graph for this function:



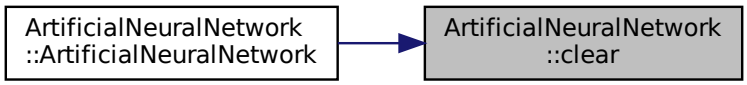
4.1.3.2 clear()

```
void ArtificialNeuralNetwork::clear ( )  [private]
```

Definition at line 211 of file artificial_neural_network.cpp.

```
211                                 {
212     this->layers.clear();
213     this->layers_activation_function.clear();
214 }
```

Here is the caller graph for this function:



4.1.3.3 compute_layer_activation_function()

```
real_t ArtificialNeuralNetwork::compute_layer_activation_function (
            std::size_t layer_ind,
            real_vector_t & weighted_sums,
            std::size_t weighted_sum_ind )  [private]
```

Definition at line 216 of file artificial_neural_network.cpp.

```
216                                               {
217     switch (this->layers_activation_function.at(layer_ind)) {
218         case ActivationFunction::SIGMOID : {
219             return 1.0 / (1.0 + std::exp(-weighted_sums.at(weighted_sum_ind)));
220             break;
221         }
222         case ActivationFunction::RELU : {
223             return std::max((real_t) 0, weighted_sums.at(weighted_sum_ind));
224             break;
225         }
226         case ActivationFunction::SOFTMAX : {
227             return std::exp(weighted_sums.at(weighted_sum_ind)) /
       std::transform_reduce(std::execution::seq, weighted_sums.cbegin(), weighted_sums.cend(), 0.0,
       std::plus<>(), [](real_t r)mutable {
228                 return std::exp(r);
229             });
230             break;
231         }
232         default: {
233             LOG(LOG_ERROR) « "Error : the activation function value usage is not defined in the
       project";
234             throw std::domain_error("Unsupported activation function!");
235             break;
236         }
237     }
238     return 0;
239 }
```

4.1.3.4 fill_from_csv()

```
void ArtificialNeuralNetwork::fill_from_csv (
            const std::filesystem::path & csv_file_path )  [override], [virtual]
```

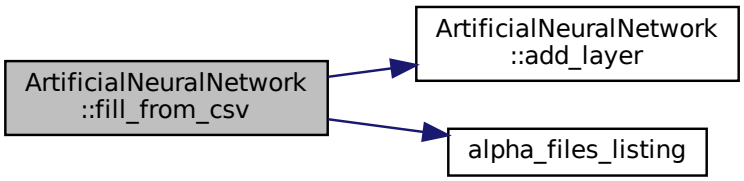
Implements MachineLearningModel.

Definition at line 98 of file artificial_neural_network.cpp.

```
98                                                                             {
99     const char delimiter = ',';
100     const std::string csv_extension = ".csv";
101
102
103     auto csv_files = alpha_files_listing(csv_folder_path.string(), csv_extension);
104     for (std::size_t layer_i = 0; layer_i < csv_files.size(); layer_i++) {
105         LOG(LOG_DEBUG) « "reading: " « layer_i « ", " « csv_files.at(layer_i);
106         std::string line = {};
107         std::string data = {};
108
109         std::ifstream input_file(csv_files.at(layer_i));
110         if (!input_file.is_open()) {
111             throw std::filesystem::filesystem_error("Can't open file!",
       std::make_error_code(std::errc::no_such_file_or_directory));
112         }
113
114         //TODO: check file extension and header
115         std::vector<Neuron> layer = {};
116         std::vector<std::string> pred_classes = {};
117         ActivationFunction layer_activation_function;
118         bool header_skipped = false;
119         while (std::getline(input_file, line)) {
120             if (!header_skipped) {
121                 header_skipped = true;
122                 continue;
123             } else {
124                 std::stringstream ss(line);
```

```
125                 size_t last = 0;
126                 size_t next = 0;
127                 // Get bias from line
128                 next = line.find(delimiter, last);
129                 real_t bias = std::stod(line.substr(last, next - last));
130                 last = next + 1;
131                 // Get the weigths from line
132                 real_vector_t weigths;
133                 while ((next = line.find(delimiter, last)) != std::string::npos) {
134                     weigths.push_back((real_t) std::stod(line.substr(last, next - last)));
135                     last = next + 1;
136                 }
137                 // If this is the output layer, the last value is the class names
138                 if (layer_i + 1 == csv_files.size()) {
139                     std::string class_name = line.substr(last);
140                     // Remove double quote characters around the class name
141                     class_name.erase(remove(class_name.begin(), class_name.end(), '"'),
    class_name.end());
142                     pred_classes.push_back(class_name);
143                 } else {
144                     weigths.push_back((real_t) std::stod(line.substr(last)));
145                 }
146 
147                 Neuron neuron = {bias, weigths};
148                 layer.push_back(neuron);
149             }
150         }
151         // Use softmax only for last layer, else use Relu
152         if (layer_i + 1 == csv_files.size()) {
153             layer_activation_function = ActivationFunction::SOFTMAX;
154         } else {
155             layer_activation_function = ActivationFunction::RELU;
156         }
157         this->add_layer(layer_i, layer, layer_activation_function);
158         this->classes = pred_classes;
159     }
160 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

4.1.3.5 get_classes()

```
const std::vector< std::string > & ArtificialNeuralNetwork::get_classes ( ) const
```

Definition at line 65 of file artificial_neural_network.cpp.

```
65                                                                  {
66     return classes;
67 }
```

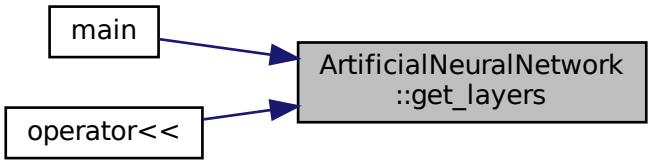
4.1.3.6 get_layers()

```
const std::map< std::size_t, std::vector< Neuron > > & ArtificialNeuralNetwork::get_layers (
) const
```

Definition at line 56 of file artificial_neural_network.cpp.

```
56                                                                                  {
57     return layers;
58 }
```

Here is the caller graph for this function:



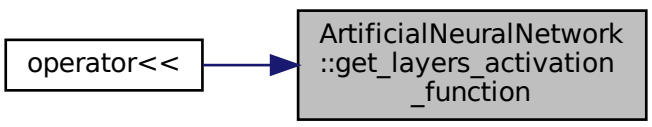
4.1.3.7 get_layers_activation_function()

```
const std::map< std::size_t, ActivationFunction > & ArtificialNeuralNetwork::get_layers_↩
activation_function ( ) const
```

Definition at line 60 of file artificial_neural_network.cpp.

```
60
       {
61     return layers_activation_function;
62 }
```

Here is the caller graph for this function:



4.1.3.8 predict_class()

```
std::string ArtificialNeuralNetwork::predict_class (
            const real_vector_t & features_vector )  [override], [virtual]
```

Implements MachineLearningModel.

Definition at line 162 of file artificial_neural_network.cpp.

```
162                                                                                 {
163     real_vector_t last_activations = features_vector;
164     real_vector_t next_activations = features_vector;
165     std::for_each(std::execution::seq, this->layers.cbegin(), this->layers.cend(), [&last_activations,
      &next_activations, this](const std::pair<std::size_t, std::vector<Neuron» &pair)mutable {
166         last_activations.clear();
167         std::move(next_activations.cbegin(), next_activations.cend(),
      std::back_inserter(last_activations));
168         next_activations.clear();
169         real_vector_t weighted_sum = {};
170         for (Neuron n: pair.second) {
171             weighted_sum.push_back(n.compute_weighted_sum(last_activations));
172         }
173         for (std::size_t neuron_i = 0; neuron_i < weighted_sum.size(); neuron_i++) {
174             switch (this->layers_activation_function.at(pair.first)) {
175                 case ActivationFunction::SIGMOID :
176                 case ActivationFunction::RELU : {
177                     next_activations.push_back(this->compute_layer_activation_function(pair.first,
      weighted_sum, neuron_i));
178                     break;
179                 }
180                 case ActivationFunction::SOFTMAX : {
181                     next_activations.push_back(this->compute_layer_activation_function(pair.first,
      weighted_sum, neuron_i));
182                     break;
183                 }
184                 default: {
185                     LOG(LOG_ERROR) « "Error : the activation function value usage is not defined in the
      project";
186                     throw std::domain_error("Unsupported activation function!");
187                     break;
188                 }
189             }
190 
191         }
192     });
193 
194     // Get the prediction result with the most choice
195     auto pr = std::max_element(std::execution::seq, next_activations.begin(), next_activations.end());
196 
197     return this->classes.at(std::distance(next_activations.begin(), pr));
198 }
```

4.1.3.9 remove_layer()

```
void ArtificialNeuralNetwork::remove_layer (
            std::size_t layer_id )  [private]
```

Definition at line 206 of file artificial_neural_network.cpp.

```
206                                                                 {
207     this->layers.erase(layer_id);
208     this->layers_activation_function.erase(layer_id);
209 }
```

4.1.4 Friends And Related Function Documentation

4.1.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const ArtificialNeuralNetwork & artificial_neural_network )  [friend]
```

Definition at line 69 of file artificial_neural_network.cpp.

```
69                                                                                                  {
70     for (const std::pair<std::size_t, std::vector<Neuron» layer: artificial_neural_network.get_layers())
       {
71         os « "layer " « layer.first « " (" « layer.second.size() « ")";
72         switch (artificial_neural_network.get_layers_activation_function().at(layer.first)) {
73             case ActivationFunction::SIGMOID : {
74                 os « " activation function SIGMOID";
75                 break;
76             }
77             case ActivationFunction::RELU : {
78                 os « " activation function RELU";
79                 break;
80             }
81             case ActivationFunction::SOFTMAX : {
82                 os « " activation function SOFTMAX";
83                 break;
84             }
85             default: {
86                 os « " activation function UNKNOWN";
87                 break;
88             }
89         }
90         os « ": \n";
91         for (const Neuron &neuron: layer.second) {
92             os « "\t - " « neuron « "\n";
93         }
94     }
95     return os;
96 }
```

4.1.5 Member Data Documentation

4.1.5.1 classes

```
std::vector<std::string> ArtificialNeuralNetwork::classes  [private]
```

Definition at line 59 of file artificial_neural_network.h.

4.1.5.2 layers

```
std::map<std::size_t, std::vector<Neuron> > ArtificialNeuralNetwork::layers  [private]
```

Definition at line 57 of file artificial_neural_network.h.

4.1.5.3 layers_activation_function

```
std::map<std::size_t, ActivationFunction> ArtificialNeuralNetwork::layers_activation_function
[private]
```

Definition at line 58 of file artificial_neural_network.h.

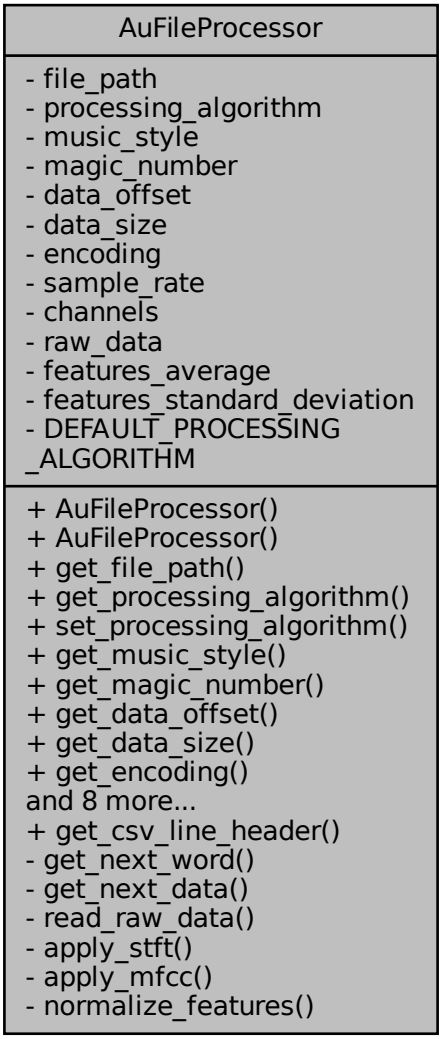
The documentation for this class was generated from the following files:

- ml_algorithms/artificial_neural_network.h
- ml_algorithms/artificial_neural_network.cpp

4.2 AuFileProcessor Class Reference

```
#include <au_file_processor.h>
```

Collaboration diagram for AuFileProcessor:



Public Member Functions

- AuFileProcessor (const std::filesystem::path &file_path)

  AuFileProcessor constructor.
- AuFileProcessor (const std::filesystem::path &file_path, AuFileProcessingAlgorithm processing_algorithm)

  AuFileProcessor constructor.
- const std::filesystem::path & get_file_path () const

  Get the .au file path.
- AuFileProcessingAlgorithm get_processing_algorithm () const

  Get the .au file current process algorithm.
- void set_processing_algorithm (AuFileProcessingAlgorithm algorithm)

  Set the .au current process algorithm. param[in] processing_algorithm a AuFileProcessingAlgorithm enum variable that represent the process algorithm to use.
- const std::string & get_music_style () const

  Get the .au file music style.
- word_t get_magic_number () const

  Get the .au file magic number.
- word_t get_data_offset () const

  Get the .au file data offset.
- word_t get_data_size () const

  Get the .au file data size.
- word_t get_encoding () const

  Get the .au file encoding format.
- word_t get_sample_rate () const

  Get the .au file sample rate (in sample/sec).
- word_t get_channels () const

  Get the .au file number of interleaved channels.
- const real_vector_t & get_raw_data () const

  Get the .au file raw data vector.
- const real_vector_t & get_features_average () const

  Get the .au file features average.
- const real_vector_t & get_features_standard_deviation () const

  Get the .au file features standard deviation.
- void read_file ()

  Parse the .au file header and data and save them to their corresponding class variables.
- void apply_processing_algorithm ()

  Apply the process algorithm corresponding to the value of the processing_algorithm class variable.
- std::string get_csv_line ()

  Parse the file style, path and computed features values in a ready to use csv line.

Static Public Member Functions

- static std::string get_csv_line_header (AuFileProcessingAlgorithm processing_algorithm=AuFileProcessor::DEFAULT_PROCE

  Return a ready to use csv header for the .au file features.

Private Member Functions

- word_t get_next_word (std::ifstream &file)

  Read next word from the file ifstream.
- int16_t get_next_data (std::ifstream &file)

  Read next data from the file ifstream.
- void read_raw_data (std::ifstream &file)

  Read all music data from the file ifstream.
- void apply_stft ()

  Apply the stft to the raw data and save the average and standard deviation to their corresponding class variables.
- void apply_mfcc ()

  Apply the mfcc to the raw data and save the average and standard deviation to their corresponding class variables.
- void normalize_features ()

  Normalize the features_average and features_standard_deviation vectors using the formula: normalized_vector = (vector - mean) / stdev.

Private Attributes

- std::filesystem::path file_path
- AuFileProcessingAlgorithm processing_algorithm
- std::string music_style
- word_t magic_number
- word_t data_offset
- word_t data_size
- word_t encoding
- word_t sample_rate
- word_t channels
- real_vector_t raw_data
- real_vector_t features_average {}
- real_vector_t features_standard_deviation {}

Static Private Attributes

- static const AuFileProcessingAlgorithm DEFAULT_PROCESSING_ALGORITHM = AuFileProcessingAlgorithm::STFT

Friends

- std::ostream & operator<< (std::ostream &os, const AuFileProcessor &au_file_processor)

  Overload the << operator for the AuFileProcessor object.

4.2.1 Detailed Description

Definition at line 44 of file au_file_processor.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 AuFileProcessor() [1/2]

```
AuFileProcessor::AuFileProcessor (
            const std::filesystem::path & file_path )  [explicit]
```

AuFileProcessor constructor.

Parameters

in	*file_path*	a std::filesystem::path object of the .au file to process @trhow <std::domain_error("Not a .au file!")> Throw an exception if the file extension is not .au

Returns

the constructed object

Definition at line 9 of file au_file_processor.cpp.

```
9                                                                   : file_path(file_path) {
10      // Testing file extension
11      if (file_path.filename().extension() != ".au") {
12          LOG(LOG_ERROR) « "Error : file " « file_path.filename() « " does not have the .au extension";
13          throw std::domain_error("Not a .au file!");
14      }
15
16      // Set the processing algorithm to the default value
17      this->processing_algorithm = AuFileProcessor::DEFAULT_PROCESSING_ALGORITHM;
18
19      // Extracting music style from file name
20      std::string string_file_path = file_path.filename().string();
21      this->music_style = string_file_path.substr(0, string_file_path.find('.'));
22
23      this->magic_number = 0;
24      this->data_offset = 0;
25      this->data_size = 0;
26      this->encoding = 0;
27      this->sample_rate = 0;
28      this->channels = 0;
29      this->features_average = {};
30      this->features_standard_deviation = {};
31
32  }
```

4.2.2.2 AuFileProcessor() [2/2]

```
AuFileProcessor::AuFileProcessor (
            const std::filesystem::path & file_path,
            AuFileProcessingAlgorithm processing_algorithm )  [explicit]
```

AuFileProcessor constructor.

Parameters

in	*file_path*	a std::filesystem::path object of the .au file to process
in	*processing_algorithm*	a AuFileProcessingAlgorithm enum variable that represent the processing algorithm to use @trhow <std::domain_error("Not a .au file!")> Throw an exception if the file extension is not .au

Returns

the constructed object

Definition at line 34 of file au_file_processor.cpp.

```
34
                : file_path(file_path), processing_algorithm(processing_algorithm) {
35      // Testing file extension
36      if (file_path.filename().extension() != ".au") {
```

```
37          LOG(LOG_ERROR) « "Error : file " « file_path.filename() « " does not have the .au extension";
38          throw std::domain_error("Not a .au file!");
39      }
40
41      // Extracting music style from file name
42      std::string string_file_path = file_path.filename().string();
43      this->music_style = string_file_path.substr(0, string_file_path.find('.'));
44
45      this->magic_number = 0;
46      this->data_offset = 0;
47      this->data_size = 0;
48      this->encoding = 0;
49      this->sample_rate = 0;
50      this->channels = 0;
51      this->features_average = {};
52      this->features_standard_deviation = {};
53  }
```

4.2.3 Member Function Documentation

4.2.3.1 apply_mfcc()

```
void AuFileProcessor::apply_mfcc ( )  [private]
```

Apply the mfcc to the raw data and save the average and standard deviation to their corresponding class variables.

The mfcc, like the stft, extract from the raw data 2 list of data block, one with a step size of N and one with a step size of N/2. After the computation of a list of filter depending on the min and max frequency and the number of filters wanted, a hamming window is applied using the windowing helper function. The log value of this data block signal energy is then computed. Then the Iterative Direct Transform Fourrier is applied, the results is converted from complex to real and only half of v1 and v2 is kept because the fft result is symmetrical regarding the origin. The filters are then applied using the apply_filterbank helper function followed by the the Discret Cosinus Transform using the dct2 helper function Finally, the average and standard deviation of all values are computed.

Returns

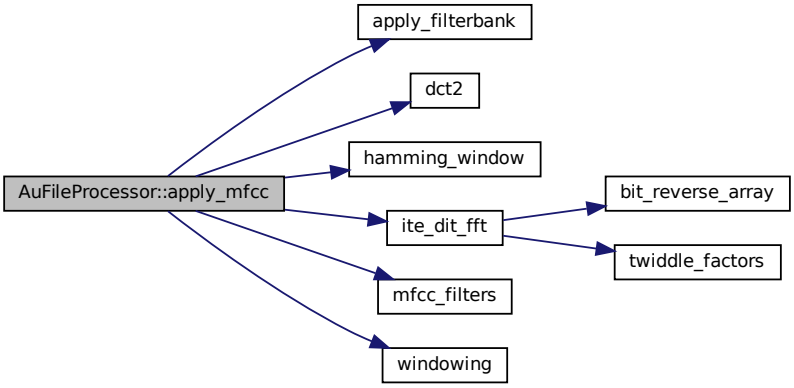
void

Definition at line 334 of file au_file_processor.cpp.

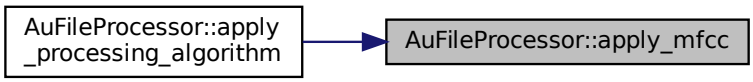
```
334                                 {
335      // Create bank of filter
336      std::array<real_fft_array_t, MEL_N> filter_bank_t = mfcc_filters();
337      std::array<real_fft_array_t, MEL_APPLIED_N> filter_bank = {};
338      std::move(filter_bank_t.cbegin(), filter_bank_t.cbegin() + filter_bank.size(), filter_bank.begin());
339
340      // Apply hamming window
341      real_n_array_t h_window = hamming_window();
342      real_vector_t energy;
343
344      real_t size = 0;
345      // Use an array of size MEL_APPLIED_N+1 because we have the filters and the energy
346      std::array<real_t, MEL_APPLIED_N + 1> mean_prev = {};
347      std::array<real_t, MEL_APPLIED_N + 1> mean = {};
348      std::array<real_t, MEL_APPLIED_N + 1> stdv = {};
349      this->features_average = {};
350      this->features_standard_deviation = {};
351
352      for (std::size_t k = 0; k < raw_data.size() / N; k++) {
353          complex_n_array_t v1;
354          complex_n_array_t v2;
355          real_t energy_v1;
356          real_t energy_v2;
357          real_fft_array_t e1;
358          real_fft_array_t e2;
359
360          std::array<real_t, MEL_APPLIED_N + 1> e1_result{};
361          std::array<real_t, MEL_APPLIED_N + 1> e2_result{};
```

```
362
363         // insert raw data into v1 using a step of size N
364         std::copy(raw_data.cbegin() + k * N, raw_data.cbegin() + k * N + N, v1.begin());
365         // insert raw data into v2 using a step of size N/2
366         std::copy(raw_data.cbegin() + k * N + N / 2, raw_data.cbegin() + k * N + N + N / 2, v2.begin());
367
368         // apply windowing on v1 and v2
369         windowing(h_window, v1);
370         windowing(h_window, v2);
371
372         // Compute frame energy (divide by 1e3 to be in the same order of magnitude as the means and
    standard deviation)
373         energy_v1 = std::transform_reduce(v1.cbegin(), v1.cend(), 0.0, std::plus<>(), [](complex_t
    c)mutable {
374             return std::log(std::max(std::abs(c * c), 2e-22)) / 1e3;
375         });
376         energy_v2 = std::transform_reduce(v2.cbegin(), v2.cend(), 0.0, std::plus<>(), [](complex_t
    c)mutable {
377             return std::log(std::max(std::abs(c * c), 2e-22)) / 1e3;
378
379         });
380
381         // compute the fft of v1 and v2
382         ite_dit_fft(v1);
383         ite_dit_fft(v2);
384
385         // We need only half of v1 and v2 because the fft result is symmetrical regarding the origin
386         // note that function send directly the magnitude of each frequency
387         // convert complex data to simple
388         std::transform(std::execution::seq, v1.cbegin(), v1.cbegin() + (FFT_SIZE - 1), e1.begin(),
    [](complex_t c) { return std::abs(sqrt(c.real() * c.real() + c.imag() * c.imag())); });
389         std::transform(std::execution::seq, v2.cbegin(), v2.cbegin() + (FFT_SIZE - 1), e2.begin(),
    [](complex_t c) { return std::abs(sqrt(c.real() * c.real() + c.imag() * c.imag())); });
390
391         e1_result.at(0) = energy_v1;
392         e2_result.at(0) = energy_v2;
393
394         // Apply the dtc and filterbank on e1 and e2 (output vector size is MEL_APPLIED_N)
395         // The function apply_filterbank take the log of each value -> non linear rectification
396         real_vector_t e1_filt = dct2(apply_filterbank(filter_bank, e1));
397         real_vector_t e2_filt = dct2(apply_filterbank(filter_bank, e2));
398
399         std::move(e1_filt.begin(), e1_filt.end(), e1_result.begin() + 1);
400         std::move(e2_filt.begin(), e2_filt.end(), e2_result.begin() + 1);
401
402         size++;
403         for (std::size_t iter = 0; iter < e1_result.size(); iter++) {
404             mean_prev.at(iter) = mean.at(iter);
405             mean.at(iter) += (e1_result.at(iter) - mean.at(iter)) / size;
406             stdv.at(iter) += (e1_result.at(iter) - mean.at(iter)) * (e1_result.at(iter) -
    mean_prev.at(iter));
407             mean_prev.at(iter) = mean.at(iter);
408             mean.at(iter) += (e2_result.at(iter) - mean.at(iter)) / size;
409             stdv.at(iter) += (e2_result.at(iter) - mean.at(iter)) * (e2_result.at(iter) -
    mean_prev.at(iter));
410         }
411     }
412
413     this->features_average.clear();
414     std::move(mean.cbegin(), mean.cend(), std::back_inserter(this->features_average));
415     this->features_standard_deviation.clear();
416     std::transform(stdv.begin(), stdv.end(), std::back_inserter(this->features_standard_deviation),
    [size](real_t c) { return std::sqrt(c / (size - 1)); });
417 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.3.2 apply_processing_algorithm()

```
void AuFileProcessor::apply_processing_algorithm ( )
```

Apply the process algorithm corresponding to the value of the processing_algorithm class variable.

To change the algorithm to use use the setter class method set_processing_algorithm.

Returns

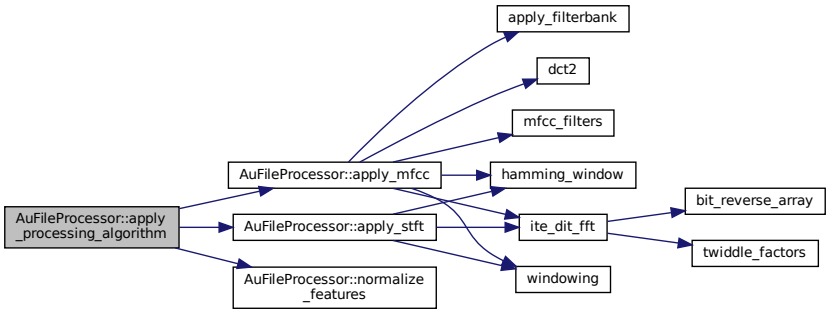
void

Definition at line 219 of file au_file_processor.cpp.

```
219                                                 {
220     // Apply process algorithm
221     switch (this->processing_algorithm) {
222         case AuFileProcessingAlgorithm::STFT: {
223             apply_stft();
224             break;
225         }
226         case AuFileProcessingAlgorithm::MFCC: {
227             apply_mfcc();
228             break;
```

```
229         }
230         default: {
231             LOG(LOG_ERROR) << "Error : the processing algorithm value usage is not defined in the
    project";
232             throw std::domain_error("Unsupported processing algorithm!");
233             break;
234         }
235     }
236 
237     // Normalize features
238     this->normalize_features();
239 }
```

Here is the call graph for this function:



4.2.3.3 apply_stft()

```
void AuFileProcessor::apply_stft ( )  [private]
```

Apply the stft to the raw data and save the average and standard deviation to their corresponding class variables.

The stft extract from the raw data 2 list of data block, one with a step size of N and one with a step size of N/2. Then a hamming window is applied using the windowing helper function and the representation in the frequency domain is computed using the ite_dit_fft (Iterative Direct Transform Fourrier) helper function. Finally, the real value of only half of v1 and v2 is kept because the fft result is symmetrical regarding the origin and the average and standard deviation of each frequency are computed.

Returns

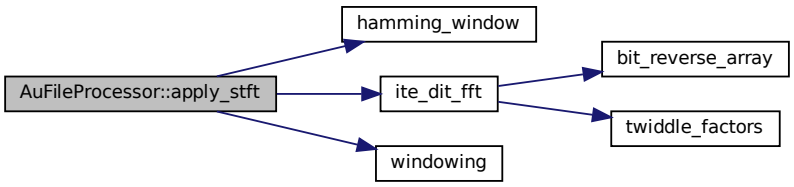
void

Definition at line 285 of file au_file_processor.cpp.

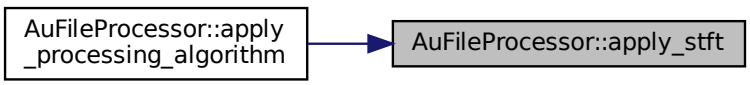
```
285                                 {
286     // Create hamming window
287     real_n_array_t h_window = hamming_window();
288 
289     real_t size = 0;
290     real_t x = 0;
291     real_fft_array_t mean_prev = {};
292     real_fft_array_t mean = {};
293     real_fft_array_t stdv = {};
294     this->features_average = {};
295     this->features_standard_deviation = {};
296 
297     for (std::size_t k = 0; k < raw_data.size() / N; k++) {
298         complex_n_array_t v1;
299         complex_n_array_t v2;
```

```
300
301         // insert raw data into v1 using a step of size N
302         std::vector<complex_t> V1;
303         std::copy(raw_data.cbegin() + k * N, raw_data.cbegin() + k * N + N, v1.begin());
304         // insert raw data into v2 using a step of size N/2
305         std::copy(raw_data.cbegin() + k * N + N / 2, raw_data.cbegin() + k * N + N + N / 2, v2.begin());
306
307         // apply windowing and fft to v1 and v2
308         windowing(h_window, v1);
309         windowing(h_window, v2);
310         // compute the fft of v1 and v2
311         ite_dit_fft(v1);
312         ite_dit_fft(v2);
313
314         // We need only half of v1 and v2 because the fft result is symmetrical regarding the origin
315         size++;
316         for (std::size_t iter = 0; iter < FFT_SIZE; iter++) {
317             x = std::abs(v1.at(iter));
318             mean_prev.at(iter) = mean.at(iter);
319             mean.at(iter) += (x - mean.at(iter)) / size;
320             stdv.at(iter) += (x - mean.at(iter)) * (x - mean_prev.at(iter));
321             mean_prev.at(iter) = mean.at(iter);
322             x = std::abs(v2.at(iter));
323             mean.at(iter) += (x - mean.at(iter)) / size;
324             stdv.at(iter) += (x - mean.at(iter)) * (x - mean_prev.at(iter));
325         }
326
327     }
328     this->features_average.clear();
329     std::move(mean.cbegin(), mean.cend(), std::back_inserter(this->features_average));
330     this->features_standard_deviation.clear();
331     std::transform(stdv.begin(), stdv.end(), std::back_inserter(this->features_standard_deviation),
       [size](real_t c) { return std::sqrt(c / (size - 1)); });
332 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.3.4 get_channels()

```
word_t AuFileProcessor::get_channels ( ) const
```

Get the .au file number of interleaved channels.

Returns

the channels class variable value.

Definition at line 91 of file au_file_processor.cpp.

```
91                                          {
92     return channels;
93 }
```

4.2.3.5 get_csv_line()

```
std::string AuFileProcessor::get_csv_line ( )
```

Parse the file style, path and computed features values in a ready to use csv line.

The format of the csv file correspond to the one described in get_csv_line_header and the computed features values depend on the processing algorithm used.

Returns

the created line

Definition at line 168 of file au_file_processor.cpp.

```
168                                         {
169     std::string csv_line;
170     std::string features_average_csv_string = {};
171     for (const real_t &r: this->features_average) {
172         features_average_csv_string += (std::to_string(r) + ",");
173     }
174     std::string features_standard_deviation_csv_string = {};
175     for (const real_t &r: this->features_standard_deviation) {
176         features_standard_deviation_csv_string += (std::to_string(r) + ",");
177     }
178     csv_line = std::string(features_average_csv_string + features_standard_deviation_csv_string + "\"" +
       this->music_style + "\",\"" + this->file_path.string() + "\"");
179
180     return csv_line;
181 }
```

4.2.3.6 get_csv_line_header()

```
std::string AuFileProcessor::get_csv_line_header (
            AuFileProcessingAlgorithm processing_algorithm = AuFileProcessor::DEFAULT_PROCESSING_ALGORITHM
)  [static]
```

Return a ready to use csv header for the .au file features.

The format of the csv file depend on the processing algorithm used; When using the stft algorithm, the header format is the following: <BIN_AVG0,...,BIN_AVG255,BIN_STDEV0,...,BIN_STDEV254,style,fileName>. BIN_↩ AVG and BIN_STDEV are casted to string from real_t type with BIN_AVG corresponding to the average values and BIN_STDEV to the standard deviation values. When using the mfcc algorithm, the header format is the following: <SIGNALENERGY_AVG,BIN_AVG0,...,BIN_AVG18,SIGNALENERGY_STDEV,BIN_STDEV0,...,BIN↩ _STDEV18,style,fileName>. SIGNALENERGY_AVG, BIN_AVG, SIGNALENERGY_STDEV and BIN_STDEV are casted to string from real_t type with BIN_AVG corresponding to the average values, BIN_STDEV to the standard deviation values and SIGNALENERGY_AVG and SIGNALENERGY_STDEV to the avg or stdev signal energy. Style and FileName are string with the double quote.

Parameters

in	*processing_algorithm*	a AuFileProcessingAlgorithm enum variable that represent the process algorithm to use (default value: AuFileProcessor::DEFAULT_PROCESSING_ALGORITHM)

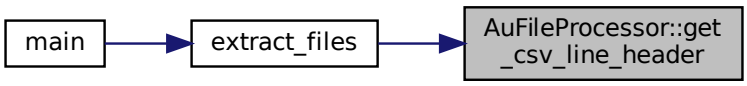
Returns

the created csv header

Definition at line 183 of file au_file_processor.cpp.

```
183                                                                                                                {
184     std::string header = {};
185     switch (processing_algorithm) {
186         case AuFileProcessingAlgorithm::STFT: {
187             for (std::size_t i = 0; i < FFT_SIZE; i++) {
188                 header += ("BIN_AVG" + std::to_string(i) + ",");
189             }
190             for (std::size_t i = 0; i < FFT_SIZE; i++) {
191                 header += ("BIN_STDEV" + std::to_string(i) + ",");
192             }
193             header += ("style,");
194             header += ("file_name\n");
195             break;
196         }
197         case AuFileProcessingAlgorithm::MFCC: {
198             header += ("SIGNALENERGY_AVG,");
199             for (std::size_t i = 0; i < MEL_APPLIED_N; i++) {
200                 header += ("BIN_AVG" + std::to_string(i) + ",");
201             }
202             header += ("SIGNALENERGY_STDEV,");
203             for (std::size_t i = 0; i < MEL_APPLIED_N; i++) {
204                 header += ("BIN_STDEV" + std::to_string(i) + ",");
205             }
206             header += ("style,");
207             header += ("file_name\n");
208             break;
209         }
210         default: {
211             LOG(LOG_ERROR) << "Error : the processing algorithm value usage is not defined in the
    project";
212             throw std::domain_error("Unsupported processing algorithm!");
213             break;
214         }
215     }
216     return header;
217 }
```

Here is the caller graph for this function:



4.2.3.7 get_data_offset()

```
word_t AuFileProcessor::get_data_offset ( ) const
```

Get the .au file data offset.

Returns

the data_offset class variable value.

Definition at line 75 of file au_file_processor.cpp.

```
75                                                  {
76     return data_offset;
77 }
```

4.2.3.8 get_data_size()

```
word_t AuFileProcessor::get_data_size ( ) const
```

Get the .au file data size.

Returns

the data_size class variable value.

Definition at line 79 of file au_file_processor.cpp.

```
79                                                 {
80     return data_size;
81 }
```

4.2.3.9 get_encoding()

```
word_t AuFileProcessor::get_encoding ( ) const
```

Get the .au file encoding format.

Returns

the encoding class variable value.

Definition at line 83 of file au_file_processor.cpp.

```
83                                                 {
84     return encoding;
85 }
```

4.2.3.10 get_features_average()

```
const real_vector_t & AuFileProcessor::get_features_average ( ) const
```

Get the .au file features average.

Returns

the features_average class variable value.

Definition at line 99 of file au_file_processor.cpp.

```
99                                                          {
100     return features_average;
101 }
```

4.2.3.11 get_features_standard_deviation()

```
const real_vector_t & AuFileProcessor::get_features_standard_deviation ( ) const
```

Get the .au file features standard deviation.

Returns

the features_standard_deviation class variable value.

Definition at line 103 of file au_file_processor.cpp.

```
103                                                                    {
104     return features_standard_deviation;
105 }
```

4.2.3.12 get_file_path()

```
const std::filesystem::path & AuFileProcessor::get_file_path ( ) const
```

Get the .au file path.

Returns

the file_path class variable value.

Definition at line 55 of file au_file_processor.cpp.

```
55                                                          {
56     return file_path;
57 }
```

4.2.3.13 get_magic_number()

```
word_t AuFileProcessor::get_magic_number ( ) const
```

Get the .au file magic number.

Returns

the magic_number class variable value.

Definition at line 71 of file au_file_processor.cpp.

```
71                                                {
72     return magic_number;
73 }
```

4.2.3.14 get_music_style()

```
const std::string & AuFileProcessor::get_music_style ( ) const
```

Get the .au file music style.

Returns

the music_style class variable value.

Definition at line 67 of file au_file_processor.cpp.

```
67                                                          {
68     return music_style;
69 }
```

4.2.3.15 get_next_data()

```
int16_t AuFileProcessor::get_next_data (
            std::ifstream & file )  [private]
```

Read next data from the file ifstream.

A data (type int16_t) is a 16 bit signed integer. The file is read byte per byte and each byte is converted from little endian to big endian.

Parameters

in	*file*	a std::ifstream object of the .au file to read.

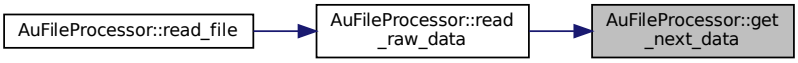
Returns

the extracted data

Definition at line 253 of file au_file_processor.cpp.

```
253                                                         {
254     int16_t data = 0;
255     uint8_t byte;
256     // Read a data (signed 16 bit) byte per byte and store it in big endian
257     for (std::size_t i = 0; i < 2; i++) {
258         file.read(reinterpret_cast<char *>(&byte), 1);
259         data = data | (byte << (((2 - 1) - i) * 8));
260     }
261     return data;
262 }
```

Here is the caller graph for this function:



4.2.3.16 get_next_word()

```
word_t AuFileProcessor::get_next_word (
            std::ifstream & file )  [private]
```

Read next word from the file ifstream.

A word (type word_t) is a 32 bit unsigned integer. The file is read byte per byte and each byte is converted from little endian to big endian.

Parameters

in	*file*	a std::ifstream object of the .au file to read.

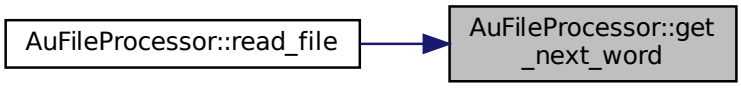
Returns

the extracted word

Definition at line 242 of file au_file_processor.cpp.

```
242                                                     {
243     word_t word = 0;
244     uint8_t byte;
245     // Read a word (unsigned 32 bit) byte per byte and store it in big endian
246     for (std::size_t i = 0; i < 4; i++) {
247         file.read(reinterpret_cast<char *>(&byte), 1);
248         word = word | (byte « (((4 - 1) - i) * 8));
249     }
250     return word;
251 }
```

Here is the caller graph for this function:



4.2.3.17 get_processing_algorithm()

```
AuFileProcessingAlgorithm AuFileProcessor::get_processing_algorithm ( ) const
```

Get the .au file current process algorithm.

Returns

the processing_algorithm class variable value.

Definition at line 59 of file au_file_processor.cpp.

```
59                                                                  {
60     return processing_algorithm;
61 }
```

4.2.3.18 get_raw_data()

```
const real_vector_t & AuFileProcessor::get_raw_data ( ) const
```

Get the .au file raw data vector.

Returns

the raw_data class variable value.

Definition at line 95 of file au_file_processor.cpp.

```
95                                                          {
96     return raw_data;
97 }
```

4.2.3.19 get_sample_rate()

```
word_t AuFileProcessor::get_sample_rate ( ) const
```

Get the .au file sample rate (in sample/sec).

Returns

the sample_rate class variable value.

Definition at line 87 of file au_file_processor.cpp.

```
87                                                  {
88     return sample_rate;
89 }
```

4.2.3.20 normalize_features()

```
void AuFileProcessor::normalize_features ( )  [private]
```

Normalize the features_average and features_standard_deviation vectors using the formula: normalized_vector = (vector - mean) / stdev.

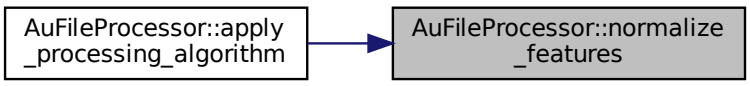
Returns

void

Definition at line 419 of file au_file_processor.cpp.

```
419                                         {
420     real_t features_size = (real_t) (this->features_average.size() +
       this->features_standard_deviation.size());
421
422     // Compute features vector mean
423     real_t sum = 0;
424     std::for_each(std::execution::seq, this->features_average.cbegin(), this->features_average.cend(),
       [&sum](real_t x) mutable {
425         sum += x;
426     });
427     std::for_each(std::execution::seq, this->features_standard_deviation.cbegin(),
       this->features_standard_deviation.cend(), [&sum](real_t x) mutable {
428         sum += x;
429     });
430     real_t mean = sum / features_size;
431
432     // Compute features vector standard deviation
433     sum = 0;
434     std::for_each(std::execution::seq, this->features_average.cbegin(), this->features_average.cend(),
       [&mean, &sum](real_t x) mutable {
435         sum += std::pow(x - mean, 2);
436     });
437     std::for_each(std::execution::seq, this->features_standard_deviation.cbegin(),
       this->features_standard_deviation.cend(), [&mean, &sum](real_t x) mutable {
438         sum += std::pow(x - mean, 2);
439     });
440     real_t stdev = std::sqrt(sum / features_size);
441
442     // Normalize features_average
443     std::transform(std::execution::seq, features_average.cbegin(), features_average.cend(),
       features_average.begin(), [&mean, &stdev](real_t x) {
444         return (x - mean) / stdev;
445     });
446     // Normalize features_standard_deviation
447     std::transform(std::execution::seq, features_standard_deviation.cbegin(),
       features_standard_deviation.cend(), features_standard_deviation.begin(), [&mean, &stdev](real_t x) {
448         return (x - mean) / stdev;
449     });
450
451 }
```

Here is the caller graph for this function:



4.2.3.21 read_file()

```
void AuFileProcessor::read_file ( )
```

Parse the .au file header and data and save them to their corresponding class variables.

The time spent to parse all the file is written in LOG_DEBUG level.

@trhow <filesystem_error("Can't open file!", std::make_error_code(std::errc::no_such_file_or_directory))> Throw an exception if the file cannot be opened @trhow <std::domain_error("Bad file encoding!")> Throw an exception if the magic number is not 0x2e736e64 (four ASCII characters ".snd")

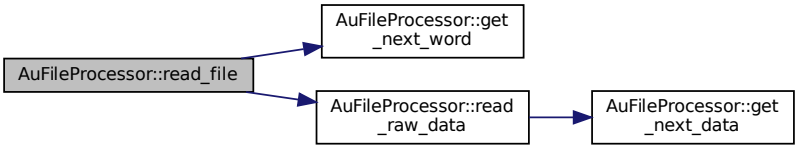
Returns

void

Definition at line 136 of file au_file_processor.cpp.

```
136                                 {
137     // Start read timer
138     auto start_time = std::chrono::high_resolution_clock::now();
139 
140     // Opening audio file
141     std::ifstream audio_file(this->file_path);
142     if (!audio_file.is_open()) {
143         LOG(LOG_ERROR) « "Error : file with path " + this->file_path.string() + "  not found.";
144         throw std::filesystem::filesystem_error("Can't open file!",
       std::make_error_code(std::errc::no_such_file_or_directory));
145     }
146     // Processing file header
147     this->magic_number = this->get_next_word(audio_file);
148     if (this->magic_number != 0x2e736e64) {
149         LOG(LOG_ERROR) « "Error : magic number does not match the required, the file encoding is not
       following the au file format";
150         throw std::domain_error("Bad file encoding!");
151     }
152     this->data_offset = this->get_next_word(audio_file);
153     this->data_size = this->get_next_word(audio_file);
154     this->encoding = this->get_next_word(audio_file);
155     this->sample_rate = this->get_next_word(audio_file);
156     this->channels = this->get_next_word(audio_file);
157     this->read_raw_data(audio_file);
158 
159     // Stop timer
160     auto stop_time = std::chrono::high_resolution_clock::now();
161 
162     if (audio_file.is_open()) {
163         audio_file.close();
164     }
165     LOG(LOG_DEBUG) « "time spent to read file " « this->file_path.filename() « ": " « (stop_time -
       start_time) / std::chrono::milliseconds(1) « "ms";
166 }
```

Here is the call graph for this function:



4.2.3.22 read_raw_data()

```
void AuFileProcessor::read_raw_data (
            std::ifstream & file )  [private]
```

Read all music data from the file ifstream.

The data is read from the data offset to the end of the file using the get_next_data class method and is stored in the raw_data class variable. The available encoding format are listed in the AuFileEncodingFormat enumeration.

Parameters

in	*file*	a std::ifstream object of the .au file to read.

@trhow <std::domain_error("Unsupported data encoding!")> Throw an exception if the encoding is not supported.

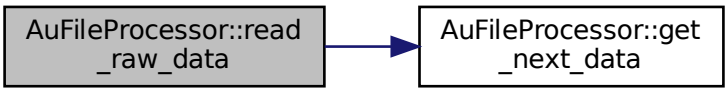
Returns

void

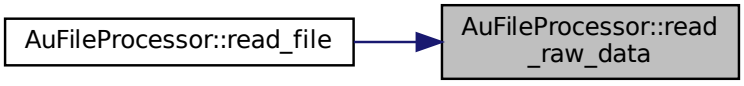
Definition at line 264 of file au_file_processor.cpp.

```
264                                                 {
265     file.seekg(this->data_offset, std::ios_base::beg);
266     std::size_t bytes_number = 0;
267     switch (this->encoding) {
268         case static_cast<word_t>(AuFileEncodingFormat::PCM_16B) : {
269             bytes_number = 2;
270             for (size_t i = 0; i < this->data_size / bytes_number; i++) {
271                 this->raw_data.push_back((real_t) this->get_next_data(file));
272             }
273             break;
274         }
275         default: {
276             LOG(LOG_ERROR) « "Error : the encoding mode value (" « this->encoding « ") usage is not
      defined in the project";
277             throw std::domain_error("Unsupported data encoding!");
278             break;
279         }
280     }
281
282     raw_data.shrink_to_fit();
283 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.2.3.23 set_processing_algorithm()

```
void AuFileProcessor::set_processing_algorithm (
            AuFileProcessingAlgorithm algorithm )
```

Set the .au current process algorithm. param[in] processing_algorithm a AuFileProcessingAlgorithm enum variable that represent the process algorithm to use.

Returns

void

Definition at line 63 of file au_file_processor.cpp.

```
63                                                                              {
64     this->processing_algorithm = algorithm;
65 }
```

4.2.4 Friends And Related Function Documentation

4.2.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const AuFileProcessor & au_file_processor )  [friend]
```

Overload the << operator for the AuFileProcessor object.

Returns

the ostream of the human readable object

Definition at line 107 of file au_file_processor.cpp.

```
107                                                                             {
108     char magic_str[5] = {
109             (char) ((au_file_processor.magic_number & 0xFF000000) » 24u),
110             (char) ((au_file_processor.magic_number & 0x00FF0000) » 16u),
111             (char) ((au_file_processor.magic_number & 0x0000FF00) » 8u),
112             (char) ((au_file_processor.magic_number & 0x000000FF) » 0u),
113             '\0'
114     };
115     std::string data_size_human_readable = {};
116     std::string size_unit;
117     if (au_file_processor.data_size < KiB) {
118         data_size_human_readable = std::to_string(au_file_processor.data_size) + "B";
119     } else if (au_file_processor.data_size < MiB) {
120         data_size_human_readable = std::to_string(au_file_processor.data_size / KiB) + "." +
       std::to_string(au_file_processor.data_size % KiB) + "KiB";
121     } else if (au_file_processor.data_size < GiB) {
122         data_size_human_readable = std::to_string(au_file_processor.data_size / MiB) + "." +
       std::to_string(au_file_processor.data_size % MiB) + "MiB";
123     } else {
124         data_size_human_readable = std::to_string(au_file_processor.data_size / GiB) + "." +
       std::to_string(au_file_processor.data_size % GiB) + "GiB";
125     }
126
127     return os « std::hex « "{magic number: 0x" « au_file_processor.magic_number « " (" « magic_str « ")"
128               « std::dec « ", data offset: " « au_file_processor.data_offset
129               « std::dec « ", data size: " « data_size_human_readable
130               « std::dec « ", encoding: " « au_file_processor.encoding
131               « std::dec « ", sample rate (sample/sec): " « au_file_processor.sample_rate
132               « std::dec « ", channels: " « au_file_processor.channels
133               « "}";
134 }
```

4.2.5 Member Data Documentation

4.2.5.1 channels

```
word_t AuFileProcessor::channels  [private]
```

Variable that store the .au file number of interleaved channels. This value is obtained through the parsing of the file header and is done in the read_file class method. Example of valid channels: 1 for mono, 2 for stereo, more channels possible, but may not be supported by all readers.

Definition at line 258 of file au_file_processor.h.

4.2.5.2 data_offset

```
word_t AuFileProcessor::data_offset  [private]
```

Variable that store the .au file data offset. This value is obtained through the parsing of the file header and is done in the read_file class method. It must be divisible by 8. The minimum valid number is 24, since this is the header length (six 32-bit words) with no space reserved for extra information (the annotation field). The minimum valid number with an annotation field present is 32 (decimal).

Definition at line 234 of file au_file_processor.h.

4.2.5.3 data_size

```
word_t AuFileProcessor::data_size  [private]
```

Variable that store the .au file data size. This value is obtained through the parsing of the file header and is done in the read_file class method. If the value is 0xffffffff (4294967295), it means that the data ize is unknown.

Definition at line 240 of file au_file_processor.h.

4.2.5.4 DEFAULT_PROCESSING_ALGORITHM

```
const AuFileProcessingAlgorithm AuFileProcessor::DEFAULT_PROCESSING_ALGORITHM = AuFileProcessingAlgorithm::STF
[static], [private]
```

Constant that store the .au file default process algorithm (STFT)

Definition at line 210 of file au_file_processor.h.

4.2.5.5 encoding

```
word_t AuFileProcessor::encoding  [private]
```

Variable that store the .au file encoding format. This value is obtained through the parsing of the file header and is done in the read_file class method. It must be a value available in the AuFileEncodingFormat enum or the file will not be processable.

Definition at line 246 of file au_file_processor.h.

4.2.5.6 features_average

```
real_vector_t AuFileProcessor::features_average {}  [private]
```

Variable that store the.au file features average in a vector. This is a vector of size SIZE_FFT(for STFT) or CEPS↩ TRAL_COEFF(for MFCC) that contain the average of all frequencies after the stft/mfcc have been computed.

Definition at line 269 of file au_file_processor.h.

4.2.5.7 features_standard_deviation

```
real_vector_t AuFileProcessor::features_standard_deviation {}  [private]
```

Variable that store the .au file features standard deviation in a vector. This is a vector of size SIZE_FFT(for STFT) or CEPSTRAL_COEFF(for MFCC) that contain the standard deviation of all frequencies after the stft/mfcc have been computed.

Definition at line 274 of file au_file_processor.h.

4.2.5.8 file_path

```
std::filesystem::path AuFileProcessor::file_path  [private]
```

Variable that store the .au file path.

Definition at line 206 of file au_file_processor.h.

4.2.5.9 magic_number

```
word_t AuFileProcessor::magic_number  [private]
```

Variable that store the .au file magic number. This value is obtained through the parsing of the file header and is done in the read_file class method. Should be 0x2e736e64 (four ASCII characters ".snd").

Definition at line 226 of file au_file_processor.h.

4.2.5.10 music_style

```
std::string AuFileProcessor::music_style  [private]
```

Variable that store the .au file music style. This value is obtained through the parsing of the file name and is done the object Constructor.

Definition at line 219 of file au_file_processor.h.

4.2.5.11 processing_algorithm

```
AuFileProcessingAlgorithm AuFileProcessor::processing_algorithm  [private]
```

Variable that store the .au file current process algorithm.

Definition at line 214 of file au_file_processor.h.

4.2.5.12 raw_data

```
real_vector_t AuFileProcessor::raw_data  [private]
```

Variable that store the .au file raw data in a vector. This vector is obtained through the parsing of the file data and is done in the read_file class method.

Definition at line 264 of file au_file_processor.h.

4.2.5.13 sample_rate

```
word_t AuFileProcessor::sample_rate  [private]
```

Variable that store the .au file sample rate (in sample/sec). This value is obtained through the parsing of the file header and is done in the read_file class method.

Definition at line 252 of file au_file_processor.h.

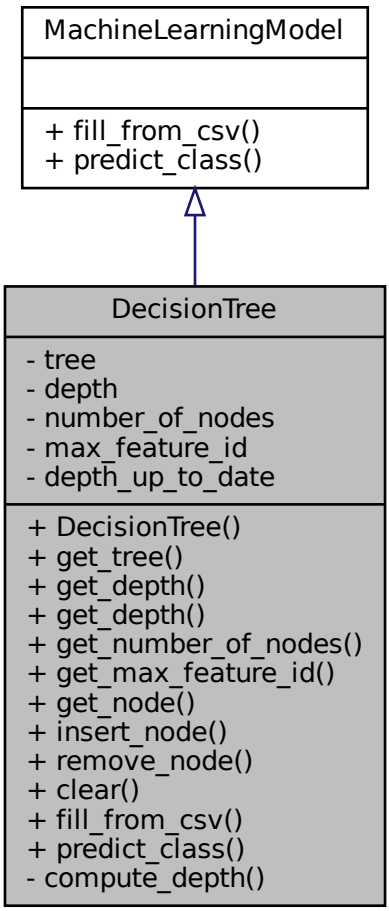
The documentation for this class was generated from the following files:

- extraction/au_file_processor.h
- extraction/au_file_processor.cpp

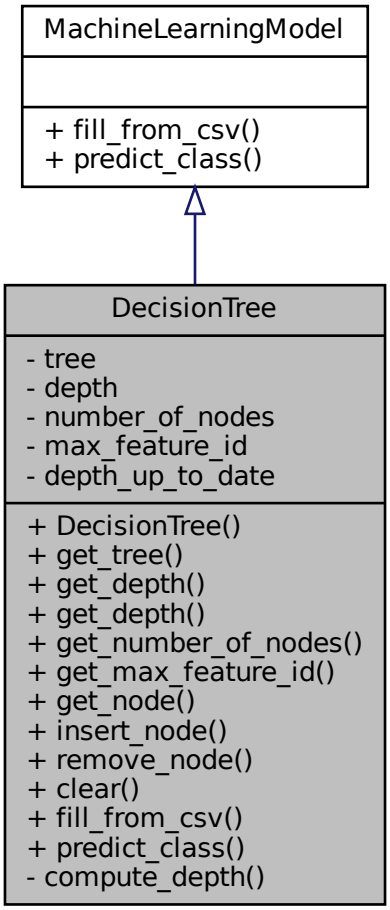
4.3 DecisionTree Class Reference

```
#include <decision_tree.h>
```

Inheritance diagram for DecisionTree:



Collaboration diagram for DecisionTree:



Public Member Functions

- DecisionTree ()
- const std::map< std::size_t, TreeNode > & get_tree () const
- std::size_t get_depth () const
- std::size_t get_depth ()
- std::size_t get_number_of_nodes () const
- int get_max_feature_id () const
- TreeNode get_node (std::size_t node_id) const
- void insert_node (std::size_t node_id, const TreeNode &node)
- void remove_node (std::size_t node_id)
- void clear ()
- void fill_from_csv (const std::filesystem::path &csv_file_path) override
- std::string predict_class (const real_vector_t &features_vector) override

Private Member Functions

- std::size_t compute_depth () const

Private Attributes

- std::map< std::size_t, TreeNode > tree
- std::size_t depth
- std::size_t number_of_nodes
- int max_feature_id
- bool depth_up_to_date

Friends

- std::ostream & operator<< (std::ostream &os, const DecisionTree &decision_tree)

4.3.1 Detailed Description

Definition at line 43 of file decision_tree.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 DecisionTree()

```
DecisionTree::DecisionTree ( )
```

Definition at line 55 of file decision_tree.cpp.

```
55                              {
56     this->tree.clear();
57     this->depth = 0;
58     this->number_of_nodes = 0;
59     this->depth_up_to_date = true;
60     this->max_feature_id = 0;
61 }
```

4.3.3 Member Function Documentation

4.3.3.1 clear()

```
void DecisionTree::clear ( )
```

Definition at line 137 of file decision_tree.cpp.

```
137                             {
138     this->tree.clear();
139
140     this->depth = 0;
141     this->depth_up_to_date = true;
142     this->number_of_nodes = 0;
143     this->max_feature_id = 0;
144 }
```

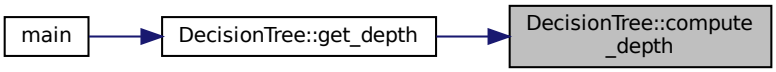
4.3.3.2 compute_depth()

```
std::size_t DecisionTree::compute_depth ( ) const  [private]
```

Definition at line 223 of file decision_tree.cpp.

```
223                                          {
224     // function <return_type(parameter_types)> function_name
225     std::function<int(TreeNode)> compute_tree_depth = [this, &compute_tree_depth](const TreeNode &node)
        {
226         int lh = (node.get_left_children_id() == -1) ? 0 :
        compute_tree_depth(this->tree.at(node.get_left_children_id()));
227         int rh = (node.get_right_children_id() == -1) ? 0 :
        compute_tree_depth(this->tree.at(node.get_right_children_id()));
228         return (std::size_t) std::max(lh, rh) + 1;
229     };
230     //TODO: parallelize the depth computation
231     if (this->tree.empty()) {
232         return 0;
233     } else {
234         return compute_tree_depth(this->tree.at(0));
235     }
236 }
```

Here is the caller graph for this function:



4.3.3.3 fill_from_csv()

```
void DecisionTree::fill_from_csv (
            const std::filesystem::path & csv_file_path )  [override], [virtual]
```

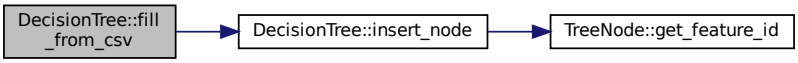
Implements MachineLearningModel.

Definition at line 146 of file decision_tree.cpp.

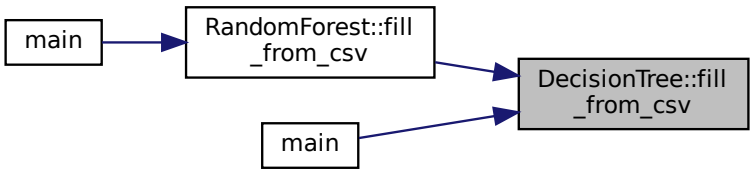
```
146                                                                                 {
147     const char delimiter = ',';
148     std::string line = {};
149     std::string data = {};
150
151     std::ifstream input_file(csv_file_path);
152     if (!input_file.is_open()) {
153         LOG(LOG_ERROR) << "Error : file with path " + csv_file_path.string() + "  not found.";
154         throw std::filesystem::filesystem_error("Can't open file!",
        std::make_error_code(std::errc::no_such_file_or_directory));
155     }
156
157     //TODO: check file extension and header
158
159     bool header_skipped = false;
160     while (std::getline(input_file, line)) {
161         if (!header_skipped) {
162             header_skipped = true;
163             continue;
164         } else {
165             std::stringstream ss(line);
166             size_t last = 0;
167             size_t next = 0;
168             // Get tree node id from line
169             next = line.find(delimiter, last);
```

```
170             int node_id = std::stoi(line.substr(last, next - last));
171             last = next + 1;
172             // Get threshold from line
173             next = line.find(delimiter, last);
174             real_t threshold = (real_t) std::stod(line.substr(last, next - last));
175             last = next + 1;
176             // Get the feature id from line
177             next = line.find(delimiter, last);
178             int feature_id = std::stoi(line.substr(last, next - last));
179             last = next + 1;
180             // Get left children id from line
181             next = line.find(delimiter, last);
182             int left_children_id = std::stoi(line.substr(last, next - last));
183             last = next + 1;
184             // Get right children id from line
185             next = line.find(delimiter, last);
186             int right_children_id = std::stoi(line.substr(last, next - last));
187             last = next + 1;
188             // Get class name from line
189             std::string class_name = line.substr(last);
190             // Remove double quote characters around the class name
191             class_name.erase(remove(class_name.begin(), class_name.end(), '"'), class_name.end());
192             TreeNode new_tree = {class_name, threshold, feature_id, left_children_id,
    right_children_id};
193             this->insert_node(node_id, new_tree);
194         }
195     }
196
197     this->depth_up_to_date = false;
198 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.3.4 get_depth() [1/2]

```
std::size_t DecisionTree::get_depth ( )
```

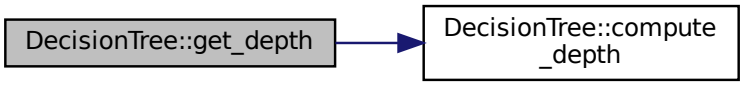
4.3.3.5 get_depth() [2/2]

```
size_t DecisionTree::get_depth ( ) const
```

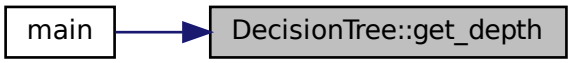
Definition at line 67 of file decision_tree.cpp.

```
67                                  {
68     if (this->depth_up_to_date) {
69         return this->depth;
70     } else {
71         this->depth = this->compute_depth();
72         this->depth_up_to_date = true;
73         return this->depth;
74     }
75 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.3.3.6 get_max_feature_id()

```
int DecisionTree::get_max_feature_id ( ) const
```

Definition at line 89 of file decision_tree.cpp.

```
89                                          {
90     return this->max_feature_id;
91 }
```

4.3.3.7 get_node()

```
TreeNode DecisionTree::get_node (
            std::size_t node_id ) const
```

Definition at line 93 of file decision_tree.cpp.

```
93                                                      {
94     return this->tree.at(node_id);
95 }
```

4.3.3.8 get_number_of_nodes()

```
std::size_t DecisionTree::get_number_of_nodes ( ) const
```

Definition at line 85 of file decision_tree.cpp.

```
85                                                  {
86     return number_of_nodes;
87 }
```

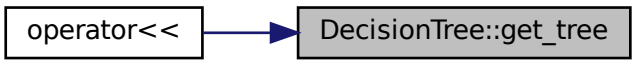
4.3.3.9 get_tree()

```
const std::map< std::size_t, TreeNode > & DecisionTree::get_tree ( ) const
```

Definition at line 63 of file decision_tree.cpp.

```
63                                                                  {
64     return this->tree;
65 }
```

Here is the caller graph for this function:



4.3.3.10 insert_node()

```
void DecisionTree::insert_node (
            std::size_t node_id,
            const TreeNode & node )
```

Definition at line 104 of file decision_tree.cpp.

```
104                                                                         {
105     if (this->tree.count(node_id) == 1) {
106         if (node_id == 0) {
107             LOG(LOG_ERROR) << "Error : trying to insert a node with id=0 but the tree already as a root
        node";
108             throw std::invalid_argument("Tree already have a root node!");
109         } else {
110             LOG(LOG_ERROR) << "Error : trying to insert a node with id=" + std::to_string(node_id) + "
        but the tree already as a node with this id";
111             throw std::invalid_argument("Tree already have a node with id " + std::to_string(node_id) +
        "!");
112         }
113     }
114     this->tree.insert(std::make_pair(node_id, node));
115     this->max_feature_id = std::max(node.get_feature_id(), this->max_feature_id);
116
117     this->depth_up_to_date = false;
118     this->number_of_nodes += 1;
119 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.3.3.11 predict_class()

```
std::string DecisionTree::predict_class (
            const real_vector_t & features_vector )  [override], [virtual]
```

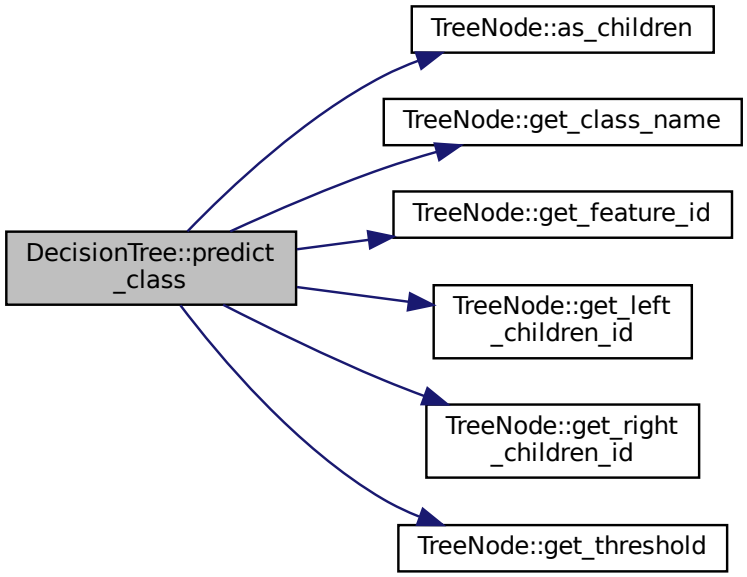
Implements MachineLearningModel.

Definition at line 200 of file decision_tree.cpp.

```
200                                                                         {
```

```
201     // Verify that the tree does not contain a feature id bigger than the feature vector size
202     if ((int) features_vector.size() < this->max_feature_id) {
203         LOG(LOG_ERROR) << "Error : trying to make prediction but the tree contain a feature id bigger
        than the feature vector size (" << features_vector.size() << " < " << this->max_feature_id << ")";
204         throw std::invalid_argument("Feature vector too small!");
205     }
206
207     // Current node point to root node
208     TreeNode *current_node = &this->tree.at(0);
209
210     while (current_node->as_children() && current_node->get_feature_id() >= 0) {
211         // Set current node to the left or right node if the features vector feature at the given id is
        bellow the current node threshold
212         if (features_vector.at(current_node->get_feature_id()) <= current_node->get_threshold()) {
213             current_node = &this->tree.at(current_node->get_left_children_id());
214         } else {
215             current_node = &this->tree.at(current_node->get_right_children_id());
216         }
217     }
218
219     return current_node->get_class_name();
220 }
```

Here is the call graph for this function:



4.3.3.12 remove_node()

```
void DecisionTree::remove_node (
            std::size_t node_id )
```

Definition at line 121 of file decision_tree.cpp.

```
121                                         {
122     if (this->tree.count(node_id) == 0) {
123         LOG(LOG_ERROR) << "Error : trying to remove a node with id=" + std::to_string(node_id) + " but
        the tree does not have a node with this id";
124         throw std::invalid_argument("Tree does not have a node with id " + std::to_string(node_id) +
        "!");
```

```
125     }
126     if (this->tree.at(node_id).as_children()) {
127         LOG(LOG_ERROR) << "Error : trying to remove the node with id=" + std::to_string(node_id) + " but
        this node as at least one children, remove children nodes first (left id=" +
        std::to_string(this->tree.at(node_id).get_left_children_id()) + ", right_id=" +
        std::to_string(this->tree.at(node_id).get_right_children_id()) + " !";
128         throw std::invalid_argument("Node as children!");
129
130     }
131     this->tree.erase(node_id);
132
133     this->depth_up_to_date = false;
134     this->number_of_nodes -= 1;
135 }
```

4.3.4 Friends And Related Function Documentation

4.3.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const DecisionTree & decision_tree )  [friend]
```

Definition at line 97 of file decision_tree.cpp.

```
97                                                                                  {
98     for (const std::pair<const unsigned long, TreeNode> &tree: decision_tree.get_tree()) {
99         os << "\n- node[" << tree.first << "]: " << tree.second;
100     }
101     return os;
102 }
```

4.3.5 Member Data Documentation

4.3.5.1 depth

```
std::size_t DecisionTree::depth  [private]
```

Definition at line 75 of file decision_tree.h.

4.3.5.2 depth_up_to_date

```
bool DecisionTree::depth_up_to_date  [private]
```

Definition at line 78 of file decision_tree.h.

4.3.5.3 max_feature_id

```
int DecisionTree::max_feature_id  [private]
```

Definition at line 77 of file decision_tree.h.

4.3.5.4 number_of_nodes

```
std::size_t DecisionTree::number_of_nodes  [private]
```

Definition at line 76 of file decision_tree.h.

4.3.5.5 tree

```
std::map<std::size_t, TreeNode> DecisionTree::tree  [private]
```

Definition at line 74 of file decision_tree.h.

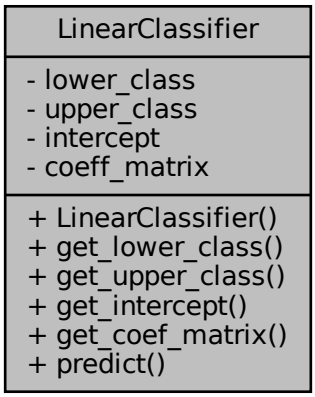
The documentation for this class was generated from the following files:

- ml_algorithms/decision_tree.h
- ml_algorithms/decision_tree.cpp

4.4 LinearClassifier Class Reference

```
#include <one_vs_one_svm.h>
```

Collaboration diagram for LinearClassifier:



Public Member Functions

- LinearClassifier (std::string lower_class, std::string upper_class, real_t intercept, real_vector_t coef_matrix)
- const std::string & get_lower_class () const
- const std::string & get_upper_class () const
- real_t get_intercept () const
- const real_vector_t & get_coef_matrix () const
- std::string predict (const real_vector_t &features_vector)

Private Attributes

- std::string lower_class
- std::string upper_class
- real_t intercept
- real_vector_t coeff_matrix

Friends

- std::ostream & operator<< (std::ostream &os, const LinearClassifier &linear_classifier)

4.4.1 Detailed Description

Definition at line 13 of file one_vs_one_svm.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 LinearClassifier()

```
LinearClassifier::LinearClassifier (
            std::string lower_class,
            std::string upper_class,
            real_t intercept,
            real_vector_t coef_matrix )
```

Definition at line 13 of file one_vs_one_svm.cpp.

```
13
                        :
14          lower_class(std::move(lower_class)), upper_class(std::move(upper_class)), intercept(intercept),
        coeff_matrix(std::move(coef_matrix)) {
15 }
```

4.4.3 Member Function Documentation

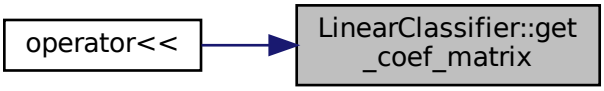
4.4.3.1 get_coef_matrix()

```
const real_vector_t & LinearClassifier::get_coef_matrix ( ) const
```

Definition at line 29 of file one_vs_one_svm.cpp.

```
29                                                          {
30     return coeff_matrix;
31 }
```

Here is the caller graph for this function:



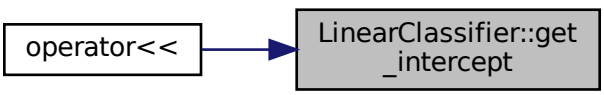
4.4.3.2 get_intercept()

```
real_t LinearClassifier::get_intercept ( ) const
```

Definition at line 25 of file one_vs_one_svm.cpp.

```
25                                                  {
26     return intercept;
27 }
```

Here is the caller graph for this function:



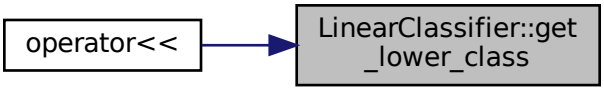
4.4.3.3 get_lower_class()

```
const std::string & LinearClassifier::get_lower_class ( ) const
```

Definition at line 17 of file one_vs_one_svm.cpp.

```
17                                                          {
18     return lower_class;
19 }
```

Here is the caller graph for this function:



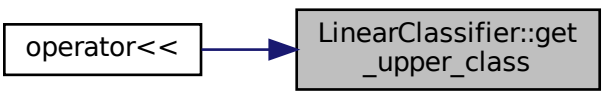
4.4.3.4 get_upper_class()

```
const std::string & LinearClassifier::get_upper_class ( ) const
```

Definition at line 21 of file one_vs_one_svm.cpp.

```
21                                                          {
22     return upper_class;
23 }
```

Here is the caller graph for this function:



4.4.3.5 predict()

```
std::string LinearClassifier::predict (
            const real_vector_t & features_vector )
```

Definition at line 44 of file one_vs_one_svm.cpp.

```
44                                                                          {
45     LOG(LOG_DEBUG) << "fv: " << features_vector;
46     LOG(LOG_DEBUG) << "intercept: " << intercept << ", coeff_matrix: " << this->coeff_matrix;
47     if (features_vector.size() != this->coeff_matrix.size()) {
48         LOG(LOG_ERROR) << "Error : trying to make prediction but the feature vector size (" <<
       features_vector.size() << ") is different from the coefficient matrix size (" <<
       this->coeff_matrix.size() << ")";
49         throw std::invalid_argument("Feature vector size differ from coefficient matrix size!");
50     }
51
52     real_t result = 0;
53     std::for_each(std::execution::seq, this->coeff_matrix.cbegin(), this->coeff_matrix.cend(),
       [&features_vector, &result, i = 0](real_t x) mutable {
54         result += features_vector.at(i) * x;
55         i++;
56     });
57     result += this->intercept;
58
59     LOG(LOG_DEBUG) << this->upper_class << " (" << result << ") " << this->lower_class;
60     return (result > 0 ? this->lower_class : this->upper_class);
61 }
```

4.4.4 Friends And Related Function Documentation

4.4.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const LinearClassifier & linear_classifier )  [friend]
```

Definition at line 33 of file one_vs_one_svm.cpp.

```
33                                                                                  {
34     // A for_each can be used here because the size of the matrix_a string will not be too long
35     size_t coef_matrix_len = linear_classifier.get_coef_matrix().size();
36     std::string matrix_a = "[";
37     std::for_each(std::execution::seq, linear_classifier.get_coef_matrix().cbegin(),
       linear_classifier.get_coef_matrix().cend(), [&matrix_a, &coef_matrix_len, i = 0](real_t r)mutable {
38         matrix_a += (i < int(coef_matrix_len)) ? std::to_string(r) + ", " : std::to_string(r);
39     });
40     matrix_a += "]";
41     return os << "y >= Ax+b -> class id is " << linear_classifier.get_upper_class() << " else class id is" <<
       linear_classifier.get_lower_class() << " with A=" << matrix_a << " and b=" <<
       std::to_string(linear_classifier.get_intercept());
42 }
```

4.4.5 Member Data Documentation

4.4.5.1 coeff_matrix

```
real_vector_t LinearClassifier::coeff_matrix  [private]
```

Definition at line 33 of file one_vs_one_svm.h.

4.4.5.2 intercept

```
real_t LinearClassifier::intercept  [private]
```

Definition at line 32 of file one_vs_one_svm.h.

4.4.5.3 lower_class

```
std::string LinearClassifier::lower_class  [private]
```

Definition at line 30 of file one_vs_one_svm.h.

4.4.5.4 upper_class

```
std::string LinearClassifier::upper_class  [private]
```

Definition at line 31 of file one_vs_one_svm.h.

The documentation for this class was generated from the following files:

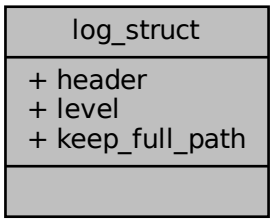
- ml_algorithms/one_vs_one_svm.h
- ml_algorithms/one_vs_one_svm.cpp

4.5 log_struct Struct Reference

Hold structure for log config.

```
#include <log.h>
```

Collaboration diagram for log_struct:



Public Attributes

- bool header = true
- log_level_enum level = LOG_DEBUG
- bool keep_full_path = false

4.5.1 Detailed Description

Hold structure for log config.

Definition at line 25 of file log.h.

4.5.2 Member Data Documentation

4.5.2.1 header

```
bool log_struct::header = true
```

True to print a header before the log message; False else

Definition at line 26 of file log.h.

4.5.2.2 keep_full_path

```
bool log_struct::keep_full_path = false
```

True to keep full path from **FILE**; False to only keep filename and extension

Definition at line 28 of file log.h.

4.5.2.3 level

```
log_level_enum log_struct::level = LOG_DEBUG
```

Default Log level

Definition at line 27 of file log.h.

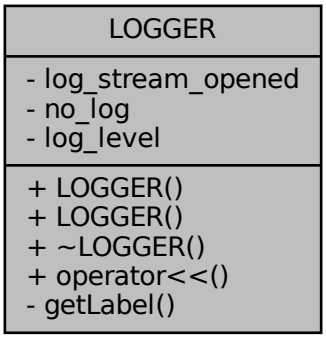
The documentation for this struct was generated from the following file:

- helpers/log.h

4.6 LOGGER Class Reference

`#include <log.h>`

Collaboration diagram for LOGGER:



Public Member Functions

- LOGGER ()
- LOGGER (log_level_enum level, const std::string &file, int line, const std::string &function)
- ~LOGGER ()
- template<class T >
  LOGGER & operator<< (const T &msg)

Private Member Functions

- std::string getLabel (log_level_enum level)

Private Attributes

- bool log_stream_opened = false
- bool no_log = false
- log_level_enum log_level = LOG_DEBUG

4.6.1 Detailed Description

Definition at line 33 of file log.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 LOGGER() [1/2]

```
LOGGER::LOGGER ( )  [inline]
```

Definition at line 35 of file log.h.

```
35              {
36          no_log = true;
37      }
```

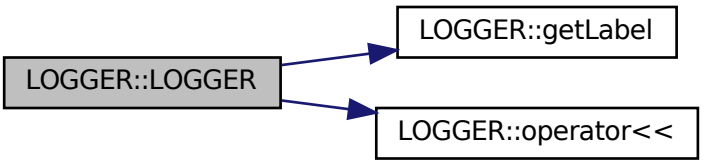
4.6.2.2 LOGGER() [2/2]

```
LOGGER::LOGGER (
              log_level_enum level,
              const std::string & file,
              int line,
              const std::string & function )  [inline]
```

Definition at line 39 of file log.h.

```
39                                                                                                  {
40          no_log = false;
41          log_level = level;
42
43          if (LOGGING_CONFIG.header) {
44              if (LOGGING_CONFIG.keep_full_path) {
45                  operator«("[" + getLabel(level) + "] " + file + ":" + std::to_string(line) + ":" +
       function + "()\t");
46              } else {
47                  operator«("[" + getLabel(level) + "] " + file.substr(file.find_last_of("/\\") + 1) + ":"
       + std::to_string(line) + ":" + function + "()\t");
48              }
49          }
50      }
```

Here is the call graph for this function:



4.6.2.3 ~LOGGER()

```
LOGGER::~LOGGER ( )  [inline]
```

Definition at line 52 of file log.h.

```
52              {
53          if (log_stream_opened) {
54              std::cout « std::endl;
55          }
56          log_stream_opened = false;
57      }
```

4.6.3 Member Function Documentation

4.6.3.1 getLabel()

```
std::string LOGGER::getLabel (
            log_level_enum level )  [inline], [private]
```

Definition at line 73 of file log.h.

```
73                                                  {
74         std::string log_level_label;
75         switch (level) {
76             case LOG_DEBUG: {
77                 log_level_label = "DEBUG";
78                 break;
79             }
80             case LOG_INFO: {
81                 log_level_label = "INFO";
82                 break;
83             }
84             case LOG_WARNING: {
85                 log_level_label = "WARNING";
86                 break;
87             }
88             case LOG_ERROR: {
89                 log_level_label = "ERROR";
90                 break;
91             }
92         }
93         return log_level_label;
94     }
```

Here is the caller graph for this function:

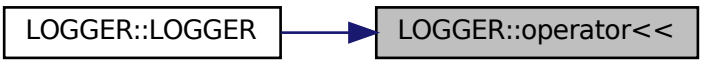
4.6.3.2 operator<<()

```
template<class T >
LOGGER& LOGGER::operator<< (
            const T & msg )  [inline]
```

Definition at line 60 of file log.h.

```
60                                      {
61         if (log_level >= LOGGING_CONFIG.level && !no_log) {
62             std::cout « msg;
63             log_stream_opened = true;
64         }
65         return *this;
66     }
```

Here is the caller graph for this function:



4.6.4 Member Data Documentation

4.6.4.1 log_level

```
log_level_enum LOGGER::log_level = LOG_DEBUG  [private]
```

Definition at line 71 of file log.h.

4.6.4.2 log_stream_opened

```
bool LOGGER::log_stream_opened = false  [private]
```

Definition at line 69 of file log.h.

4.6.4.3 no_log

```
bool LOGGER::no_log = false  [private]
```

Definition at line 70 of file log.h.

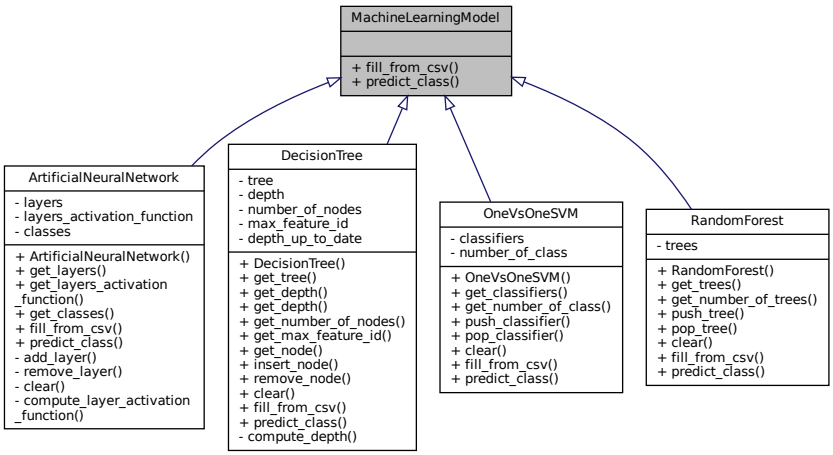
The documentation for this class was generated from the following file:

- helpers/log.h

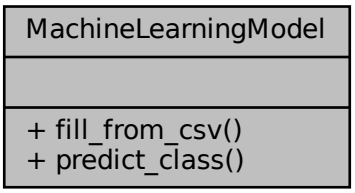
4.7 MachineLearningModel Class Reference

`#include <machine_learning_model.h>`

Inheritance diagram for MachineLearningModel:



Collaboration diagram for MachineLearningModel:



Public Member Functions

- virtual void fill_from_csv (const std::filesystem::path &csv_folder_path)=0
- virtual std::string predict_class (const real_vector_t &features_vector)=0

4.7.1 Detailed Description

Definition at line 8 of file machine_learning_model.h.

4.7.2 Member Function Documentation

4.7.2.1 fill_from_csv()

```
virtual void MachineLearningModel::fill_from_csv (
            const std::filesystem::path & csv_folder_path )  [pure virtual]
```

Implemented in RandomForest, DecisionTree, OneVsOneSVM, and ArtificialNeuralNetwork.

4.7.2.2 predict_class()

```
virtual std::string MachineLearningModel::predict_class (
            const real_vector_t & features_vector )  [pure virtual]
```

Implemented in DecisionTree, OneVsOneSVM, ArtificialNeuralNetwork, and RandomForest.

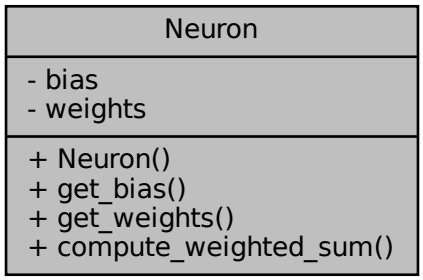
The documentation for this class was generated from the following file:

- ml_algorithms/machine_learning_model.h

4.8 Neuron Class Reference

```
#include <artificial_neural_network.h>
```

Collaboration diagram for Neuron:



Public Member Functions

- Neuron (real_t bias, real_vector_t weights)
- real_t get_bias () const
- const real_vector_t & get_weights () const
- real_t compute_weighted_sum (real_vector_t &features_vector)

Private Attributes

- real_t bias
- real_vector_t weights

Friends

- std::ostream & operator<< (std::ostream &os, const Neuron &neuron)

4.8.1 Detailed Description

Definition at line 20 of file artificial_neural_network.h.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 Neuron()

```
Neuron::Neuron (
            real_t bias,
            real_vector_t weights )
```

Definition at line 15 of file artificial_neural_network.cpp.

```
15                                                  :
16         bias(bias), weights(std::move(weights)) {}
```

4.8.3 Member Function Documentation

4.8.3.1 compute_weighted_sum()

```
real_t Neuron::compute_weighted_sum (
            real_vector_t & features_vector )
```

Definition at line 30 of file artificial_neural_network.cpp.

```
30                                                                  {
31     if (features_vector.size() != this->weights.size()) {
32         LOG(LOG_ERROR) « "Error : trying to compute the activation of a neuron but the feature vector
       size (" « features_vector.size() « ") is different from the neuron weights size (" «
       this->weights.size() « ")";
33         throw std::invalid_argument("Feature vector size differ from neuron weights size!");
34     }
35
36     real_t weighted_sum = 0.0;
37     std::for_each(std::execution::seq, this->weights.cbegin(), this->weights.cend(), [&features_vector,
       &weighted_sum, i = 0](real_t x) mutable {
38         weighted_sum += features_vector.at(i) * x;
39         i++;
40     });
41     weighted_sum += this->bias;
42
43     return weighted_sum;
44 }
```

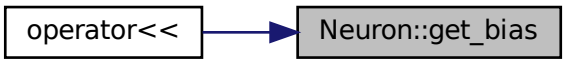
4.8.3.2 get_bias()

```
real_t Neuron::get_bias ( ) const
```

Definition at line 18 of file artificial_neural_network.cpp.

```
18                                {
19     return bias;
20 }
```

Here is the caller graph for this function:



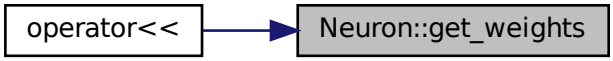
4.8.3.3 get_weights()

```
const real_vector_t & Neuron::get_weights ( ) const
```

Definition at line 22 of file artificial_neural_network.cpp.

```
22                                                {
23     return weights;
24 }
```

Here is the caller graph for this function:



4.8.4 Friends And Related Function Documentation

4.8.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const Neuron & neuron )  [friend]
```

Definition at line 26 of file artificial_neural_network.cpp.

```
26                                                          {
27     return os « "(bias: " « neuron.get_bias() « ", weights (" « neuron.get_weights().size() « "): " «
       neuron.get_weights() « ")";
28 }
```

4.8.5 Member Data Documentation

4.8.5.1 bias

```
real_t Neuron::bias  [private]
```

Definition at line 33 of file artificial_neural_network.h.

4.8.5.2 weights

```
real_vector_t Neuron::weights  [private]
```

Definition at line 34 of file artificial_neural_network.h.

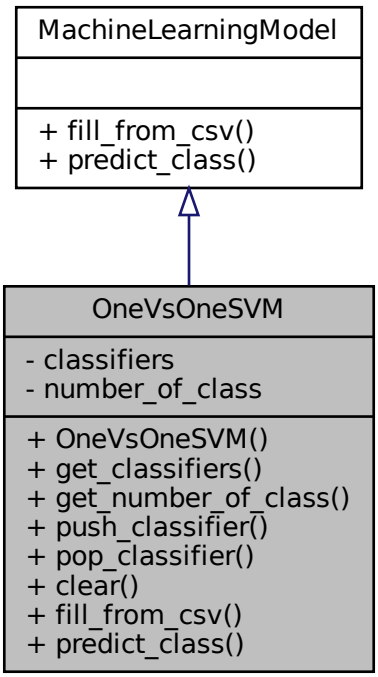
The documentation for this class was generated from the following files:

- ml_algorithms/artificial_neural_network.h
- ml_algorithms/artificial_neural_network.cpp

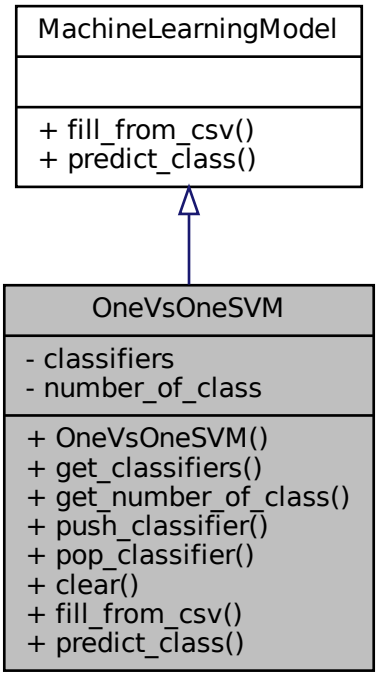
4.9 OneVsOneSVM Class Reference

`#include <one_vs_one_svm.h>`

Inheritance diagram for OneVsOneSVM:



Collaboration diagram for OneVsOneSVM:



Public Member Functions

- OneVsOneSVM ()
- const std::vector< LinearClassifier > & get_classifiers () const
- size_t get_number_of_class () const
- void push_classifier (const LinearClassifier &linear_classifier)
- void pop_classifier ()
- void clear ()
- void fill_from_csv (const std::filesystem::path &csv_file_path) override
- std::string predict_class (const real_vector_t &features_vector) override

Private Attributes

- std::vector< LinearClassifier > classifiers
- std::size_t number_of_class

Friends

- std::ostream & operator<< (std::ostream &os, const OneVsOneSVM &one_vs_one_svm)

4.9.1 Detailed Description

Definition at line 39 of file one_vs_one_svm.h.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 OneVsOneSVM()

```
OneVsOneSVM::OneVsOneSVM ( )
```

Definition at line 71 of file one_vs_one_svm.cpp.

```
71                                {
72     this->classifiers.clear();
73     this->number_of_class = 0;
74 }
```

4.9.3 Member Function Documentation

4.9.3.1 clear()

```
void OneVsOneSVM::clear ( )
```

Definition at line 103 of file one_vs_one_svm.cpp.

```
103                           {
104     this->classifiers.clear();
105 }
```

4.9.3.2 fill_from_csv()

```
void OneVsOneSVM::fill_from_csv (
             const std::filesystem::path & csv_file_path )  [override], [virtual]
```

Implements MachineLearningModel.

Definition at line 107 of file one_vs_one_svm.cpp.

```
107                                                                  {
108     const char delimiter = ',';
109     std::string line = {};
110     std::string data = {};
111 
112     std::ifstream input_file(csv_file_path);
113     if (!input_file.is_open()) {
114         LOG(LOG_ERROR) << "Error : file with path " + csv_file_path.string() + "  not found.";
115         throw std::filesystem::filesystem_error("Can't open file!",
       std::make_error_code(std::errc::no_such_file_or_directory));
116     }
117 
118     //TODO: check file extension and header
119 
120     bool header_skipped = false;
```

```
121     while (std::getline(input_file, line)) {
122         if (!header_skipped) {
123             header_skipped = true;
124             continue;
125         } else {
126             std::stringstream ss(line);
127             size_t last = 0;
128             size_t next = 0;
129             // Get class if prediction result is positive from line
130             next = line.find(delimiter, last);
131             std::string positive_class = line.substr(last, next - last);
132             // Remove double quote characters around the class name
133             positive_class.erase(remove(positive_class.begin(), positive_class.end(), '"'),
    positive_class.end());
134             last = next + 1;
135             // Get class if prediction result is negative from line
136             next = line.find(delimiter, last);
137             std::string negative_class = line.substr(last, next - last);
138             // Remove double quote characters around the class name
139             negative_class.erase(remove(negative_class.begin(), negative_class.end(), '"'),
    negative_class.end());
140             last = next + 1;
141             // Get the intercept from line
142             next = line.find(delimiter, last);
143             real_t intercept = (real_t) std::stod(line.substr(last, next - last));
144             last = next + 1;
145             // Get the coeff matrix from line
146             real_vector_t coeff_matrix;
147             while ((next = line.find(delimiter, last)) != std::string::npos) {
148                 coeff_matrix.push_back((real_t) std::stod(line.substr(last, next - last)));
149                 last = next + 1;
150             }
151             coeff_matrix.push_back((real_t) std::stod(line.substr(last)));
152 
153             LinearClassifier new_linear_classifier = {positive_class, negative_class, intercept,
    coeff_matrix};
154             this->push_classifier(new_linear_classifier);
155         }
156     }
157 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

4.9.3.3 get_classifiers()

```
const std::vector< LinearClassifier > & OneVsOneSVM::get_classifiers ( ) const
```

Definition at line 76 of file one_vs_one_svm.cpp.

```
76                                                                  {
77     return classifiers;
78 }
```

Here is the caller graph for this function:

4.9.3.4 get_number_of_class()

```
size_t OneVsOneSVM::get_number_of_class ( ) const
```

Definition at line 80 of file one_vs_one_svm.cpp.

```
80                                                  {
81     return number_of_class;
82 }
```

4.9.3.5 pop_classifier()

```
void OneVsOneSVM::pop_classifier ( )
```

Definition at line 99 of file one_vs_one_svm.cpp.

```
99                                  {
100     this->classifiers.pop_back();
101 }
```

4.9.3.6 predict_class()

```
std::string OneVsOneSVM::predict_class (
              const real_vector_t & features_vector )  [override], [virtual]
```

Implements MachineLearningModel.

Definition at line 159 of file one_vs_one_svm.cpp.

```
159                                                                                 {
160     std::map<std::string, std::size_t> pred_results;
161     using pred_results_pair_t = decltype(pred_results)::value_type;
162 
163     // Get results for each linear classifier
164     std::for_each(std::execution::seq, this->classifiers.begin(), this->classifiers.end(),
       [&pred_results, &features_vector](LinearClassifier &linear_classifier) mutable {
165         std::string pred_result = linear_classifier.predict(features_vector);
166         if (pred_results.count(pred_result) == 0) {
```

```
167             pred_results.insert(std::make_pair(pred_result, 1));
168         } else {
169             pred_results.at(pred_result) += 1;
170         }
171     });
172
173     // Get the prediction result with the most choice
174     auto pr = std::max_element(std::execution::seq, pred_results.cbegin(), pred_results.cend(),
175                                [](const pred_results_pair_t &p1, const pred_results_pair_t &p2) {
176                                    return p1.second < p2.second;
177                                });
178
179     return pr->first;
180 }
```

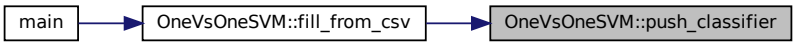
4.9.3.7 push_classifier()

```
void OneVsOneSVM::push_classifier (
            const LinearClassifier & linear_classifier )
```

Definition at line 95 of file one_vs_one_svm.cpp.

```
95                                                                                  {
96     this->classifiers.push_back(linear_classifier);
97 }
```

Here is the caller graph for this function:



4.9.4 Friends And Related Function Documentation

4.9.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const OneVsOneSVM & one_vs_one_svm )  [friend]
```

Definition at line 84 of file one_vs_one_svm.cpp.

```
84                                                                                  {
85     // A for_each is not used here because the size of the final string can be really long
86     os « "One Vs One Linear SVM classifiers:" « std::endl;
87     std::size_t i = 0;
88     for (const LinearClassifier &c: one_vs_one_svm.get_classifiers()) {
89         os « "\t[" « i « "] " « c « std::endl;
90         i++;
91     }
92     return os;
93 }
```

4.9.5 Member Data Documentation

4.9.5.1 classifiers

```
std::vector<LinearClassifier> OneVsOneSVM::classifiers  [private]
```

Definition at line 62 of file one_vs_one_svm.h.

4.9.5.2 number_of_class

```
std::size_t OneVsOneSVM::number_of_class  [private]
```

Definition at line 63 of file one_vs_one_svm.h.

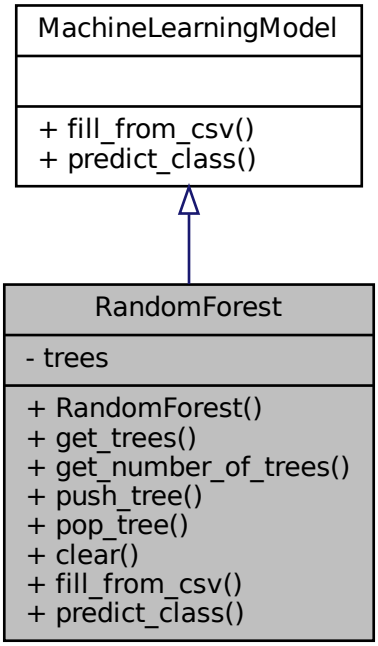
The documentation for this class was generated from the following files:

- ml_algorithms/one_vs_one_svm.h
- ml_algorithms/one_vs_one_svm.cpp

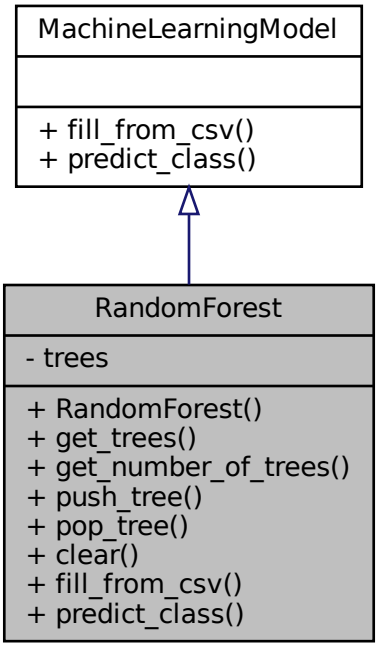
4.10 RandomForest Class Reference

```
#include <random_forest.h>
```

Inheritance diagram for RandomForest:



Collaboration diagram for RandomForest:



Public Member Functions

- RandomForest ()
- const std::vector< DecisionTree > & get_trees () const
- size_t get_number_of_trees () const
- void push_tree (const DecisionTree &tree)
- void pop_tree ()
- void clear ()
- void fill_from_csv (const std::filesystem::path &csv_folder_path) override
- std::string predict_class (const real_vector_t &features_vector) override

Private Attributes

- std::vector< DecisionTree > trees

Friends

- std::ostream & operator<< (std::ostream &os, const RandomForest &random_forest)

4.10.1 Detailed Description

Definition at line 13 of file random_forest.h.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 RandomForest()

```
RandomForest::RandomForest ( )
```

Definition at line 13 of file random_forest.cpp.

```
13                                {
14     trees = {};
15 }
```

4.10.3 Member Function Documentation

4.10.3.1 clear()

```
void RandomForest::clear ( )
```

Definition at line 40 of file random_forest.cpp.

```
40                               {
41     this->trees.clear();
42 }
```

4.10.3.2 fill_from_csv()

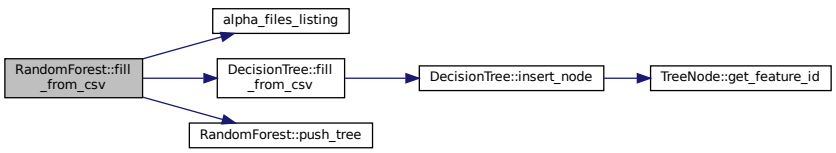
```
void RandomForest::fill_from_csv (
            const std::filesystem::path & csv_folder_path )  [override], [virtual]
```

Implements MachineLearningModel.

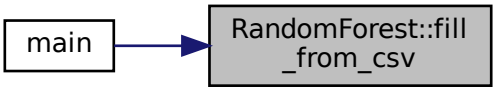
Definition at line 44 of file random_forest.cpp.

```
44                                                                                {
45     auto csv_files = alpha_files_listing(csv_folder_path.string());
46     for (const auto &csv_file_path: csv_files) {
47         DecisionTree new_tree = {};
48         new_tree.fill_from_csv(csv_file_path);
49         this->push_tree(new_tree);
50     }
51 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



4.10.3.3 get_number_of_trees()

```
unsigned long RandomForest::get_number_of_trees ( ) const
```

Definition at line 21 of file random_forest.cpp.

```
21                                                  {
22     return this->trees.size();
23 }
```

Here is the caller graph for this function:

4.10.3.4 get_trees()

```
const std::vector< DecisionTree > & RandomForest::get_trees ( ) const
```

Definition at line 17 of file random_forest.cpp.

```
17                                                      {
18     return trees;
19 }
```

Here is the caller graph for this function:

4.10.3.5 pop_tree()

```
void RandomForest::pop_tree ( )
```

Definition at line 36 of file random_forest.cpp.

```
36                              {
37     this->trees.pop_back();
38 }
```

4.10.3.6 predict_class()

```
std::string RandomForest::predict_class (
            const real_vector_t & features_vector )  [override], [virtual]
```

Implements MachineLearningModel.

Definition at line 53 of file random_forest.cpp.

```
53                                                                                  {
54     std::map<std::string, std::size_t> pred_results;
55     using pred_results_pair_t = decltype(pred_results)::value_type;
56
57     // Get results for each tree
58     std::for_each(std::execution::seq, this->trees.begin(), this->trees.end(), [&pred_results,
       &features_vector](DecisionTree &tree) mutable {
59         std::string pred_result = tree.predict_class(features_vector);
60         if (pred_results.count(pred_result) == 0) {
61             pred_results.insert(std::make_pair(pred_result, 1));
62         } else {
63             pred_results.at(pred_result) += 1;
64         }
65     });
66
67     // Get the prediction result with the most choice
68     auto pr = std::max_element(std::execution::seq, pred_results.cbegin(), pred_results.cend(),
69                                [](const pred_results_pair_t &p1, const pred_results_pair_t &p2) {
70                                    return p1.second < p2.second;
71                                });
72
73     return pr->first;
74 }
```

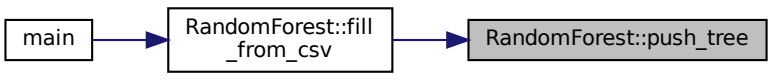
4.10.3.7 push_tree()

```
void RandomForest::push_tree (
            const DecisionTree & tree )
```

Definition at line 32 of file random_forest.cpp.

```
32                                                  {
33     this->trees.push_back(tree);
34 }
```

Here is the caller graph for this function:



4.10.4 Friends And Related Function Documentation

4.10.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const RandomForest & random_forest )  [friend]
```

Definition at line 25 of file random_forest.cpp.

```
25                                                                          {
26     for (const DecisionTree &tree: random_forest.get_trees()) {
27         os « "\n - tree[depth: " « tree.get_depth() « ", number_of_nodes: " « tree.get_number_of_nodes()
       « "]";
28     }
29     return os;
30 }
```

4.10.5 Member Data Documentation

4.10.5.1 trees

```
std::vector<DecisionTree> RandomForest::trees  [private]
```

Definition at line 35 of file random_forest.h.

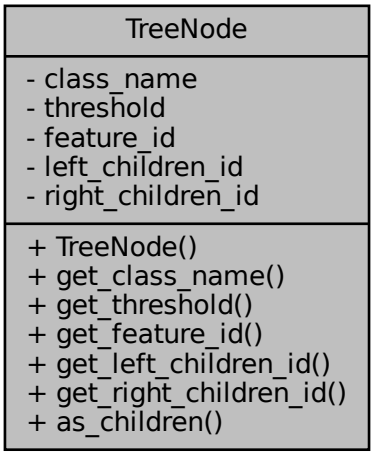
The documentation for this class was generated from the following files:

- ml_algorithms/random_forest.h
- ml_algorithms/random_forest.cpp

4.11 TreeNode Class Reference

```
#include <decision_tree.h>
```

Collaboration diagram for TreeNode:



Public Member Functions

- TreeNode (std::string class_name, real_t threshold, int feature_id, int left_children_id, int right_children_id)
- const std::string & get_class_name () const
- real_t get_threshold () const
- int get_feature_id () const
- int get_left_children_id () const
- int get_right_children_id () const
- bool as_children () const

Private Attributes

- std::string class_name
- real_t threshold
- std::size_t feature_id
- int left_children_id
- int right_children_id

Friends

- std::ostream & operator<< (std::ostream &os, const TreeNode &tree_node)

4.11.1 Detailed Description

Definition at line 13 of file decision_tree.h.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 TreeNode()

```
TreeNode::TreeNode (
            std::string class_name,
            real_t threshold,
            int feature_id,
            int left_children_id,
            int right_children_id )
```

Definition at line 13 of file decision_tree.cpp.

```
13
                :
14         class_name(std::move(class_name)), threshold(threshold), feature_id(feature_id),
       left_children_id(left_children_id), right_children_id(right_children_id) {
15 }
```

4.11.3 Member Function Documentation

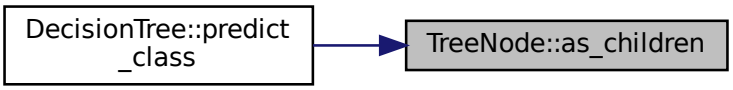
4.11.3.1 as_children()

```
bool TreeNode::as_children ( ) const
```

Definition at line 41 of file decision_tree.cpp.

```
41                                  {
42     if (this->left_children_id != -1 || this->right_children_id != -1) {
43         return true;
44     } else {
45         return false;
46     }
47 }
```

Here is the caller graph for this function:



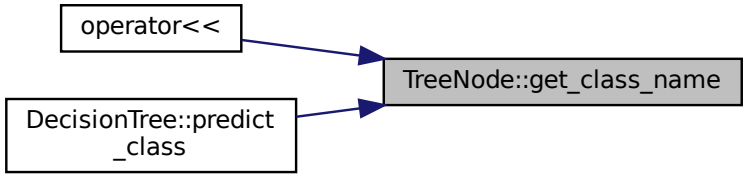
4.11.3.2 get_class_name()

```
const std::string & TreeNode::get_class_name ( ) const
```

Definition at line 17 of file decision_tree.cpp.

```
17                                              {
18     return class_name;
19 }
```

Here is the caller graph for this function:



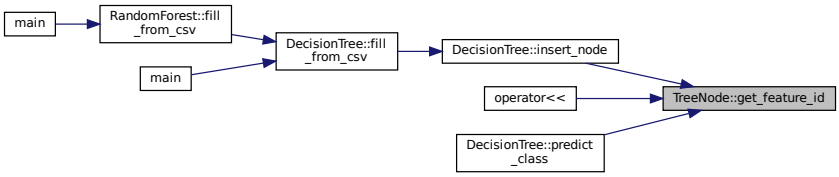
4.11.3.3 get_feature_id()

```
int TreeNode::get_feature_id ( ) const
```

Definition at line 25 of file decision_tree.cpp.

```
25                                             {
26     return feature_id;
27 }
```

Here is the caller graph for this function:



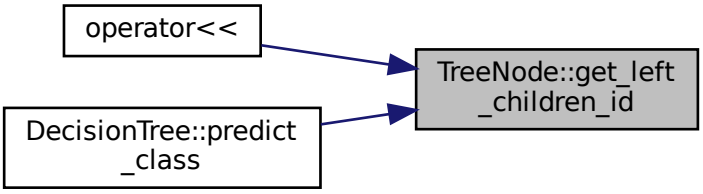
4.11.3.4 get_left_children_id()

```
int TreeNode::get_left_children_id ( ) const
```

Definition at line 29 of file decision_tree.cpp.

```
29                                          {
30     return left_children_id;
31 }
```

Here is the caller graph for this function:



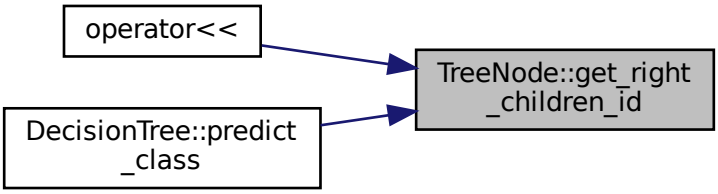
4.11.3.5 get_right_children_id()

```
int TreeNode::get_right_children_id ( ) const
```

Definition at line 33 of file decision_tree.cpp.

```
33                                           {
34     return right_children_id;
35 }
```

Here is the caller graph for this function:



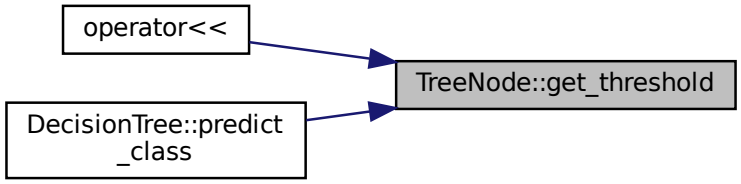
4.11.3.6 get_threshold()

```
real_t TreeNode::get_threshold ( ) const
```

Definition at line 21 of file decision_tree.cpp.

```
21                                      {
22     return threshold;
23 }
```

Here is the caller graph for this function:



4.11.4 Friends And Related Function Documentation

4.11.4.1 operator<<

```
std::ostream& operator<< (
             std::ostream & os,
             const TreeNode & tree_node )  [friend]
```

Definition at line 37 of file decision_tree.cpp.

```
37                                                                  {
38     return os « "(class: " « tree_node.get_class_name() « ", treshold: " « tree_node.get_threshold() « ",
       feature_id: " « tree_node.get_feature_id() « ", left_id: " « tree_node.get_left_children_id() « ",
       right_id: " « tree_node.get_right_children_id() « ")";
39 }
```

4.11.5 Member Data Documentation

4.11.5.1 class_name

```
std::string TreeNode::class_name  [private]
```

Definition at line 32 of file decision_tree.h.

4.11.5.2 feature_id

```
std::size_t TreeNode::feature_id  [private]
```

Definition at line 34 of file decision_tree.h.

4.11.5.3 left_children_id

```
int TreeNode::left_children_id  [private]
```

Definition at line 35 of file decision_tree.h.

4.11.5.4 right_children_id

```
int TreeNode::right_children_id  [private]
```

Definition at line 36 of file decision_tree.h.

4.11.5.5 threshold

```
real_t TreeNode::threshold  [private]
```

Definition at line 33 of file decision_tree.h.

The documentation for this class was generated from the following files:

- ml_algorithms/decision_tree.h
- ml_algorithms/decision_tree.cpp

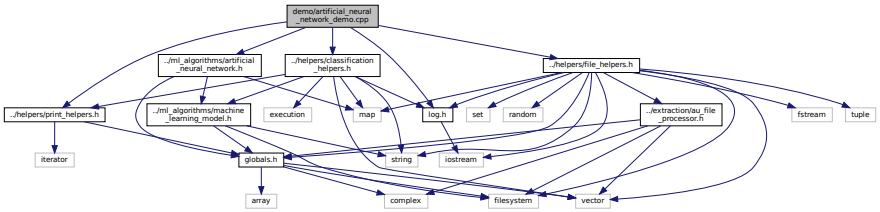
Chapter 5

File Documentation

5.1 demo/artificial_neural_network_demo.cpp File Reference

```
#include "../helpers/file_helpers.h"
#include "../helpers/print_helpers.h"
#include "../helpers/classification_helpers.h"
#include "../helpers/log.h"
#include "../ml_algorithms/artificial_neural_network.h"
```

Include dependency graph for artificial_neural_network_demo.cpp:



Functions

- int main ()

Variables

- log_struct LOGGING_CONFIG = {}

5.1.1 Function Documentation

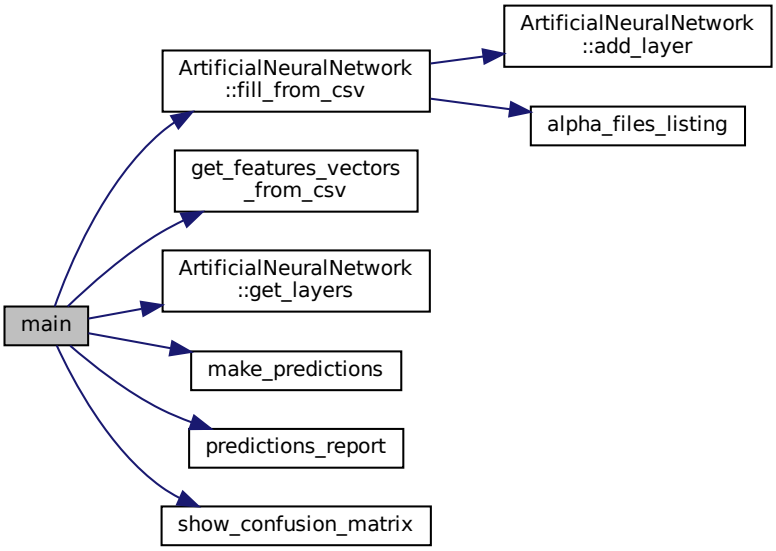
5.1.1.1 main()

```
int main ( )
```

Definition at line 8 of file artificial_neural_network_demo.cpp.

```
8         {
9     // Log config
10     LOGGING_CONFIG.level = LOG_INFO;
11
12     LOG(LOG_INFO) « "------------ Testing Artificial Neural Network using STFT algorithm ------------";
13     // Get features from csv file
14     LOG(LOG_INFO) « "Getting features vectors from the csv file " «
       absolute(MUSIC_FEATURES_STFT_CSV_TEST_PATH) « "...";
15     auto fvs_stft = get_features_vectors_from_csv(MUSIC_FEATURES_STFT_CSV_TEST_PATH,
       AuFileProcessingAlgorithm::STFT);
16     ArtificialNeuralNetwork artificial_neural_network_stft = {};
17     // Create an artificial neural network from csv files
18     LOG(LOG_INFO) « "Creating an Artificial Neural Network from all csv files in the following dir " «
       ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_PATH_STFT « " ...";
19     artificial_neural_network_stft.fill_from_csv(ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_PATH_STFT);
20     LOG(LOG_DEBUG) « "artificial neural network (" « artificial_neural_network_stft.get_layers().size() «
       " layers):\n" « artificial_neural_network_stft;
21
22     // Predict and test prediction results
23     auto predictions_stft = make_predictions(artificial_neural_network_stft, fvs_stft);
24     auto prediction_accuracy_stft = predictions_report(predictions_stft);
25     LOG(LOG_INFO) « "Model accuracy: "« prediction_accuracy_stft;
26     show_confusion_matrix(predictions_stft);
27
28     LOG(LOG_INFO) « "------------ Testing Artificial Neural Network using MFCC algorithm ------------";
29     // Get features from csv file
30     LOG(LOG_INFO) « "Getting features vectors from the csv file " «
       absolute(MUSIC_FEATURES_MFCC_CSV_TEST_PATH) « " ...";
31     auto fvs_mfcc = get_features_vectors_from_csv(MUSIC_FEATURES_MFCC_CSV_TEST_PATH,
       AuFileProcessingAlgorithm::MFCC);
32     ArtificialNeuralNetwork artificial_neural_network_mfcc = {};
33     // Create a tree from csv file
34     LOG(LOG_INFO) « "Creating an Artificial Neural Network from all csv files in the following dir " «
       ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_PATH_MFCC « " ...";
35     artificial_neural_network_mfcc.fill_from_csv(ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_PATH_MFCC);
36     LOG(LOG_DEBUG) « "artificial neural network (" « artificial_neural_network_mfcc.get_layers().size() «
       " layers):\n" « artificial_neural_network_mfcc;
37
38     // Predict and test prediction results
39     auto predictions_mfcc = make_predictions(artificial_neural_network_mfcc, fvs_mfcc);
40     auto prediction_accuracy_mfcc = predictions_report(predictions_mfcc);
41     LOG(LOG_INFO) « "Model accuracy: "« prediction_accuracy_mfcc;
42     show_confusion_matrix(predictions_mfcc);
43
44     return 0;
45 }
```

Here is the call graph for this function:



5.1.2 Variable Documentation

5.1.2.1 LOGGING_CONFIG

```
log_struct LOGGING_CONFIG = {}
```

Definition at line 6 of file artificial_neural_network_demo.cpp.

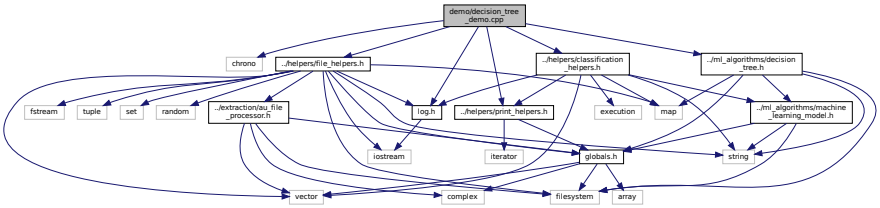
5.2 demo/CMakeLists.txt File Reference

5.3 demo/decision_tree_demo.cpp File Reference

```
#include <chrono>
#include "../helpers/file_helpers.h"
#include "../helpers/print_helpers.h"
#include "../helpers/log.h"
#include "../helpers/classification_helpers.h"
```

```
#include "../ml_algorithms/decision_tree.h"
```

Include dependency graph for decision_tree_demo.cpp:



Functions

- int main ()

Variables

- log_struct LOGGING_CONFIG = {}

5.3.1 Function Documentation

5.3.1.1 main()

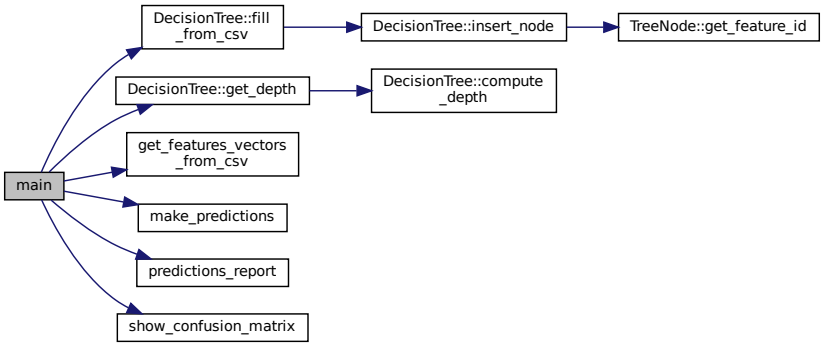
```
int main ( )
```

Definition at line 10 of file decision_tree_demo.cpp.

```
10          {
11      // Log config
12      LOGGING_CONFIG.level = LOG_INFO;
13
14      LOG(LOG_INFO) « "------------ Testing Decision Tree using STFT algorithm ------------";
15      // Get features from csv file
16      LOG(LOG_INFO) « "Getting features vectors from the csv file " «
        absolute(MUSIC_FEATURES_STFT_CSV_TEST_PATH) « "...";
17      auto fvs_stft = get_features_vectors_from_csv(MUSIC_FEATURES_STFT_CSV_TEST_PATH,
        AuFileProcessingAlgorithm::STFT);
18      DecisionTree decision_tree_model_stft = {};
19      // Create a tree from csv file
20      LOG(LOG_INFO) « "Creating a Decision Tree from the csv file " « DECISION_TREE_CSV_PATH_STFT « " ...";
21      decision_tree_model_stft.fill_from_csv(DECISION_TREE_CSV_PATH_STFT);
22      LOG(LOG_DEBUG) « "decision tree (depth: " « decision_tree_model_stft.get_depth() « "): " «
        decision_tree_model_stft;
23
24      // Predict and test prediction results
25      auto predictions_stft = make_predictions(decision_tree_model_stft, fvs_stft);
26      auto prediction_accuracy_stft = predictions_report(predictions_stft);
27      LOG(LOG_INFO) « "Model accuracy: "« prediction_accuracy_stft;
28      show_confusion_matrix(predictions_stft);
29
30
31      LOG(LOG_INFO) « "------------ Testing Decision Tree using MFCC algorithm ------------";
32      // Get features from csv file
33      LOG(LOG_INFO) « "Getting features vectors from the csv file " «
        absolute(MUSIC_FEATURES_MFCC_CSV_TEST_PATH) « " ...";
34      auto fvs_mfcc = get_features_vectors_from_csv(MUSIC_FEATURES_MFCC_CSV_TEST_PATH,
        AuFileProcessingAlgorithm::MFCC);
35      DecisionTree decision_tree_model_mfcc = {};
36      // Create a tree from csv file
37      LOG(LOG_INFO) « "Creating a Decision Tree from the csv file " « DECISION_TREE_CSV_PATH_MFCC « " ...";
```

```
38     decision_tree_model_mfcc.fill_from_csv(DECISION_TREE_CSV_PATH_MFCC);
39     LOG(LOG_DEBUG) « "decision tree (depth: " « decision_tree_model_mfcc.get_depth() « "): " «
       decision_tree_model_mfcc;
40
41     // Predict and test prediction results
42     auto predictions_mfcc = make_predictions(decision_tree_model_mfcc, fvs_mfcc);
43     auto prediction_accuracy_mfcc = predictions_report(predictions_mfcc);
44     LOG(LOG_INFO) « "Model accuracy: "« prediction_accuracy_mfcc;
45     show_confusion_matrix(predictions_mfcc);
46
47     return 0;
48 }
```

Here is the call graph for this function:



5.3.2 Variable Documentation

5.3.2.1 LOGGING_CONFIG

```
log_struct LOGGING_CONFIG = {}
```

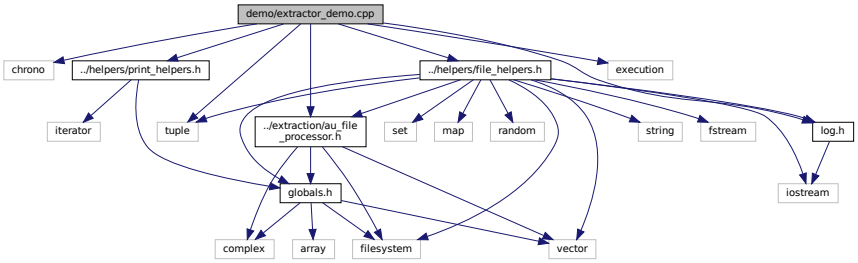
Definition at line 8 of file decision_tree_demo.cpp.

5.4 demo/extractor_demo.cpp File Reference

```
#include <chrono>
#include <tuple>
#include <execution>
#include "../helpers/file_helpers.h"
#include "../helpers/print_helpers.h"
#include "../helpers/log.h"
```

```
#include "../extraction/au_file_processor.h"
```

Include dependency graph for extractor_demo.cpp:



Enumerations

- enum DatasetType : std::size_t { DatasetType::TRAIN = 0, DatasetType::TEST = 1 }

  extractor types of dataset

Functions

- void extract_files (const std::vector< std::filesystem::path > &file_list, AuFileProcessingAlgorithm processing_algorithm, DatasetType dataset_type)

  Extract all files from the given file list using a specified processing algorithm and dataset.
- int main ()

Variables

- log_struct LOGGING_CONFIG = {}

5.4.1 Enumeration Type Documentation

5.4.1.1 DatasetType

```
enum DatasetType :  std::size_t  [strong]
```

extractor types of dataset

Enumerator

TRAIN	
TEST	The train dataset

Definition at line 11 of file extractor_demo.cpp.

```
11                          : std::size_t {
12     TRAIN = 0,
13     TEST = 1
14 };
```

5.4.2 Function Documentation

5.4.2.1 extract_files()

```
void extract_files (
            const std::vector< std::filesystem::path > & file_list,
            AuFileProcessingAlgorithm processing_algorithm,
            DatasetType dataset_type )
```

Extract all files from the given file list using a specified processing algorithm and dataset.

Parameters

in	*file_list*	a std::vector<std::filesystem::path> vector of all files to process
in	*processing_algorithm*	a AuFileProcessingAlgorithm enum object designating the processing algorithm to use
in	*dataset_type*	a DatasetType enum object designating the dataset to use

Returns

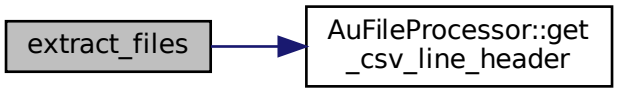
void

Definition at line 24 of file extractor_demo.cpp.

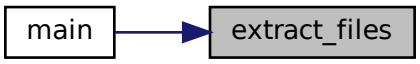
```
24                                          {
25     std::filesystem::path csv_file_path;
26     std::string processing_algorithm_string, dataset_type_string;
27 
28     // Parse parameters
29     switch (dataset_type) {
30         case DatasetType::TRAIN: {
31             dataset_type_string = "TRAIN";
32             switch (processing_algorithm) {
33                 case AuFileProcessingAlgorithm::STFT: {
34                     csv_file_path = MUSIC_FEATURES_STFT_CSV_TRAIN_PATH;
35                     processing_algorithm_string = "STFT";
36                     break;
37                 }
38                 case AuFileProcessingAlgorithm::MFCC: {
39                     csv_file_path = MUSIC_FEATURES_MFCC_CSV_TRAIN_PATH;
40                     processing_algorithm_string = "MFCC";
41                     break;
42                 }
43                 default: {
44                     LOG(LOG_ERROR) « "Error : the processing algorithm value usage is not defined in the
     project";
45                     throw std::domain_error("Unsupported processing algorithm!");
46                     break;
47                 }
48             }
49             break;
50         }
51         case DatasetType::TEST: {
52             dataset_type_string = "TEST";
53             switch (processing_algorithm) {
54                 case AuFileProcessingAlgorithm::STFT: {
55                     csv_file_path = MUSIC_FEATURES_STFT_CSV_TEST_PATH;
```

```
56                      processing_algorithm_string = "STFT";
57                      break;
58                  }
59                  case AuFileProcessingAlgorithm::MFCC: {
60                      csv_file_path = MUSIC_FEATURES_MFCC_CSV_TEST_PATH;
61                      processing_algorithm_string = "MFCC";
62                      break;
63                  }
64                  default: {
65                      LOG(LOG_ERROR) << "Error : the processing algorithm value usage is not defined in the
      project";
66                      throw std::domain_error("Unsupported processing algorithm!");
67                      break;
68                  }
69              }
70              break;
71          }
72          default: {
73              LOG(LOG_ERROR) << "Error : the dataset type value usage is not defined in the project";
74              throw std::domain_error("Unsupported dataset type!");
75              break;
76          }
77      }
78
79      // Open the csv file and write the header for the dataset
80      std::ofstream csv_file(csv_file_path);
81      csv_file << AuFileProcessor::get_csv_line_header(processing_algorithm);
82
83      // Processing all files and writing processed features into the csv file
84      LOG(LOG_INFO) << "------------ Processing files from the " << dataset_type_string << " dataset using the
      " << processing_algorithm_string << " algorithm ------------";
85      auto start_time = std::chrono::high_resolution_clock::now();
86
87      std::for_each(std::execution::seq, file_list.cbegin(), file_list.cend(), [&processing_algorithm,
      &csv_file](const std::filesystem::path& file) {
88          try {
89              LOG(LOG_DEBUG) << "Processing file " << file.filename().string() << "...";
90              auto started_chrono = std::chrono::high_resolution_clock::now();
91              AuFileProcessor au_file(file, processing_algorithm);
92              au_file.read_file();
93              LOG(LOG_DEBUG) << au_file.get_file_path().filename().string() << " details: " << au_file;
94              auto stopped_chrono = std::chrono::high_resolution_clock::now();
95              auto elapsed_time = std::chrono::duration_cast<std::chrono::milliseconds>(stopped_chrono -
      started_chrono).count();
96              LOG(LOG_DEBUG) << "File read in " << elapsed_time / 1000 << "s and " << elapsed_time % 1000 <<
      "ms";
97              started_chrono = std::chrono::high_resolution_clock::now();
98              LOG(LOG_DEBUG) << "Applying processing algorithm on file " << file.filename().string() << "
      data...";
99              au_file.apply_processing_algorithm();
100              LOG(LOG_DEBUG) << au_file.get_file_path().filename().string() << " avg[" <<
      au_file.get_features_average().size() << "]: " << au_file.get_features_average();
101              LOG(LOG_DEBUG) << au_file.get_file_path().filename().string() << " std[" <<
      au_file.get_features_standard_deviation().size() << "]: " << au_file.get_features_standard_deviation();
102              stopped_chrono = std::chrono::high_resolution_clock::now();
103              elapsed_time = std::chrono::duration_cast<std::chrono::milliseconds>(stopped_chrono -
      started_chrono).count();
104              LOG(LOG_DEBUG) << "File processed in " << elapsed_time / 1000 << "s and " << elapsed_time % 1000
      << "ms";
105              LOG(LOG_DEBUG) << "Adding file " << file.filename().string() << " features to csv file...";
106              csv_file << au_file.get_csv_line() + "\n";
107          } catch (const std::exception &e) {
108              LOG(LOG_ERROR) << "File " << file.filename().string() << " not processed due to the following
      error : " << e.what();
109          }
110      });
111      csv_file.close();
112      LOG(LOG_INFO) << "Files features vector, music style and path writen in the CSV file " <<
      absolute(csv_file_path);
113      auto stop_time = std::chrono::high_resolution_clock::now();
114      auto elapsed_time = std::chrono::duration_cast<std::chrono::milliseconds>(stop_time -
      start_time).count();
115      LOG(LOG_INFO) << "------------ " << file_list.size() << " music files read in " << elapsed_time / 1000 <<
      "s and " << elapsed_time % 1000 << "ms ------------";
116 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



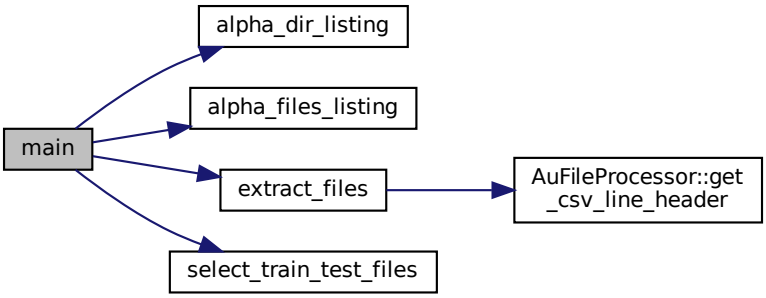
5.4.2.2 main()

```
int main ( )
```

Definition at line 120 of file extractor_demo.cpp.

```
120           {
121     // log config
122     LOGGING_CONFIG.level = LOG_INFO;
123 
124     auto dirs = alpha_dir_listing("../../../datasets/music");
125     std::vector<std::filesystem::path> training_files;
126     std::vector<std::filesystem::path> testing_files;
127 
128     // Select random files of each music style
129     for (const auto &dir_path: dirs) {
130         auto files = alpha_files_listing(dir_path);
131         std::vector<std::filesystem::path> training;
132         std::vector<std::filesystem::path> testing;
133         std::tie(training, testing) = select_train_test_files(files, 0.2);
134         training_files.insert(training_files.end(), training.begin(), training.end());
135         testing_files.insert(testing_files.end(), testing.begin(), testing.end());
136     }
137 
138     LOG(LOG_INFO) << "------------ Found " << training_files.size() << " training files ------------";
139     LOG(LOG_INFO) << "------------ Found " << testing_files.size() << " testing files ------------";
140 
141     // Extracting training files using STFT
142     extract_files(training_files, AuFileProcessingAlgorithm::STFT, DatasetType::TRAIN);
143     // Extracting testing files using STFT
144     extract_files(testing_files, AuFileProcessingAlgorithm::STFT, DatasetType::TEST);
145     // Extracting training files using MFCC
146     extract_files(training_files, AuFileProcessingAlgorithm::MFCC, DatasetType::TRAIN);
147     // Extracting testing files using MFCC
148     extract_files(testing_files, AuFileProcessingAlgorithm::MFCC, DatasetType::TEST);
149 
150     return 0;
151 }
```

Here is the call graph for this function:



5.4.3 Variable Documentation

5.4.3.1 LOGGING_CONFIG

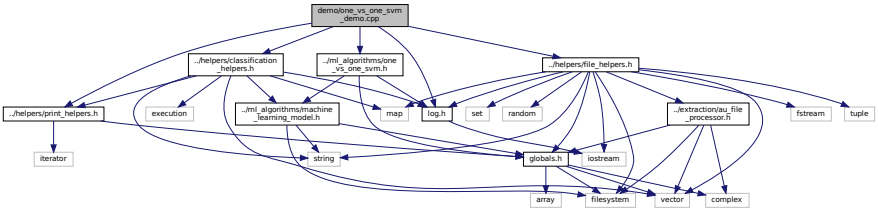
```
log_struct LOGGING_CONFIG = {}
```

Definition at line 118 of file extractor_demo.cpp.

5.5 demo/one_vs_one_svm_demo.cpp File Reference

```
#include "../helpers/file_helpers.h"
#include "../helpers/print_helpers.h"
#include "../helpers/classification_helpers.h"
#include "../helpers/log.h"
#include "../ml_algorithms/one_vs_one_svm.h"
```

Include dependency graph for one_vs_one_svm_demo.cpp:



Functions

- int main ()

Variables

- log_struct LOGGING_CONFIG = {}

5.5.1 Function Documentation

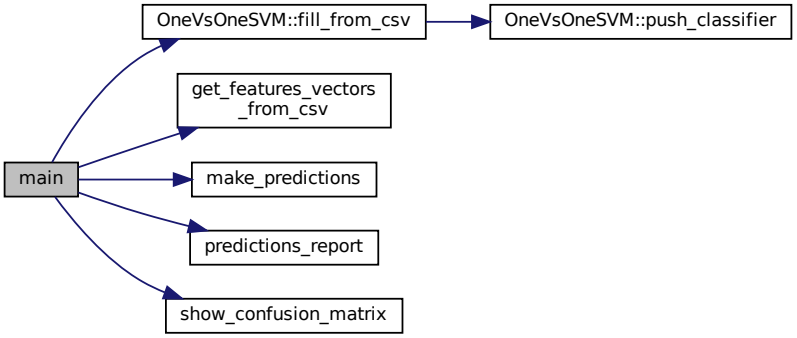
5.5.1.1 main()

```
int main ( )
```

Definition at line 9 of file one_vs_one_svm_demo.cpp.

```
9           {
10      // Log config
11      LOGGING_CONFIG.level = LOG_INFO;
12
13      LOG(LOG_INFO) « "------------ Testing one vs one SVM model using STFT algorithm ------------";
14      // Get features from csv file
15      LOG(LOG_INFO) « "Getting features vectors from the csv file " «
        absolute(MUSIC_FEATURES_STFT_CSV_TEST_PATH) « "...";
16      auto fvs_stft = get_features_vectors_from_csv(MUSIC_FEATURES_STFT_CSV_TEST_PATH,
        AuFileProcessingAlgorithm::STFT);
17      OneVsOneSVM svm_model_stft = {};
18      // Create a one vs one SVM model from csv file
19      LOG(LOG_INFO) « "Creating a one vs one SVM model from the csv file " « ONE_VS_ONE_SVM_CSV_PATH_STFT «
        " ...";
20      svm_model_stft.fill_from_csv(ONE_VS_ONE_SVM_CSV_PATH_STFT);
21      LOG(LOG_DEBUG) « "one vs one SVM: " « svm_model_stft;
22
23      // Predict and test prediction results
24      auto predictions_stft = make_predictions(svm_model_stft, fvs_stft);
25      auto prediction_accuracy_stft = predictions_report(predictions_stft);
26      LOG(LOG_INFO) « "Model accuracy: "« prediction_accuracy_stft;
27      show_confusion_matrix(predictions_stft);
28
29
30      LOG(LOG_INFO) « "------------ Testing one vs one SVM model using MFCC algorithm ------------";
31      // Get features from csv file
32      LOG(LOG_INFO) « "Getting features vectors from the csv file " «
        absolute(MUSIC_FEATURES_MFCC_CSV_TEST_PATH) « " ...";
33      auto fvs_mfcc = get_features_vectors_from_csv(MUSIC_FEATURES_MFCC_CSV_TEST_PATH,
        AuFileProcessingAlgorithm::MFCC);
34      OneVsOneSVM svm_model_mfcc= {};
35      // Create a one vs one SVM model from csv file
36      LOG(LOG_INFO) « "Creating a one vs one SVM model from the csv file " « ONE_VS_ONE_SVM_CSV_PATH_MFCC «
        " ...";
37      svm_model_mfcc.fill_from_csv(ONE_VS_ONE_SVM_CSV_PATH_MFCC);
38      LOG(LOG_DEBUG) « "one vs one SVM: " « svm_model_mfcc;
39
40      // Predict and test prediction results
41      auto predictions_mfcc = make_predictions(svm_model_mfcc, fvs_mfcc);
42      auto prediction_accuracy_mfcc = predictions_report(predictions_mfcc);
43      LOG(LOG_INFO) « "Model accuracy: "« prediction_accuracy_mfcc;
44      show_confusion_matrix(predictions_mfcc);
45
46      return 0;
47  }
```

Here is the call graph for this function:



5.5.2 Variable Documentation

5.5.2.1 LOGGING_CONFIG

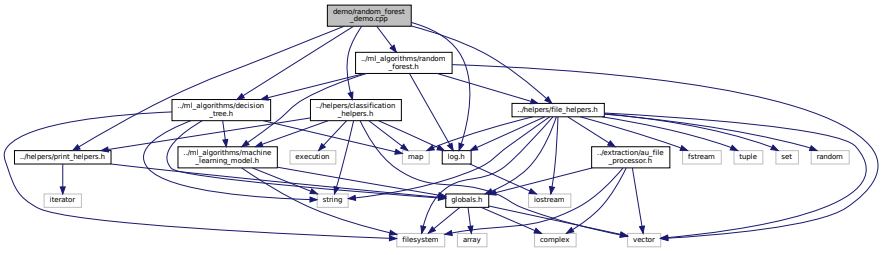
```
log_struct LOGGING_CONFIG = {}
```

Definition at line 7 of file one_vs_one_svm_demo.cpp.

5.6 demo/random_forest_demo.cpp File Reference

```
#include "../helpers/file_helpers.h"
#include "../helpers/print_helpers.h"
#include "../helpers/classification_helpers.h"
#include "../helpers/log.h"
#include "../ml_algorithms/decision_tree.h"
#include "../ml_algorithms/random_forest.h"
```

Include dependency graph for random_forest_demo.cpp:



Functions

- int main ()

Variables

- log_struct LOGGING_CONFIG = {}

5.6.1 Function Documentation

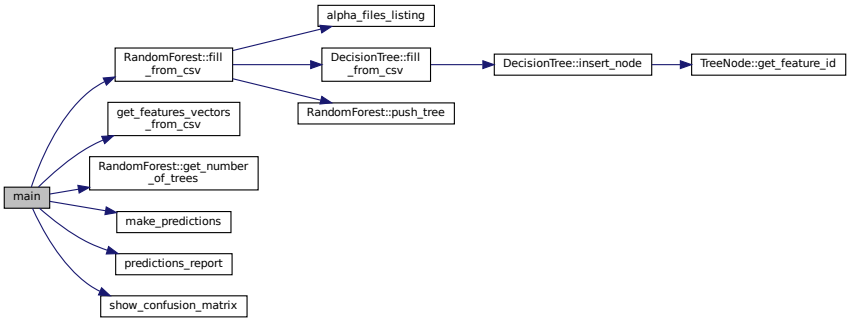
5.6.1.1 main()

```
int main ( )
```

Definition at line 9 of file random_forest_demo.cpp.

```
9           {
10      // Log config
11      LOGGING_CONFIG.level = LOG_INFO;
12
13      LOG(LOG_INFO) « "------------ Testing Random Forest using STFT algorithm ------------";
14      // Get features from csv file
15      LOG(LOG_INFO) « "Getting features vectors from the csv file " «
        absolute(MUSIC_FEATURES_STFT_CSV_TEST_PATH) « "...";
16      auto fvs_stft = get_features_vectors_from_csv(MUSIC_FEATURES_STFT_CSV_TEST_PATH,
        AuFileProcessingAlgorithm::STFT);
17      RandomForest random_forest_model_stft = {};
18      // Create a random forest from csv files
19      LOG(LOG_INFO) « "Creating a Random Forest from all csv files in the following dir " «
        RANDOM_FOREST_TREES_FOLDER_PATH_STFT « " ...";
20      random_forest_model_stft.fill_from_csv(RANDOM_FOREST_TREES_FOLDER_PATH_STFT);
21      LOG(LOG_DEBUG) « "random forest (" « random_forest_model_stft.get_number_of_trees() « " trees): " «
        random_forest_model_stft;
22
23      // Predict and test prediction results
24      auto predictions_stft = make_predictions(random_forest_model_stft, fvs_stft);
25      auto prediction_accuracy_stft = predictions_report(predictions_stft);
26      LOG(LOG_INFO) « "Model accuracy: "« prediction_accuracy_stft;
27      show_confusion_matrix(predictions_stft);
28
29
30      LOG(LOG_INFO) « "------------ Testing Decision Tree using MFCC algorithm ------------";
31      // Get features from csv file
32      LOG(LOG_INFO) « "Getting features vectors from the csv file " «
        absolute(MUSIC_FEATURES_MFCC_CSV_TEST_PATH) « " ...";
33      auto fvs_mfcc = get_features_vectors_from_csv(MUSIC_FEATURES_MFCC_CSV_TEST_PATH,
        AuFileProcessingAlgorithm::MFCC);
34      RandomForest random_forest_model_mfcc = {};
35      // Create a tree from csv file
36      LOG(LOG_INFO) « "Creating a Random Forest from all csv files in the following dir " «
        RANDOM_FOREST_TREES_FOLDER_PATH_MFCC « " ...";
37      random_forest_model_mfcc.fill_from_csv(RANDOM_FOREST_TREES_FOLDER_PATH_MFCC);
38      LOG(LOG_DEBUG) « "random forest (" « random_forest_model_mfcc.get_number_of_trees() « " trees): " «
        random_forest_model_mfcc;
39
40      // Predict and test prediction results
41      auto predictions_mfcc = make_predictions(random_forest_model_mfcc, fvs_mfcc);
42      auto prediction_accuracy_mfcc = predictions_report(predictions_mfcc);
43      LOG(LOG_INFO) « "Model accuracy: "« prediction_accuracy_mfcc;
44      show_confusion_matrix(predictions_mfcc);
45
46      return 0;
47  }
```

Here is the call graph for this function:



5.6.2 Variable Documentation

5.6.2.1 LOGGING_CONFIG

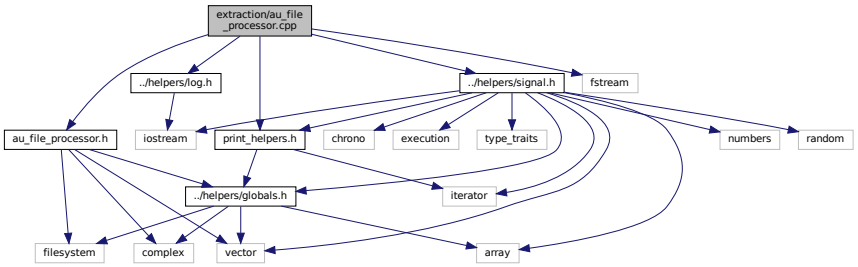
```
log_struct LOGGING_CONFIG = {}
```

Definition at line 7 of file random_forest_demo.cpp.

5.7 extraction/au_file_processor.cpp File Reference

```
#include "au_file_processor.h"
#include "../helpers/log.h"
#include "../helpers/signal.h"
#include "../helpers/print_helpers.h"
#include <fstream>
```

Include dependency graph for au_file_processor.cpp:



Functions

- std::ostream & operator<< (std::ostream &os, const AuFileProcessor &au_file_processor)

5.7.1 Function Documentation

5.7.1.1 operator<<()

```
std::ostream& operator<< (
            std::ostream & os,
            const AuFileProcessor & au_file_processor )
```

Returns

the ostream of the human readable object

Definition at line 107 of file au_file_processor.cpp.

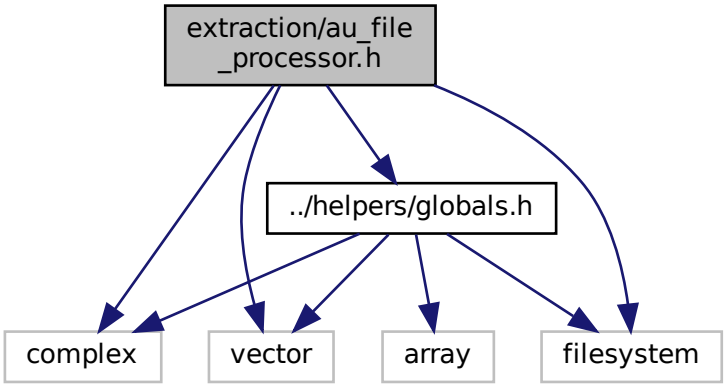
```
107                                                                                                 {
108     char magic_str[5] = {
109             (char) ((au_file_processor.magic_number & 0xFF000000) » 24u),
110             (char) ((au_file_processor.magic_number & 0x00FF0000) » 16u),
111             (char) ((au_file_processor.magic_number & 0x0000FF00) » 8u),
112             (char) ((au_file_processor.magic_number & 0x000000FF) » 0u),
113             '\0'
114     };
115     std::string data_size_human_readable = {};
116     std::string size_unit;
117     if (au_file_processor.data_size < KiB) {
118         data_size_human_readable = std::to_string(au_file_processor.data_size) + "B";
119     } else if (au_file_processor.data_size < MiB) {
120         data_size_human_readable = std::to_string(au_file_processor.data_size / KiB) + "." +
      std::to_string(au_file_processor.data_size % KiB) + "KiB";
121     } else if (au_file_processor.data_size < GiB) {
122         data_size_human_readable = std::to_string(au_file_processor.data_size / MiB) + "." +
      std::to_string(au_file_processor.data_size % MiB) + "MiB";
123     } else {
124         data_size_human_readable = std::to_string(au_file_processor.data_size / GiB) + "." +
      std::to_string(au_file_processor.data_size % GiB) + "GiB";
125     }
126
127     return os « std::hex « "{magic number: 0x" « au_file_processor.magic_number « " (" « magic_str « ")"
128              « std::dec « ", data offset: " « au_file_processor.data_offset
129              « std::dec « ", data size: " « data_size_human_readable
130              « std::dec « ", encoding: " « au_file_processor.encoding
131              « std::dec « ", sample rate (sample/sec): " « au_file_processor.sample_rate
132              « std::dec « ", channels: " « au_file_processor.channels
133              « "}";
134 }
```

5.8 extraction/au_file_processor.h File Reference

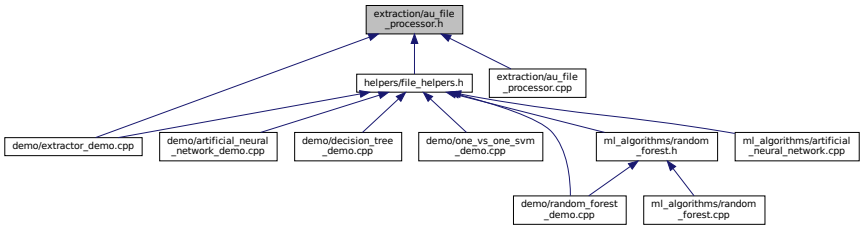
```
#include "../helpers/globals.h"
#include <filesystem>
#include <vector>
```

```
#include <complex>
```

Include dependency graph for au_file_processor.h:



This graph shows which files directly or indirectly include this file:



Classes

- class AuFileProcessor

Enumerations

- enum AuFileEncodingFormat : word_t {
  AuFileEncodingFormat::MICRO_8B = 1, AuFileEncodingFormat::PCM_8B = 2, AuFileEncodingFormat::PCM_16B
  = 3, AuFileEncodingFormat::PCM_24B = 4,
  AuFileEncodingFormat::PCM_32B = 5, AuFileEncodingFormat::IEEE_32B = 6, AuFileEncodingFormat::IEEE_64B
  = 7, AuFileEncodingFormat::FRAGMENTED = 8,
  AuFileEncodingFormat::DSP_PROGRAM = 9, AuFileEncodingFormat::FIXED_8B = 10, AuFileEncodingFormat::FIXED_16B
  = 11, AuFileEncodingFormat::FIXED_24B = 12,
  AuFileEncodingFormat::FIXED_32B = 13, AuFileEncodingFormat::LINEAR_EMPHASIS_16B = 18,
  AuFileEncodingFormat::LINEAR_COMPRESSES_16B = 19, AuFileEncodingFormat::LINEAR_EMPHASIS_COMPRESSED_1
  = 20,
  AuFileEncodingFormat::MUSIC_KIT_DSP_PROGRAM = 21, AuFileEncodingFormat::ITUT_G721_ADPCM_4B
  = 23, AuFileEncodingFormat::ITUT_G722_SB_ADPCM_4B = 24, AuFileEncodingFormat::ITUT_G723_ADPCM_3B
  = 25,
  AuFileEncodingFormat::ITUT_G723_ADPCM_5B = 26, AuFileEncodingFormat::ALAX_G711_8B = 27 }

.au file encoding formats

- enum AuFileProcessingAlgorithm : std::size_t { AuFileProcessingAlgorithm::STFT = 0, AuFileProcessingAlgorithm::MFCC = 1 }

  .au file process methods

5.8.1 Enumeration Type Documentation

5.8.1.1 AuFileEncodingFormat

```
enum AuFileEncodingFormat : word_t [strong]
```

.au file encoding formats

Enumerator

MICRO_8B	
PCM_8B	8-bit G.711 \GenericError LaTeX Error: Can be used only in preambleSee the LaTeX manual or LaTeX Companion for explanation.Your command was ignored.˙(inputenc) Type I <command> <return> to replace it with another command,˙(inputenc) or <return> to continue without it.03BCμ-law
PCM_16B	8-bit linear PCM
PCM_24B	16-bit linear PCM
PCM_32B	24-bit linear PCM
IEEE_32B	32-bit linear PCM
IEEE_64B	32-bit IEEE floating point
FRAGMENTED	64-bit IEEE floating point
DSP_PROGRAM	Fragmented sample data
FIXED_8B	DSP program
FIXED_16B	8-bit fixed point
FIXED_24B	16-bit fixed point
FIXED_32B	24-bit fixed point
LINEAR_EMPHASIS_16B	32-bit fixed point
LINEAR_COMPRESSES_16B	16-bit linear with emphasis
LINEAR_EMPHASIS_COMPRESSED_16B	16-bit linear compressed
MUSIC_KIT_DSP_PROGRAM	16-bit linear with emphasis and compression
ITUT_G721_ADPCM_4B	Music kit DSP commands
ITUT_G722_SB_ADPCM_4B	4-bit compressed using the ITU-T G.721 ADPCM voice data encoding scheme
ITUT_G723_ADPCM_3B	ITU-T G.722 SB-ADPCM
ITUT_G723_ADPCM_5B	ITU-T G.723 3-bit ADPCM
ALAX_G711_8B	ITU-T G.723 5-bit ADPCM

Definition at line 10 of file au_file_processor.h.

```
10                                  : word_t {
11     MICRO_8B = 1,
12     PCM_8B = 2,
```

```
13     PCM_16B = 3,
14     PCM_24B = 4,
15     PCM_32B = 5,
16     IEEE_32B = 6,
17     IEEE_64B = 7,
18     FRAGMENTED = 8,
19     DSP_PROGRAM = 9,
20     FIXED_8B = 10,
21     FIXED_16B = 11,
22     FIXED_24B = 12,
23     FIXED_32B = 13,
24     LINEAR_EMPHASIS_16B = 18,
25     LINEAR_COMPRESSES_16B = 19,
26     LINEAR_EMPHASIS_COMPRESSED_16B = 20,
27     MUSIC_KIT_DSP_PROGRAM = 21,
28     ITUT_G721_ADPCM_4B = 23,
29     ITUT_G722_SB_ADPCM_4B = 24,
30     ITUT_G723_ADPCM_3B = 25,
31     ITUT_G723_ADPCM_5B = 26,
32     ALAX_G711_8B = 27
33 };
```

5.8.1.2 AuFileProcessingAlgorithm

```
enum AuFileProcessingAlgorithm :  std::size_t  [strong]
```

.au file process methods

Enumerator

STFT	
MFCC	Process the data using only the STFT algorithm

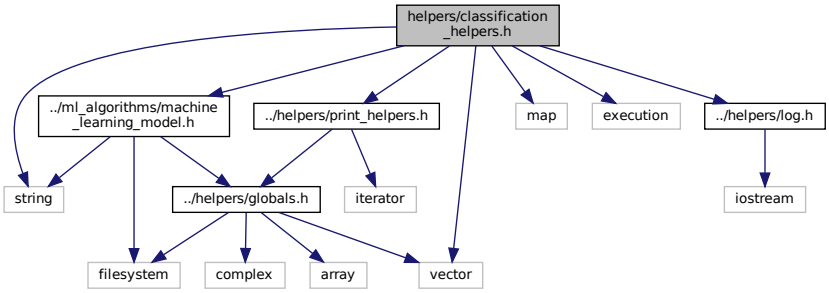
Definition at line 36 of file au_file_processor.h.

```
36                                      : std::size_t {
37     STFT = 0,
38     MFCC = 1
39 };
```

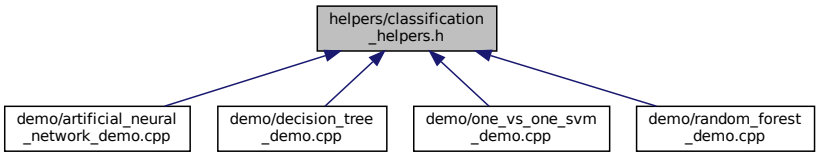
5.9 helpers/classification_helpers.h File Reference

```
#include <string>
#include <vector>
#include <map>
#include <execution>
#include "../ml_algorithms/machine_learning_model.h"
#include "../helpers/log.h"
#include "../helpers/print_helpers.h"
```

Include dependency graph for classification_helpers.h:



This graph shows which files directly or indirectly include this file:



Functions

- static real_t predictions_report (const std::vector< std::pair< std::string, std::string >> &predictions)
- static void show_confusion_matrix (const std::vector< std::pair< std::string, std::string >> &predictions)
- static std::vector< std::pair< std::string, std::string > > make_predictions (MachineLearningModel &model, const std::vector< std::pair< std::string, real_vector_t >> &feature_vectors)

5.9.1 Function Documentation

5.9.1.1 make_predictions()

```
static std::vector<std::pair<std::string, std::string> > make_predictions (
            MachineLearningModel & model,
            const std::vector< std::pair< std::string, real_vector_t >> & feature_vectors )
[inline], [static]
```

Definition at line 90 of file classification_helpers.h.

```
90                                                                                {
91     std::vector<std::pair<std::string, std::string» predictions;
92
93     // Predict for all feature vectors
94     auto start_time = std::chrono::high_resolution_clock::now();
```

```
95      LOG(LOG_INFO) « "Making " « feature_vectors.size() « " prediction using the machine learning
        model...";
96      std::for_each(std::execution::seq, feature_vectors.cbegin(), feature_vectors.cend(), [&predictions,
        &model](const std::pair<std::string, real_vector_t> &pair)mutable {
97          std::string predicted_class = model.predict_class(pair.second);
98          LOG(LOG_DEBUG) « "\ttrue class: " « pair.first « ", predicted class: " « predicted_class;
99          predictions.emplace_back(pair.first, predicted_class);
100     });
101     auto stop_time = std::chrono::high_resolution_clock::now();
102     auto elapsed_time = std::chrono::duration_cast<std::chrono::milliseconds>(stop_time -
        start_time).count();
103     LOG(LOG_INFO) « "Prediction done in " « elapsed_time / 1000 « "s and " « elapsed_time % 1000 « "ms";
104
105     return predictions;
106 }
```

Here is the caller graph for this function:

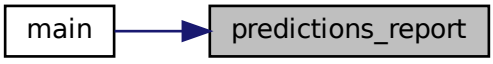
5.9.1.2 predictions_report()

```
static real_t predictions_report (
              const std::vector< std::pair< std::string, std::string >> & predictions )  [inline],
[static]
```

Definition at line 13 of file classification_helpers.h.

```
13                                                                                                  {
14      std::size_t good_predictions = 0;
15      std::for_each(std::execution::seq, predictions.cbegin(), predictions.cend(),
        [&good_predictions](const std::pair<std::string, std::string> &pair)mutable {
16          if (pair.first == pair.second) {
17              good_predictions += 1;
18          }
19      });
20
21      return (real_t) good_predictions / (real_t) predictions.size();
22  }
```

Here is the caller graph for this function:



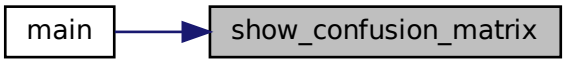
5.9.1.3 show_confusion_matrix()

```
static void show_confusion_matrix (
            const std::vector< std::pair< std::string, std::string >> & predictions )  [inline],
[static]
```

Definition at line 25 of file classification_helpers.h.

```
25                                                                                                                {
26     std::map<std::string, std::map<std::string, int» confusion_matrix;
27     std::size_t longest_unique_label_size = 0;
28     std::set<std::string> unique_labels;
29
30     // Find unique labels
31     std::for_each(std::execution::seq, predictions.cbegin(), predictions.cend(), [&unique_labels,
       &longest_unique_label_size](const std::pair<std::string, std::string> &pair) mutable {
32         if (!unique_labels.contains(pair.first)) {
33             unique_labels.insert(pair.first);
34             longest_unique_label_size = pair.first.size() > longest_unique_label_size ? pair.first.size()
       : longest_unique_label_size;
35         }
36         if (!unique_labels.contains(pair.second)) {
37             unique_labels.insert(pair.second);
38             longest_unique_label_size = pair.second.size() > longest_unique_label_size ?
       pair.second.size() : longest_unique_label_size;
39         }
40     });
41
42     // Generate empty 2D hashmap
43     std::for_each(std::execution::seq, unique_labels.cbegin(), unique_labels.cend(), [&confusion_matrix,
       &unique_labels](const std::string &label) mutable {
44         confusion_matrix.insert(std::make_pair(label, std::map<std::string, int>{}));
45
46         std::transform(unique_labels.cbegin(), unique_labels.cend(),
47                        std::inserter(confusion_matrix.at(label), confusion_matrix.at(label).end()),
48                        [](const std::string &unique_label) { return std::make_pair(unique_label, 0); }
49         );
50     });
51
52     // Fill the 2D hashmap
53     std::for_each(std::execution::seq, predictions.cbegin(), predictions.cend(),
       [&confusion_matrix](const std::pair<std::string, std::string> &pair) mutable {
54         confusion_matrix.at(pair.first).at(pair.second) += 1;
55     });
56
57     std::cout « (std::string(longest_unique_label_size + 5, ' ') + "\t ");
58     for (std::string label: unique_labels) {
59         try {
60             std::cout « (char) toupper(label.at(0));
61             std::cout « label.at(1);
62         } catch (std::out_of_range &e) {
63         }
64         std::cout « "\t";
65     }
66     std::cout « std::endl;
67     for (const auto &confusion_matrix_line: confusion_matrix) {
68         int space_len = (int) longest_unique_label_size - (int) confusion_matrix_line.first.size();
69         try {
70             std::cout « "(" « (char) toupper(confusion_matrix_line.first.at(0));
71             std::cout « confusion_matrix_line.first.at(1);
72         } catch (std::out_of_range &e) {
73         }
74         std::cout « ") " « confusion_matrix_line.first « std::string(space_len, ' ') « "\t[";
75         std::size_t i = 0;
76         for (const auto &confusion_matrix_line_i: confusion_matrix_line.second) {
77             if (i >= confusion_matrix_line.second.size() - 1) {
78                 std::cout « (std::to_string(confusion_matrix_line_i.second));
79             } else {
80                 std::cout « (std::to_string(confusion_matrix_line_i.second) + "\t");
81             }
82             i++;
83         }
84         std::cout « "]" « std::endl;
85     }
86
87 }
```

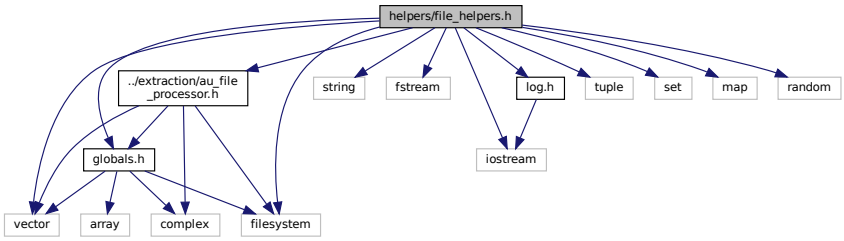
Here is the caller graph for this function:



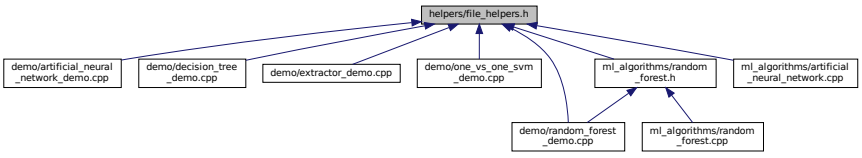
5.10 helpers/file_helpers.h File Reference

```
#include <vector>
#include <string>
#include <fstream>
#include <iostream>
#include <tuple>
#include <set>
#include <filesystem>
#include <map>
#include <random>
#include "globals.h"
#include "log.h"
#include "../extraction/au_file_processor.h"
```

Include dependency graph for file_helpers.h:



This graph shows which files directly or indirectly include this file:



Functions

- static std::vector< std::filesystem::path > alpha_dir_listing (const std::string &dir_path)
- static std::vector< std::filesystem::path > alpha_files_listing (const std::string &dir_path, const std::string &file_extension="none")
- static std::pair< std::vector< std::filesystem::path >, std::vector< std::filesystem::path > > select_train_test_files (std::vector< std::filesystem::path > files, double ratio)
- static std::vector< std::pair< std::string, real_vector_t > > get_features_vectors_from_csv (const std::filesystem::path &csv_file_path, const AuFileProcessingAlgorithm &processing_algorithm)

5.10.1 Function Documentation

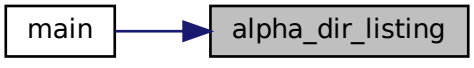
5.10.1.1 alpha_dir_listing()

```
static std::vector<std::filesystem::path> alpha_dir_listing (
            const std::string & dir_path )  [inline], [static]
```

Definition at line 17 of file file_helpers.h.

```
17                                                                                                  {
18     std::set<std::filesystem::path> sorted_by_name_dirs;
19     std::set<std::filesystem::path> sorted_by_name_files;
20     std::vector<std::filesystem::path> dirs_listing;
21     for (auto &entry: std::filesystem::directory_iterator(dir_path)) {
22         if (std::filesystem::is_directory(entry.path())) {
23             sorted_by_name_dirs.insert(entry.path());
24         }
25     }
26     std::copy(sorted_by_name_dirs.cbegin(), sorted_by_name_dirs.cend(),
       std::back_inserter(dirs_listing));
27     return dirs_listing;
28 }
```

Here is the caller graph for this function:



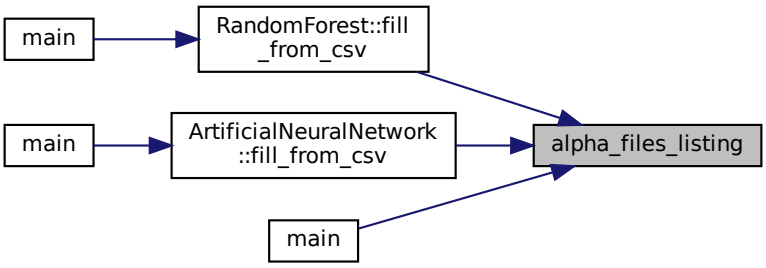
5.10.1.2 alpha_files_listing()

```
static std::vector<std::filesystem::path> alpha_files_listing (
            const std::string & dir_path,
            const std::string & file_extension = "none" )  [inline], [static]
```

Definition at line 30 of file file_helpers.h.

```
30                                                                              {
31     std::set<std::filesystem::path> sorted_by_name_files;
32     std::vector<std::filesystem::path> files_listing;
33     for (const std::filesystem::path &file: std::filesystem::directory_iterator(dir_path)) {
34         if (file_extension == "none") {
35             sorted_by_name_files.insert(file);
36         } else {
37             if (file.extension() == file_extension) {
38                 sorted_by_name_files.insert(file);
39             }
40         }
41     }
42     std::copy(sorted_by_name_files.cbegin(), sorted_by_name_files.cend(),
       std::back_inserter(files_listing));
43     return files_listing;
44 }
```

Here is the caller graph for this function:



5.10.1.3 get_features_vectors_from_csv()

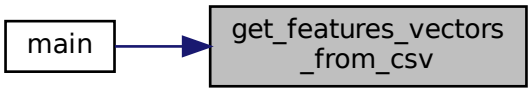
```
static std::vector<std::pair<std::string, real_vector_t> > get_features_vectors_from_csv (
            const std::filesystem::path & csv_file_path,
            const AuFileProcessingAlgorithm & processing_algorithm )  [inline], [static]
```

Definition at line 75 of file file_helpers.h.

```
75                                                                                                                  {
76     std::vector<std::pair<std::string, real_vector_t» features_vector_list = {};
77
78     // Get data length depending on processing algorithm
79     std::size_t data_length;
80     switch (processing_algorithm) {
81         case AuFileProcessingAlgorithm::STFT: {
82             // The data length for the STFT algorithm is FFT_SIZE*(2->avg+stdev)
83             data_length = FFT_SIZE * 2;
84             break;
85         }
86         case AuFileProcessingAlgorithm::MFCC: {
87             // The data length for the MFCC algorithm is [MEL_APPLIED_N+ (1->energy)] * (2->avg+stdev)
```

```
88              data_length = (MEL_APPLIED_N + 1) * 2;
89              break;
90          }
91          default: {
92              LOG(LOG_ERROR) « "Error : the processing algorithm value usage is not defined in the
    project";
93              throw std::domain_error("Unsupported processing algorithm!");
94              break;
95          }
96      }
97
98      const char delimiter = ',';
99      std::string line = {};
100      std::string data = {};
101
102      std::ifstream input_file(csv_file_path);
103      if (!input_file.is_open()) {
104          LOG(LOG_ERROR) « "Error : file with path " + csv_file_path.string() + "  not found.";
105          throw std::filesystem::filesystem_error("Can't open file!",
    std::make_error_code(std::errc::no_such_file_or_directory));
106      }
107
108      bool header_skipped = false;
109      while (std::getline(input_file, line)) {
110          if (!header_skipped) {
111              header_skipped = true;
112              continue;
113          } else {
114              std::stringstream ss(line);
115              size_t last = 0;
116              size_t next = 0;
117
118              // Get all values (data_length first data of csv line)
119              std::vector<real_t> new_fv = {};
120              for (std::size_t i = 0; i < data_length; i++) {
121                  next = line.find(delimiter, last);
122                  new_fv.push_back((real_t) std::stod(line.substr(last, next - last)));
123                  last = next + 1;
124              }
125              // Get the music style (first data after the N values of csv line)
126              next = line.find(delimiter, last);
127              std::string new_fv_style = line.substr(last, next - last);
128              // Remove double quote characters around the music style
129              new_fv_style.erase(remove(new_fv_style.begin(), new_fv_style.end(), '"'),
    new_fv_style.end());
130
131              features_vector_list.emplace_back(new_fv_style, new_fv);
132          }
133      }
134
135      return features_vector_list;
136  }
```

Here is the caller graph for this function:



5.10.1.4 select_train_test_files()

```
static std::pair<std::vector<std::filesystem::path>, std::vector<std::filesystem::path> >
select_train_test_files (
```

```
            std::vector< std::filesystem::path > files,
            double ratio )  [inline], [static]
```

Definition at line 47 of file file_helpers.h.

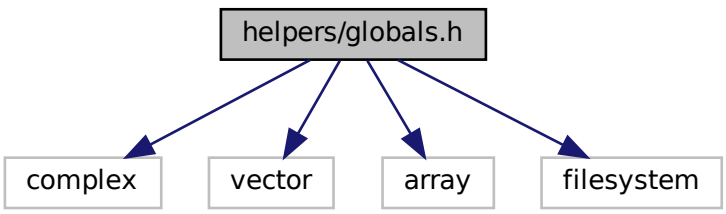
```
47                                                                                  {
48     std::size_t training_size = std::floor(files.size() * (1.0 - ratio));
49     //std::size_t testing_size = files.size() - training_size;
50     //std::cout « training_size « " " « testing_size « std::endl;
51     std::random_device random_device;
52     //std::mt19937 engine{66};
53     std::mt19937 engine{random_device()};
54     std::uniform_int_distribution<int> dist(0, files.size() - 1);
55     std::set<std::filesystem::path> training_files_set;
56     std::set<int> indexes;
57     for (std::size_t k = 0; k < training_size; k++) {
58         int random_index;
59         do {
60             random_index = dist(engine);
61         } while (indexes.contains(random_index));
62         indexes.insert(random_index);
63         training_files_set.insert(files[random_index]);
64     }
65     std::vector<std::filesystem::path> testing_files;
66     for (std::size_t k = 0; k < files.size(); k++) {
67         if (!indexes.contains(k))
68             testing_files.push_back(files[k]);
69     }
70     std::vector<std::filesystem::path> training_files;
71     std::copy(training_files_set.cbegin(), training_files_set.cend(),
       std::back_inserter(training_files));
72     return std::make_pair(training_files, testing_files);
73 }
```

Here is the caller graph for this function:

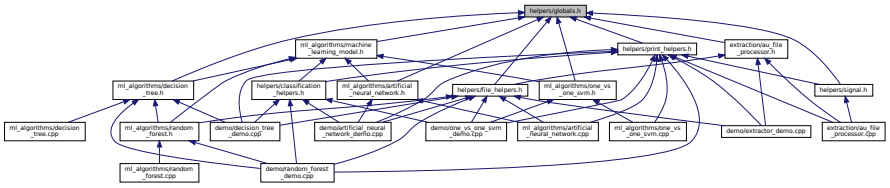
5.11 helpers/globals.h File Reference

```
#include <complex>
#include <vector>
#include <array>
#include <filesystem>
```

Include dependency graph for globals.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define KiB 1024
- #define MiB (1024∗1024)
- #define GiB (1024∗1024∗1024)

Typedefs

- typedef double real_t
- typedef std::complex< real_t > complex_t
- typedef uint32_t word_t
- typedef std::vector< real_t > real_vector_t
- typedef std::complex< complex_t > complex_vector_t
- typedef std::array< real_t, FFT_SIZE > real_fft_array_t
- typedef std::array< real_t, N > real_n_array_t
- typedef std::array< complex_t, FFT_SIZE > complex_fft_array_t
- typedef std::array< complex_t, N > complex_n_array_t

Variables

- const std::string MUSIC_FEATURES_STFT_CSV_TRAIN = "music_features_stft_train.csv"
- const std::string MUSIC_FEATURES_MFCC_CSV_TRAIN = "music_features_mfcc_train.csv"
- const std::string MUSIC_FEATURES_STFT_CSV_TEST = "music_features_stft_test.csv"
- const std::string MUSIC_FEATURES_MFCC_CSV_TEST = "music_features_mfcc_test.csv"
- const std::string DATASET_FOLDER = {"../../../datasets/"}
- const std::filesystem::path MUSIC_FEATURES_STFT_CSV_TRAIN_PATH = {DATASET_FOLDER + MUSIC_FEATURES_STFT_CSV_TRAIN}
- const std::filesystem::path MUSIC_FEATURES_MFCC_CSV_TRAIN_PATH = {DATASET_FOLDER + MUSIC_FEATURES_MFCC_CSV_TRAIN}
- const std::filesystem::path MUSIC_FEATURES_STFT_CSV_TEST_PATH = {DATASET_FOLDER + MUSIC_FEATURES_STFT_CSV_TEST}
- const std::filesystem::path MUSIC_FEATURES_MFCC_CSV_TEST_PATH = {DATASET_FOLDER + MUSIC_FEATURES_MFCC_CSV_TEST}
- const std::string DECISION_TREE_CSV_STFT = "cart_model_stft.csv"
- const std::string DECISION_TREE_CSV_MFCC = "cart_model_mfcc.csv"
- const std::string DECISION_TREE_FOLDER = {"../../../training/decision_tree/"}
- const std::filesystem::path DECISION_TREE_CSV_PATH_STFT = {DECISION_TREE_FOLDER + DECISION_TREE_CSV_STFT}
- const std::filesystem::path DECISION_TREE_CSV_PATH_MFCC = {DECISION_TREE_FOLDER + DECISION_TREE_CSV_MFCC}
- const std::string RANDOM_FOREST_TREES_FOLDER_STFT = "random_forest_trees_stft"

- const std::string RANDOM_FOREST_TREES_FOLDER_MFCC = "random_forest_trees_mfcc"
- const std::string RANDOM_FOREST_FOLDER = {"../../../training/random_forest/"}
- const std::filesystem::path RANDOM_FOREST_TREES_FOLDER_PATH_STFT = {RANDOM_FOREST_FOLDER + RANDOM_FOREST_TREES_FOLDER_STFT}
- const std::filesystem::path RANDOM_FOREST_TREES_FOLDER_PATH_MFCC = {RANDOM_FOREST_FOLDER + RANDOM_FOREST_TREES_FOLDER_MFCC}
- const std::string ONE_VS_ONE_SVM_CSV_STFT = "support_vector_machine_stft.csv"
- const std::string ONE_VS_ONE_SVM_CSV_MFCC = "support_vector_machine_mfcc.csv"
- const std::string ONE_VS_ONE_SVM_FOLDER = {"../../../training/support_vector_machine/"}
- const std::filesystem::path ONE_VS_ONE_SVM_CSV_PATH_STFT = {ONE_VS_ONE_SVM_FOLDER + ONE_VS_ONE_SVM_CSV_STFT}
- const std::filesystem::path ONE_VS_ONE_SVM_CSV_PATH_MFCC = {ONE_VS_ONE_SVM_FOLDER + ONE_VS_ONE_SVM_CSV_MFCC}
- const std::string ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_STFT = "artificial_neural_↩ network_stft"
- const std::string ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_MFCC = "artificial_neural_↩ network_mfcc"
- const std::string ARTIFICIAL_NEURAL_NETWORK_FOLDER = {"../../../training/artificial_neural_network/"}
- const std::filesystem::path ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_PATH_STFT = {ARTIFICIAL_NEURAL_NETWORK_FOLDER + ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_STFT}
- const std::filesystem::path ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_PATH_MFCC = {ARTIFICIAL_NEURAL_NETWORK_FOLDER + ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_MFCC}
- constexpr std::size_t N = 512
- constexpr std::size_t FFT_SIZE = N / 2
- constexpr std::size_t MEL_N = 26
- constexpr std::size_t MEL_APPLIED_N = 20
- constexpr real_t Fs = 22050.0

5.11.1 Macro Definition Documentation

5.11.1.1 GiB

```
#define GiB (1024*1024*1024)
```

Definition at line 11 of file globals.h.

5.11.1.2 KiB

```
#define KiB 1024
```

Definition at line 9 of file globals.h.

5.11.1.3 MiB

```
#define MiB (1024*1024)
```

Definition at line 10 of file globals.h.

5.11.2 Typedef Documentation

5.11.2.1 complex_fft_array_t

```
typedef std::array<complex_t, FFT_SIZE> complex_fft_array_t
```

Definition at line 66 of file globals.h.

5.11.2.2 complex_n_array_t

```
typedef std::array<complex_t, N> complex_n_array_t
```

Definition at line 67 of file globals.h.

5.11.2.3 complex_t

```
typedef std::complex<real_t> complex_t
```

Definition at line 51 of file globals.h.

5.11.2.4 complex_vector_t

```
typedef std::complex<complex_t> complex_vector_t
```

Definition at line 63 of file globals.h.

5.11.2.5 real_fft_array_t

```
typedef std::array<real_t, FFT_SIZE> real_fft_array_t
```

Definition at line 64 of file globals.h.

5.11.2.6 real_n_array_t

```
typedef std::array<real_t, N> real_n_array_t
```

Definition at line 65 of file globals.h.

5.11.2.7 real_t

```
typedef double real_t
```

Definition at line 50 of file globals.h.

5.11.2.8 real_vector_t

```
typedef std::vector<real_t> real_vector_t
```

Definition at line 62 of file globals.h.

5.11.2.9 word_t

```
typedef uint32_t word_t
```

Definition at line 52 of file globals.h.

5.11.3 Variable Documentation

5.11.3.1 ARTIFICIAL_NEURAL_NETWORK_FOLDER

```
const std::string ARTIFICIAL_NEURAL_NETWORK_FOLDER = {"../../../training/artificial_neural_↩
network/"}
```

Definition at line 44 of file globals.h.

5.11.3.2 ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_MFCC

```
const std::string ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_MFCC = "artificial_neural_network_↩
mfcc"
```

Definition at line 43 of file globals.h.

5.11.3.3 ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_PATH_MFCC

```
const std::filesystem::path ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_PATH_MFCC = {ARTIFICIAL_NEURAL_NETWORK_FOL
+ ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_MFCC}
```

Definition at line 46 of file globals.h.

5.11.3.4 ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_PATH_STFT

```
const std::filesystem::path ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_PATH_STFT = {ARTIFICIAL_NEURAL_NETWORK_FOL
+ ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_STFT}
```

Definition at line 45 of file globals.h.

5.11.3.5 ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_STFT

```
const std::string ARTIFICIAL_NEURAL_NETWORK_LAYERS_FOLDER_STFT = "artificial_neural_network_↩
stft"
```

Definition at line 42 of file globals.h.

5.11.3.6 DATASET_FOLDER

```
const std::string DATASET_FOLDER = {"../../../datasets/"}
```

Definition at line 18 of file globals.h.

5.11.3.7 DECISION_TREE_CSV_MFCC

```
const std::string DECISION_TREE_CSV_MFCC = "cart_model_mfcc.csv"
```

Definition at line 25 of file globals.h.

5.11.3.8 DECISION_TREE_CSV_PATH_MFCC

```
const std::filesystem::path DECISION_TREE_CSV_PATH_MFCC = {DECISION_TREE_FOLDER + DECISION_TREE_CSV_MFCC}
```

Definition at line 28 of file globals.h.

5.11.3.9 DECISION_TREE_CSV_PATH_STFT

```
const std::filesystem::path DECISION_TREE_CSV_PATH_STFT = {DECISION_TREE_FOLDER + DECISION_TREE_CSV_STFT}
```

Definition at line 27 of file globals.h.

5.11.3.10 DECISION_TREE_CSV_STFT

```
const std::string DECISION_TREE_CSV_STFT = "cart_model_stft.csv"
```

Definition at line 24 of file globals.h.

5.11.3.11 DECISION_TREE_FOLDER

```
const std::string DECISION_TREE_FOLDER = {"../../../training/decision_tree/"}
```

Definition at line 26 of file globals.h.

5.11.3.12 FFT_SIZE

```
constexpr std::size_t FFT_SIZE = N / 2  [constexpr]
```

Definition at line 56 of file globals.h.

5.11.3.13 Fs

```
constexpr real_t Fs = 22050.0  [constexpr]
```

Definition at line 59 of file globals.h.

5.11.3.14 MEL_APPLIED_N

```
constexpr std::size_t MEL_APPLIED_N = 20  [constexpr]
```

Definition at line 58 of file globals.h.

5.11.3.15 MEL_N

```
constexpr std::size_t MEL_N = 26  [constexpr]
```

Definition at line 57 of file globals.h.

5.11.3.16 MUSIC_FEATURES_MFCC_CSV_TEST

```
const std::string MUSIC_FEATURES_MFCC_CSV_TEST = "music_features_mfcc_test.csv"
```

Definition at line 17 of file globals.h.

5.11.3.17 MUSIC_FEATURES_MFCC_CSV_TEST_PATH

```
const std::filesystem::path MUSIC_FEATURES_MFCC_CSV_TEST_PATH = {DATASET_FOLDER + MUSIC_FEATURES_MFCC_CSV_TEST
```

Definition at line 22 of file globals.h.

5.11.3.18 MUSIC_FEATURES_MFCC_CSV_TRAIN

```
const std::string MUSIC_FEATURES_MFCC_CSV_TRAIN = "music_features_mfcc_train.csv"
```

Definition at line 15 of file globals.h.

5.11.3.19 MUSIC_FEATURES_MFCC_CSV_TRAIN_PATH

```
const std::filesystem::path MUSIC_FEATURES_MFCC_CSV_TRAIN_PATH = {DATASET_FOLDER + MUSIC_FEATURES_MFCC_CSV_TRA
```

Definition at line 20 of file globals.h.

5.11.3.20 MUSIC_FEATURES_STFT_CSV_TEST

```
const std::string MUSIC_FEATURES_STFT_CSV_TEST = "music_features_stft_test.csv"
```

Definition at line 16 of file globals.h.

5.11.3.21 MUSIC_FEATURES_STFT_CSV_TEST_PATH

```
const std::filesystem::path MUSIC_FEATURES_STFT_CSV_TEST_PATH = {DATASET_FOLDER + MUSIC_FEATURES_STFT_CSV_TEST
```

Definition at line 21 of file globals.h.

5.11.3.22 MUSIC_FEATURES_STFT_CSV_TRAIN

```
const std::string MUSIC_FEATURES_STFT_CSV_TRAIN = "music_features_stft_train.csv"
```

Definition at line 14 of file globals.h.

5.11.3.23 MUSIC_FEATURES_STFT_CSV_TRAIN_PATH

```
const std::filesystem::path MUSIC_FEATURES_STFT_CSV_TRAIN_PATH = {DATASET_FOLDER + MUSIC_FEATURES_STFT_CSV_TRA
```

Definition at line 19 of file globals.h.

5.11.3.24 N

```
constexpr std::size_t N = 512  [constexpr]
```

Definition at line 55 of file globals.h.

5.11.3.25 ONE_VS_ONE_SVM_CSV_MFCC

```
const std::string ONE_VS_ONE_SVM_CSV_MFCC = "support_vector_machine_mfcc.csv"
```

Definition at line 37 of file globals.h.

5.11.3.26 ONE_VS_ONE_SVM_CSV_PATH_MFCC

```
const std::filesystem::path ONE_VS_ONE_SVM_CSV_PATH_MFCC = {ONE_VS_ONE_SVM_FOLDER + ONE_VS_ONE_SVM_CSV_MFCC}
```

Definition at line 40 of file globals.h.

5.11.3.27 ONE_VS_ONE_SVM_CSV_PATH_STFT

```
const std::filesystem::path ONE_VS_ONE_SVM_CSV_PATH_STFT = {ONE_VS_ONE_SVM_FOLDER + ONE_VS_ONE_SVM_CSV_STFT}
```

Definition at line 39 of file globals.h.

5.11.3.28 ONE_VS_ONE_SVM_CSV_STFT

```
const std::string ONE_VS_ONE_SVM_CSV_STFT = "support_vector_machine_stft.csv"
```

Definition at line 36 of file globals.h.

5.11.3.29 ONE_VS_ONE_SVM_FOLDER

```
const std::string ONE_VS_ONE_SVM_FOLDER = {"../../../training/support_vector_machine/"}
```

Definition at line 38 of file globals.h.

5.11.3.30 RANDOM_FOREST_FOLDER

```
const std::string RANDOM_FOREST_FOLDER = {"../../../training/random_forest/"}
```

Definition at line 32 of file globals.h.

5.11.3.31 RANDOM_FOREST_TREES_FOLDER_MFCC

```
const std::string RANDOM_FOREST_TREES_FOLDER_MFCC = "random_forest_trees_mfcc"
```

Definition at line 31 of file globals.h.

5.11.3.32 RANDOM_FOREST_TREES_FOLDER_PATH_MFCC

```
const std::filesystem::path RANDOM_FOREST_TREES_FOLDER_PATH_MFCC = {RANDOM_FOREST_FOLDER +
RANDOM_FOREST_TREES_FOLDER_MFCC}
```

Definition at line 34 of file globals.h.

5.11.3.33 RANDOM_FOREST_TREES_FOLDER_PATH_STFT

```
const std::filesystem::path RANDOM_FOREST_TREES_FOLDER_PATH_STFT = {RANDOM_FOREST_FOLDER +
RANDOM_FOREST_TREES_FOLDER_STFT}
```

Definition at line 33 of file globals.h.

5.11.3.34 RANDOM_FOREST_TREES_FOLDER_STFT

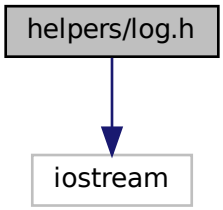
```
const std::string RANDOM_FOREST_TREES_FOLDER_STFT = "random_forest_trees_stft"
```

Definition at line 30 of file globals.h.

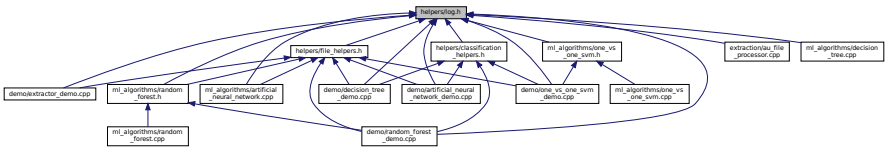
5.12 helpers/log.h File Reference

```
#include <iostream>
```

Include dependency graph for log.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct log_struct

  Hold structure for log config.
- class LOGGER

Macros

- #define LOG(type) LOGGER(type, __FILE__, __LINE__, __FUNCTION__)

Enumerations

- enum log_level_enum { LOG_DEBUG, LOG_INFO, LOG_WARNING, LOG_ERROR }

  List of log levels

Variables

- log_struct LOGGING_CONFIG

5.12.1 Macro Definition Documentation

5.12.1.1 LOG

```
#define LOG(
              type ) LOGGER(type, __FILE__, __LINE__, __FUNCTION__)
```

Definition at line 6 of file log.h.

5.12.2 Enumeration Type Documentation

5.12.2.1 log_level_enum

```
enum log_level_enum
```

List of log levels

Enumerator

LOG_DEBUG	Use for debuging messages
LOG_INFO	Use for basic execution message
LOG_WARNING	Use for non blocking errors
LOG_ERROR	Use for critical errors

Definition at line 15 of file log.h.

```
15                      {
16     LOG_DEBUG,
17     LOG_INFO,
18     LOG_WARNING,
19     LOG_ERROR
20 };
```

5.12.3 Variable Documentation

5.12.3.1 LOGGING_CONFIG

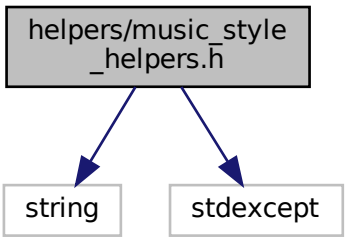
```
log_struct LOGGING_CONFIG
```

Definition at line 6 of file artificial_neural_network_demo.cpp.

5.13 helpers/music_style_helpers.h File Reference

```
#include <string>
#include <stdexcept>
```

Include dependency graph for music_style_helpers.h:



Enumerations

- enum MUSIC_STYLE {
  MUSIC_STYLE::BLUES = 0, MUSIC_STYLE::CLASSICAL = 1, MUSIC_STYLE::COUNTRY = 2,
  MUSIC_STYLE::DISCO = 3,
  MUSIC_STYLE::HIPHOP = 4, MUSIC_STYLE::JAZZ = 5, MUSIC_STYLE::METAL = 6, MUSIC_STYLE::POP
  = 7,
  MUSIC_STYLE::REGGAE = 8, MUSIC_STYLE::ROCK = 9 }

Functions

- static std::string music_style_to_string (const MUSIC_STYLE &s)
- static MUSIC_STYLE music_style_from_string (std::string str)
- static MUSIC_STYLE music_style_from_int (unsigned int s)
- std::ostream & operator<< (std::ostream &os, MUSIC_STYLE music_style)

5.13.1 Enumeration Type Documentation

5.13.1.1 MUSIC_STYLE

```
enum MUSIC_STYLE  [strong]
```

Enumerator

BLUES	
CLASSICAL	
COUNTRY	
DISCO	
HIPHOP	
JAZZ	
METAL	
POP	
REGGAE	
ROCK	

Definition at line 7 of file music_style_helpers.h.

```
7                          {
8      BLUES = 0,
9      CLASSICAL = 1,
10      COUNTRY = 2,
11      DISCO = 3,
12      HIPHOP = 4,
13      JAZZ = 5,
14      METAL = 6,
15      POP = 7,
16      REGGAE = 8,
17      ROCK = 9
18 };
```

5.13.2 Function Documentation

5.13.2.1 music_style_from_int()

```
static MUSIC_STYLE music_style_from_int (
            unsigned int s )  [inline], [static]
```

Definition at line 84 of file music_style_helpers.h.

```
84                                                                  {
85     switch (s) {
86         case 0: {
87             return MUSIC_STYLE::BLUES;
88         }
89         case 1: {
90             return MUSIC_STYLE::CLASSICAL;
91         }
92         case 2: {
93             return MUSIC_STYLE::COUNTRY;
94         }
95         case 3: {
96             return MUSIC_STYLE::DISCO;
97         }
98         case 4: {
99             return MUSIC_STYLE::HIPHOP;
100         }
101         case 5: {
102             return MUSIC_STYLE::JAZZ;
103         }
104         case 6: {
105             return MUSIC_STYLE::METAL;
106         }
107         case 7: {
108             return MUSIC_STYLE::POP;
109         }
110         case 8: {
111             return MUSIC_STYLE::REGGAE;
112         }
113         case 9: {
114             return MUSIC_STYLE::ROCK;
115         }
116         default: {
117             throw std::logic_error("to_int: MUSIC_STYLE enum values not found.");
118         }
119     }
120 }
```

5.13.2.2 music_style_from_string()

```
static MUSIC_STYLE music_style_from_string (
             std::string str )  [inline], [static]
```

Definition at line 58 of file music_style_helpers.h.

```
58                                                                   {
59     if (str == "blues") {
60         return MUSIC_STYLE::BLUES;
61     } else if (str == "classical") {
62         return MUSIC_STYLE::CLASSICAL;
63     } else if (str == "country") {
64         return MUSIC_STYLE::COUNTRY;
65     } else if (str == "disco") {
66         return MUSIC_STYLE::DISCO;
67     } else if (str == "hiphop") {
68         return MUSIC_STYLE::HIPHOP;
69     } else if (str == "jazz") {
70         return MUSIC_STYLE::JAZZ;
71     } else if (str == "metal") {
72         return MUSIC_STYLE::METAL;
73     } else if (str == "pop") {
74         return MUSIC_STYLE::POP;
75     } else if (str == "reggae") {
76         return MUSIC_STYLE::REGGAE;
77     } else if (str == "rock") {
78         return MUSIC_STYLE::ROCK;
79     } else {
80         throw std::logic_error("from_string: MUSIC_STYLE enum values not found.");
81     }
82 }
```

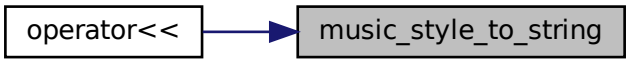
5.13.2.3 music_style_to_string()

```
static std::string music_style_to_string (
            const MUSIC_STYLE & s )  [inline], [static]
```

Definition at line 20 of file music_style_helpers.h.

```
20                                                                  {
21     switch (s) {
22         case MUSIC_STYLE::BLUES: {
23             return "blues";
24         }
25         case MUSIC_STYLE::CLASSICAL: {
26             return "classical";
27         }
28         case MUSIC_STYLE::COUNTRY: {
29             return "country";
30         }
31         case MUSIC_STYLE::DISCO: {
32             return "disco";
33         }
34         case MUSIC_STYLE::HIPHOP: {
35             return "hiphop";
36         }
37         case MUSIC_STYLE::JAZZ: {
38             return "jazz";
39         }
40         case MUSIC_STYLE::METAL: {
41             return "metal";
42         }
43         case MUSIC_STYLE::POP: {
44             return "pop";
45         }
46         case MUSIC_STYLE::REGGAE: {
47             return "reggae";
48         }
49         case MUSIC_STYLE::ROCK: {
50             return "rock";
51         }
52         default: {
53             throw std::logic_error("to_string: MUSIC_STYLE enum values not found.");
54         }
55     }
56 }
```

Here is the caller graph for this function:



5.13.2.4 operator<<()

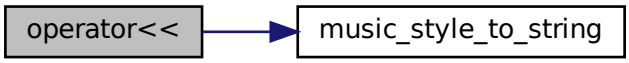
```
std::ostream& operator<< (
            std::ostream & os,
            MUSIC_STYLE music_style )
```

Definition at line 122 of file music_style_helpers.h.

```
122                                                                 {
123     os « music_style_to_string(music_style);
124     return os;
```

```
125 }
```

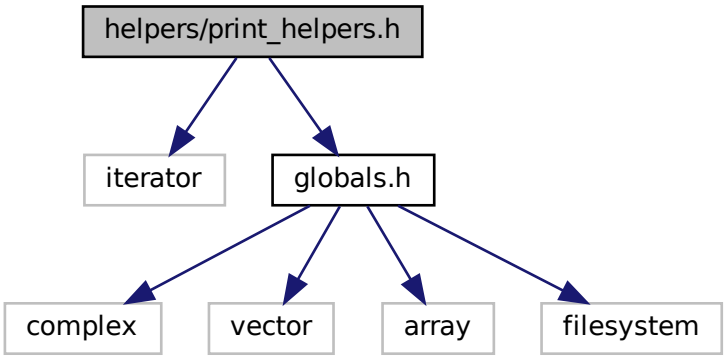
Here is the call graph for this function:



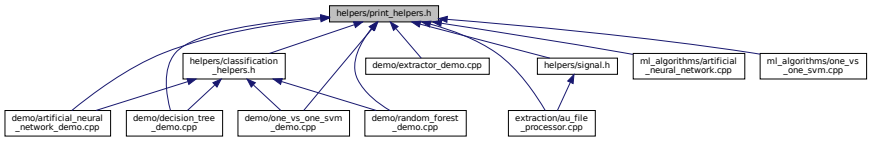
5.14 helpers/print_helpers.h File Reference

```
#include <iterator>
#include "globals.h"
```

Include dependency graph for print_helpers.h:



This graph shows which files directly or indirectly include this file:



Functions

- template<typename T >
  std::ostream & operator<< (std::ostream &os, const std::vector< T > &v)
- template<typename T , std::size_t SIZE>
  std::ostream & operator<< (std::ostream &os, const std::array< T, SIZE > &a)

5.14.1 Function Documentation

5.14.1.1 operator<<() [1/2]

```
template<typename T , std::size_t SIZE>
std::ostream& operator<< (
            std::ostream & os,
            const std::array< T, SIZE > & a )
```

Definition at line 22 of file print_helpers.h.

```
22                                                                      {
23     os « "(";
24     for (typename std::array<T, SIZE>::const_iterator i = a.begin(); i != a.end(); ++i) {
25         if (i == a.end() - 1) {
26             os « *i;
27         } else {
28             os « *i « ", ";
29         }
30     }
31     os « ")";
32     return os;
33 }
```

5.14.1.2 operator<<() [2/2]

```
template<typename T >
std::ostream& operator<< (
            std::ostream & os,
            const std::vector< T > & v )
```

Definition at line 8 of file print_helpers.h.

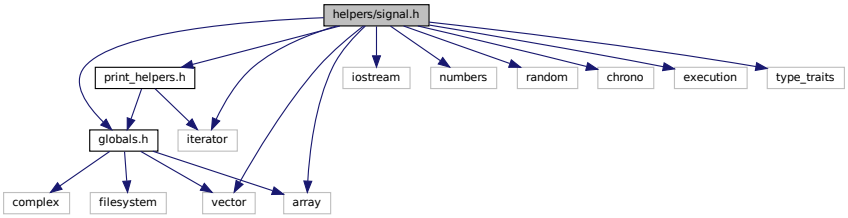
```
8                                                                   {
9     os « "[";
10     for (typename std::vector<T>::const_iterator i = v.begin(); i != v.end(); ++i) {
11         if (i == v.end() - 1) {
12             os « *i;
13         } else {
14             os « *i « ", ";
15         }
16     }
17     os « "]";
18     return os;
19 }
```

5.15 helpers/signal.h File Reference

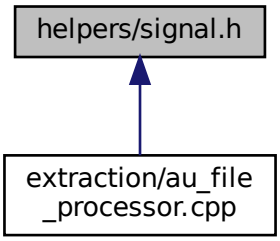
```
#include "globals.h"
#include <iostream>
#include <iterator>
#include <numbers>
#include <array>
#include <random>
#include <chrono>
#include <execution>
#include <type_traits>
#include <vector>
```

```
#include "print_helpers.h"
```

Include dependency graph for signal.h:



This graph shows which files directly or indirectly include this file:



Functions

- constexpr std::array< real_fft_array_t, MEL_N > mfcc_filters ()
- constexpr real_n_array_t hamming_window ()
- static void windowing (const real_n_array_t &w, complex_n_array_t &a)
- real_vector_t apply_filterbank (std::array< real_fft_array_t, MEL_APPLIED_N > filterbank, real_fft_array_t e)
- std::array< real_t, MEL_APPLIED_N > dct (real_vector_t e)
- real_vector_t dct2 (const real_vector_t &v_in)
- constexpr complex_fft_array_t twiddle_factors ()
- constexpr std::array< std::size_t, N > bit_reverse_array ()
- static void ite_dit_fft (complex_n_array_t &x)

5.15.1 Function Documentation

5.15.1.1 apply_filterbank()

```
real_vector_t apply_filterbank (
            std::array< real_fft_array_t, MEL_APPLIED_N > filterbank,
            real_fft_array_t e )
```

Definition at line 111 of file signal.h.

```
111
      {
112     real_vector_t value_filtering;
113     double filter_value;
114     for (real_fft_array_t filter: filterbank) {
115         filter_value = std::transform_reduce(e.cbegin(),
116                                              e.cend(),
117                                              filter.cbegin(),
118                                              0.0,
119                                              std::plus<>(),
120                                              std::multiplies<>());
121         value_filtering.push_back(std::log(std::max(filter_value, 2e-22)));
122     }
123     return value_filtering;
124 }
```

Here is the caller graph for this function:

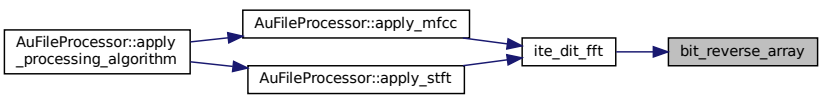
5.15.1.2 bit_reverse_array()

```
constexpr std::array<std::size_t, N> bit_reverse_array ( )  [constexpr]
```

Definition at line 186 of file signal.h.

```
186                                                 {
187     std::array<std::size_t, N> unscrambled{};
188     std::size_t m = std::log2(N);
189     //std::size_t m = ((unsigned) (8 * sizeof(unsigned long long) - __builtin_clzll((N)) - 1));
190     for (std::size_t i = 0; i < N; i++) {
191         std::size_t j = i;
192         j = (((j & 0xaaaaaaaa) >> 1) | ((j & 0x55555555) << 1));
193         j = (((j & 0xcccccccc) >> 2) | ((j & 0x33333333) << 2));
194         j = (((j & 0xf0f0f0f0) >> 4) | ((j & 0x0f0f0f0f) << 4));
195         j = (((j & 0xff00ff00) >> 8) | ((j & 0x00ff00ff) << 8));
196         j = ((j >> 16) | (j << 16)) >> (32 - m);
197         if (i < j) {
198             unscrambled[i] = j;
199         } else
200             unscrambled[i] = i;
201     }
202     return unscrambled;
203 }
```

Here is the caller graph for this function:



5.15.1.3 dct()

```
std::array<real_t, MEL_APPLIED_N> dct (
            real_vector_t e )
```

Definition at line 132 of file signal.h.

```
132                                                 {
133     std::array<real_t, MEL_APPLIED_N> cepstrum{};
134     size_t sous_index, index;
135     for (index = 1; index <= MEL_APPLIED_N; index++) {
136         cepstrum[index - 1] = 0.0;
137         for (sous_index = 1; sous_index <= MEL_APPLIED_N; sous_index++) {
138             cepstrum[index - 1] = cepstrum[index - 1] + e[sous_index - 1] *
139                                                         cos(M_PI * ((real_t) index) / ((real_t)
       MEL_APPLIED_N) *
140                                                             ((real_t) sous_index - 0.5));
141             cepstrum[index - 1] = sqrt(2.0 / (real_t) MEL_APPLIED_N) * cepstrum[index - 1];
142         }
143     }
144     return cepstrum;
145 }
```

5.15.1.4 dct2()

```
real_vector_t dct2 (
            const real_vector_t & v_in )
```

Definition at line 151 of file signal.h.

```
151                                             {
152     real_vector_t v_out = {};
153     for (std::size_t k = 0; k < v_in.size(); k++) {
154         real_vector_t u_k = {};
155         real_t a = k > 0 ? std::sqrt(2.0 / (real_t) v_in.size()) : std::sqrt(1.0 / (real_t)
       v_in.size());
156         for (std::size_t n = 0; n < v_in.size(); n++) {
157             u_k.push_back(a * std::cos(M_PI / (real_t) v_in.size() * ((real_t) n + 0.5) * (real_t) k));
158         }
159
160         real_t t_k = std::transform_reduce(v_in.cbegin(), v_in.cend(), u_k.cbegin(), 0.0, std::plus<>(),
       std::multiplies<>());
161         v_out.push_back(t_k);
162     }
163
164     return v_out;
165 }
```

Here is the caller graph for this function:

5.15.1.5 hamming_window()

```
constexpr real_n_array_t hamming_window ( )  [constexpr]
```

Definition at line 81 of file signal.h.

```
81                                          {
82     real_n_array_t w;
83     std::generate(w.begin(), w.end(),
84                   [&, index = -1]()mutable {
85                       index++;
86                       return (0.54 - 0.46 * std::cos(2 * std::numbers::pi * index / (N - 1)));
87                   });
88     return w;
89 }
```

Here is the caller graph for this function:

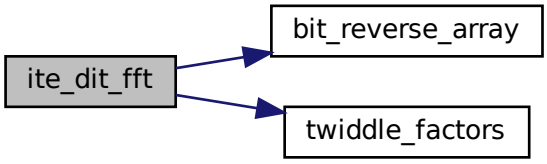
5.15.1.6 ite_dit_fft()

```
static void ite_dit_fft (
             complex_n_array_t & x )  [inline], [static]
```

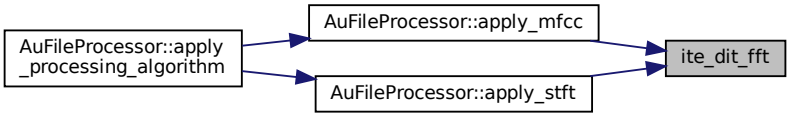
Definition at line 209 of file signal.h.

```
209                                                  {
210     std::size_t problemSize = x.size();
211     std::size_t stages = std::log2(problemSize);
212     auto tf = twiddle_factors();
213 
214     constexpr std::array<std::size_t, N> unscrambled = bit_reverse_array();
215     for (std::size_t i = 0; i < x.size(); i++) {
216         std::size_t j = unscrambled[i];
217         if (i < j) {
218             swap(x[i], x[j]);
219         }
220     }
221 
222     for (std::size_t stage = 0; stage <= stages; stage++) {
223         std::size_t currentSize = 1 « stage;
224         std::size_t step = stages - stage;
225         std::size_t halfSize = currentSize / 2;
226         for (std::size_t k = 0; k < problemSize; k = k + currentSize) {
227             //for (std::size_t k = 0; k <= problemSize - currentSize; k = k + currentSize) {
228             for (std::size_t j = 0; j < halfSize; j++) {
229                 auto u = x[k + j];
230                 auto v = x[k + j + halfSize] * tf[j * (1 « step)];
231                 x[k + j] = (u + v);
232                 x[k + j + halfSize] = (u - v);
233             }
234         }
235     }
236 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.15.1.7 mfcc_filters()

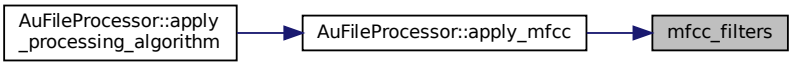
```
constexpr std::array<real_fft_array_t, MEL_N> mfcc_filters ( )  [constexpr]
```

Definition at line 21 of file signal.h.

```
21                                                      {
22     // Création de 24 filtres se recoupant allant de 20Hz à 22050Hz
23     // Soit 31mels à 3923 mels d'aprés l'équations m=2595*log10(1+f/700)
24     // On obtient:
25     const int mel_min = 31;
26     const int mel_max = 3923;
27 
28     double mel_inc = (mel_max - mel_min) / (real_t) (MEL_N + 1);
29     std::size_t index, sous_index;
30     std::array<real_t, MEL_N> mel_centers;
31     std::array<real_t, MEL_N> fcenters_norm;
32     std::array<real_fft_array_t, MEL_N> w = {}; // ToReturn
33     double index_start[MEL_N];   //FFT index for the first sample of each filter
34     double index_stop[MEL_N];    //FFT index for the last sample of each filter
35     float increment, decrement; // increment and decrement of the left and right ramp
36     double sum = 0.0;
37     //for(index=0; index<MEL_FILTERS_N; index++){
38     //    for(sous_index=0; sous_index<= FFT_SIZE/2 + 1; sous_index++){
39     //        w[index][sous_index] = 0.0;
40     //    }
41     //}
42     for (index = 1; index <= MEL_N; index++) {
43         mel_centers[index - 1] = (real_t) index * mel_inc + mel_min;
44         fcenters_norm[index - 1] = 700.0 * (pow(10.0, mel_centers[index - 1] / 2595.0) - 1.0);
45         fcenters_norm[index - 1] = round(fcenters_norm[index - 1] / (Fs / N));
46     }
47     //std::cout « std::endl;
48     for (index = 1; index <= (MEL_N - 1); index++) {
49         index_start[index] = fcenters_norm[index - 1];
50         index_stop[index - 1] = fcenters_norm[index];
```

```
51      }
52      index_start[0] = round((real_t) N * 20.0 / (real_t) Fs);
53      index_stop[MEL_N - 1] = round((real_t) N * 22050.0 / (real_t) Fs);
54      for (index = 1; index < MEL_N; index++) {
55          increment = 1. / ((real_t) fcenters_norm[index - 1] - (real_t) index_start[index - 1]); //Parite
        montante du triangle
56          for (sous_index = index_start[index - 1]; sous_index <= fcenters_norm[index - 1]; sous_index++) {
57              w[index - 1][sous_index] = ((real_t) sous_index - (real_t) index_start[index - 1]) *
        increment;
58          }
59          decrement = 1. / ((real_t) index_stop[index - 1] - (real_t) fcenters_norm[index - 1]); //Partie
        descendante du triangle
60          for (sous_index = fcenters_norm[index - 1]; sous_index <= index_stop[index - 1]; sous_index++) {
61              w[index - 1][sous_index] = ((real_t) sous_index - (real_t) fcenters_norm[index - 1]) *
        decrement;
62          }
63      }
64      for (index = 1; index <= MEL_N; index++) {
65          for (sous_index = 1; sous_index <= FFT_SIZE; sous_index++) {
66              sum = sum + w[index - 1][sous_index - 1];
67          }
68          for (sous_index = 1; sous_index <= FFT_SIZE; sous_index++) {
69              w[index - 1][sous_index - 1] = w[index - 1][sous_index - 1] / sum;
70          }
71          sum = 0.0;
72
73      }
74      return w;
75  }
```

Here is the caller graph for this function:



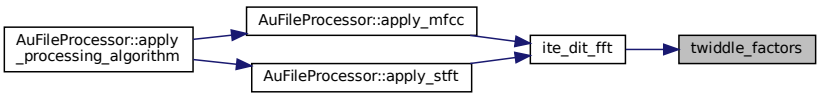
5.15.1.8 twiddle_factors()

```
constexpr complex_fft_array_t twiddle_factors ( )  [constexpr]
```

Definition at line 174 of file signal.h.

```
174                                             {
175      std::array<complex_t, N / 2> t;
176      for (std::size_t k = 0; k < N / 2; k++) {
177          t[k] = std::polar(1.0, -2.0 * std::numbers::pi * k / N);
178      }
179      return t;
180  }
```

Here is the caller graph for this function:



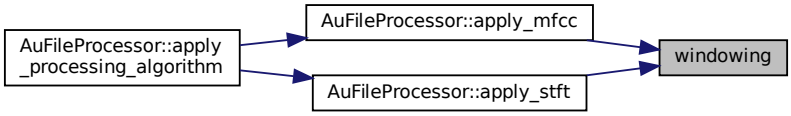
5.15.1.9 windowing()

```
static void windowing (
             const real_n_array_t & w,
             complex_n_array_t & a )  [inline], [static]
```

Definition at line 96 of file signal.h.

```
96                                                                          {
97     std::transform(a.cbegin(),
98                    a.cend(),
99                    a.begin(),
100                    [&, index = -1](complex_t c)mutable {
101                        index++;
102                        return w[index] * c;
103                    });
104 }
```

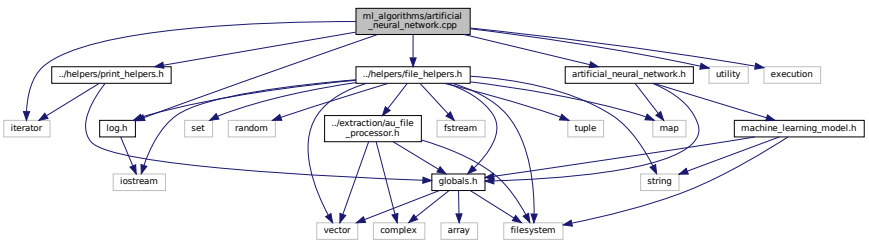
Here is the caller graph for this function:



5.16 ml_algorithms/artificial_neural_network.cpp File Reference

```
#include "../helpers/print_helpers.h"
#include "../helpers/file_helpers.h"
#include "../helpers/log.h"
#include "artificial_neural_network.h"
#include <utility>
#include <execution>
#include <iterator>
```

Include dependency graph for artificial_neural_network.cpp:



Functions

- std::ostream & operator<< (std::ostream &os, const Neuron &neuron)
- std::ostream & operator<< (std::ostream &os, const ArtificialNeuralNetwork &artificial_neural_network)

5.16.1 Function Documentation

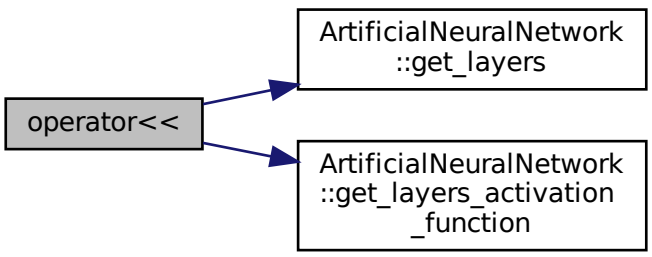
5.16.1.1 operator<<() [1/2]

```
std::ostream& operator<< (
            std::ostream & os,
            const ArtificialNeuralNetwork & artificial_neural_network )
```

Definition at line 69 of file artificial_neural_network.cpp.

```
69                                                                                                  {
70     for (const std::pair<std::size_t, std::vector<Neuron» layer: artificial_neural_network.get_layers())
       {
71         os « "layer " « layer.first « " (" « layer.second.size() « ")";
72         switch (artificial_neural_network.get_layers_activation_function().at(layer.first)) {
73             case ActivationFunction::SIGMOID : {
74                 os « " activation function SIGMOID";
75                 break;
76             }
77             case ActivationFunction::RELU : {
78                 os « " activation function RELU";
79                 break;
80             }
81             case ActivationFunction::SOFTMAX : {
82                 os « " activation function SOFTMAX";
83                 break;
84             }
85             default: {
86                 os « " activation function UNKNOWN";
87                 break;
88             }
89         }
90         os « ": \n";
91         for (const Neuron &neuron: layer.second) {
92             os « "\t - " « neuron « "\n";
93         }
94     }
95     return os;
96 }
```

Here is the call graph for this function:



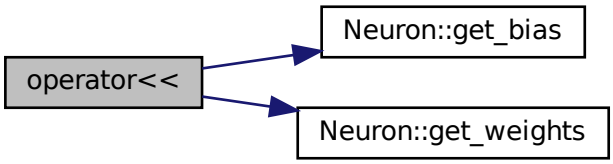
5.16.1.2 operator<<() [2/2]

```
std::ostream& operator<< (
            std::ostream & os,
            const Neuron & neuron )
```

Definition at line 26 of file artificial_neural_network.cpp.

```
26                                                          {
27     return os « "(bias: " « neuron.get_bias() « ", weights (" « neuron.get_weights().size() « "): " «
       neuron.get_weights() « ")";
28 }
```

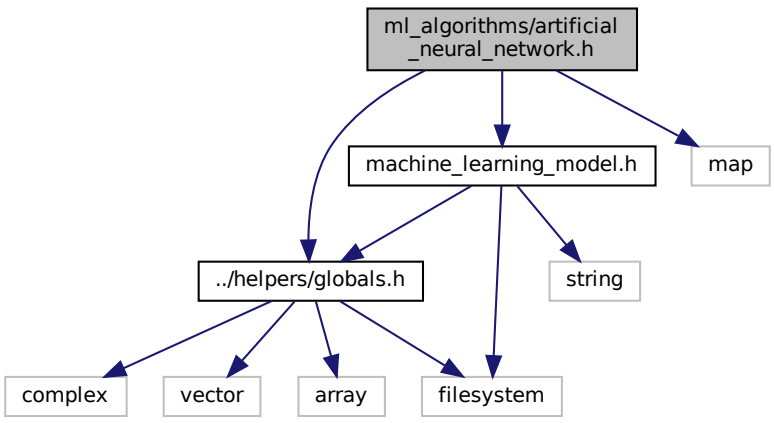
Here is the call graph for this function:



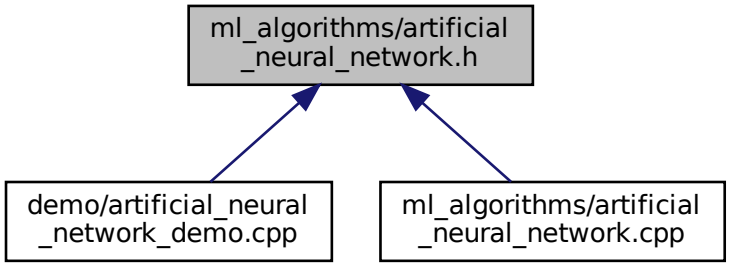
5.17 ml_algorithms/artificial_neural_network.h File Reference

```
#include "../helpers/globals.h"
#include "machine_learning_model.h"
#include <map>
```

Include dependency graph for artificial_neural_network.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Neuron
- class ArtificialNeuralNetwork

Enumerations

- enum ActivationFunction { ActivationFunction::SIGMOID = 0, ActivationFunction::RELU = 1, ActivationFunction::SOFTMAX = 2 }

  List of activation functions.

5.17.1 Enumeration Type Documentation

5.17.1.1 ActivationFunction

```
enum ActivationFunction  [strong]
```

List of activation functions.

Enumerator

SIGMOID	
RELU	The sigmoid function y = 1/(1+exp(-x))
SOFTMAX	The relu function y = max(0, x)

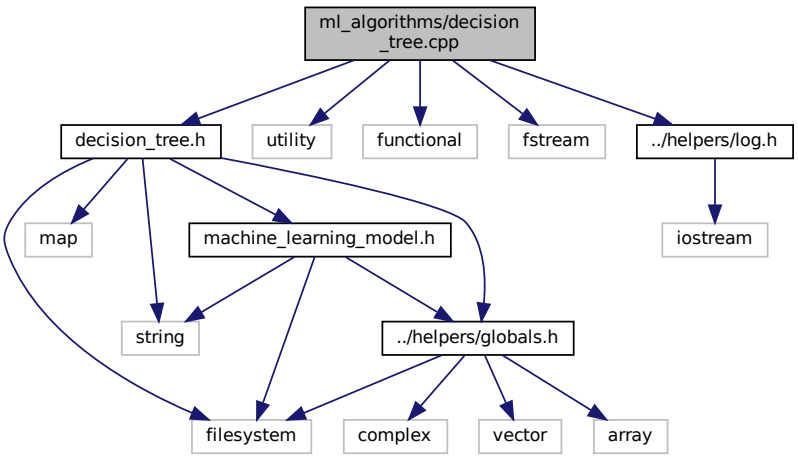
Definition at line 11 of file artificial_neural_network.h.

```
11                                  {
12     SIGMOID = 0,
13     RELU = 1,
14     SOFTMAX = 2
15 };
```

5.18 ml_algorithms/decision_tree.cpp File Reference

```
#include "decision_tree.h"
#include <utility>
#include <functional>
#include <fstream>
#include "../helpers/log.h"
```

Include dependency graph for decision_tree.cpp:



Functions

- std::ostream & operator<< (std::ostream &os, const TreeNode &tree_node)
- std::ostream & operator<< (std::ostream &os, const DecisionTree &decision_tree)

5.18.1 Function Documentation

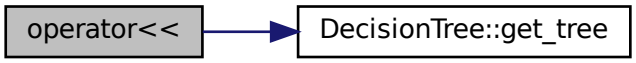
5.18.1.1 operator<<() [1/2]

```
std::ostream& operator<< (
            std::ostream & os,
            const DecisionTree & decision_tree )
```

Definition at line 97 of file decision_tree.cpp.

```
97                                                                          {
98     for (const std::pair<const unsigned long, TreeNode> &tree: decision_tree.get_tree()) {
99         os « "\n- node[" « tree.first « "]: " « tree.second;
100     }
101     return os;
102 }
```

Here is the call graph for this function:



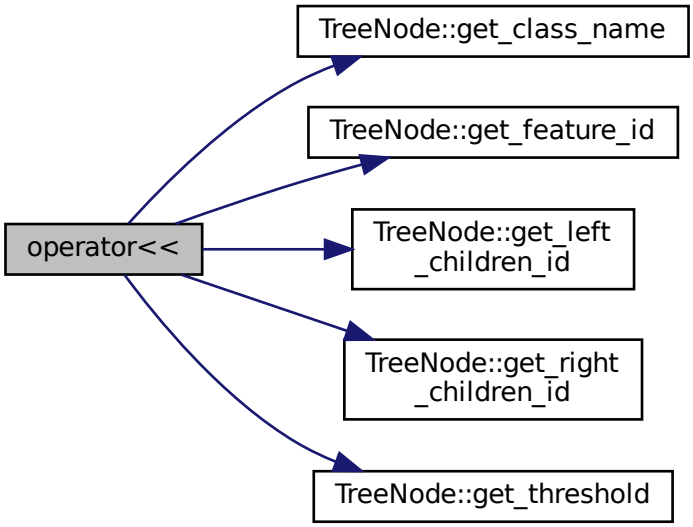
5.18.1.2 operator<<() [2/2]

```
std::ostream& operator<< (
            std::ostream & os,
            const TreeNode & tree_node )
```

Definition at line 37 of file decision_tree.cpp.

```
37                                                                  {
38     return os « "(class: " « tree_node.get_class_name() « ", treshold: " « tree_node.get_threshold() « ",
       feature_id: " « tree_node.get_feature_id() « ", left_id: " « tree_node.get_left_children_id() « ",
       right_id: " « tree_node.get_right_children_id() « ")";
39 }
```

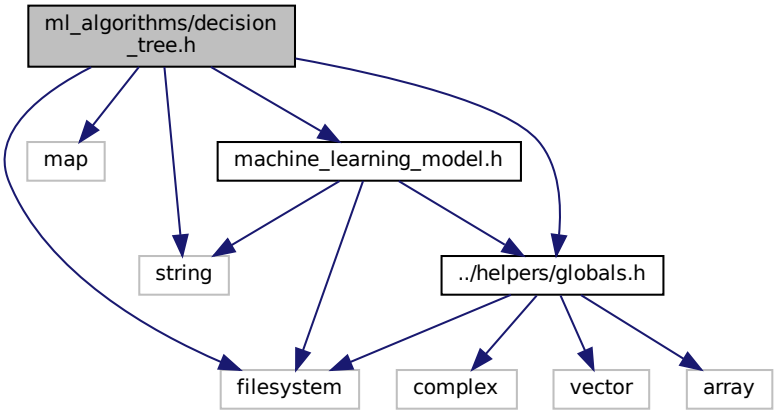
Here is the call graph for this function:



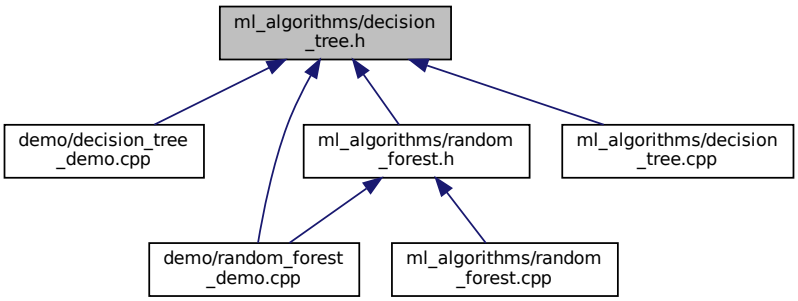
5.19 ml_algorithms/decision_tree.h File Reference

```
#include <filesystem>
#include <map>
#include <string>
#include "machine_learning_model.h"
#include "../helpers/globals.h"
```

Include dependency graph for decision_tree.h:



This graph shows which files directly or indirectly include this file:



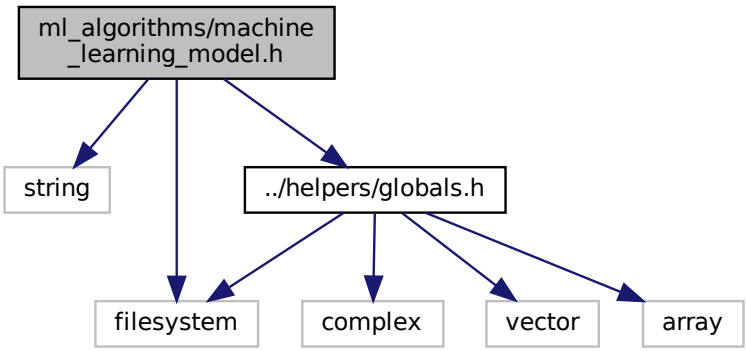
Classes

- class TreeNode
- class DecisionTree

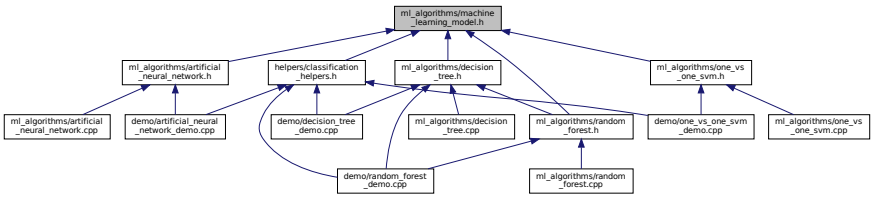
5.20 ml_algorithms/machine_learning_model.h File Reference

```
#include <string>
#include <filesystem>
#include "../helpers/globals.h"
```

Include dependency graph for machine_learning_model.h:



This graph shows which files directly or indirectly include this file:



Classes

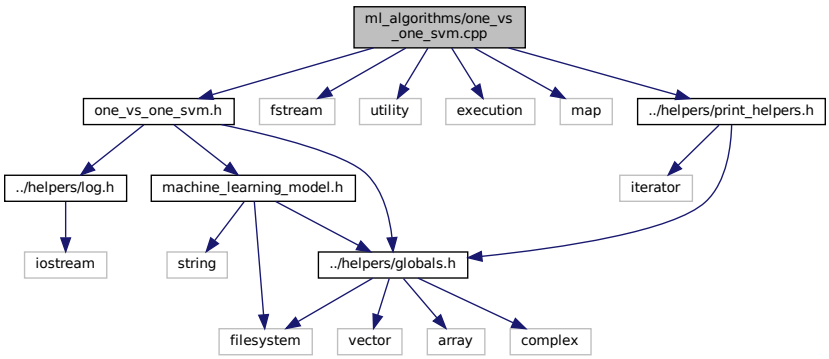
- class MachineLearningModel

5.21 ml_algorithms/one_vs_one_svm.cpp File Reference

```
#include "one_vs_one_svm.h"
#include <fstream>
#include <utility>
#include <execution>
#include <map>
```

```
#include "../helpers/print_helpers.h"
```

Include dependency graph for one_vs_one_svm.cpp:



Functions

- std::ostream & operator<< (std::ostream &os, const LinearClassifier &linear_classifier)
- std::ostream & operator<< (std::ostream &os, const OneVsOneSVM &one_vs_one_svm)

5.21.1 Function Documentation

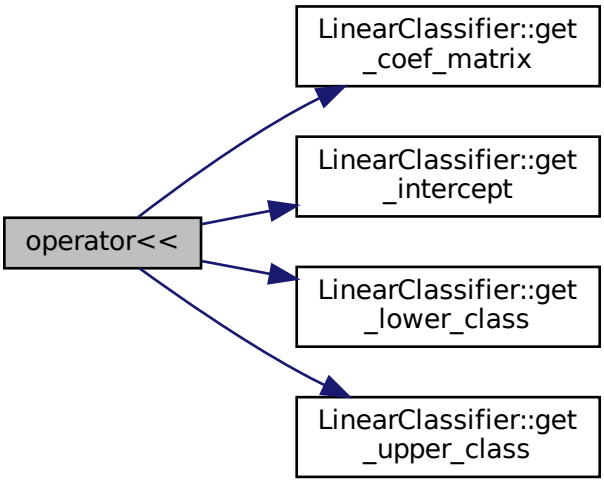
5.21.1.1 operator<<() [1/2]

```
std::ostream& operator<< (
            std::ostream & os,
            const LinearClassifier & linear_classifier )
```

Definition at line 33 of file one_vs_one_svm.cpp.

```
33                                                                                  {
34     // A for_each can be used here because the size of the matrix_a string will not be too long
35     size_t coef_matrix_len = linear_classifier.get_coef_matrix().size();
36     std::string matrix_a = "[";
37     std::for_each(std::execution::seq, linear_classifier.get_coef_matrix().cbegin(),
       linear_classifier.get_coef_matrix().cend(), [&matrix_a, &coef_matrix_len, i = 0](real_t r)mutable {
38         matrix_a += (i < int(coef_matrix_len)) ? std::to_string(r) + ", " : std::to_string(r);
39     });
40     matrix_a += "]";
41     return os « "y >= Ax+b -> class id is " « linear_classifier.get_upper_class() « " else class id is" «
       linear_classifier.get_lower_class() « " with A=" « matrix_a « " and b=" «
       std::to_string(linear_classifier.get_intercept());
42 }
```

Here is the call graph for this function:



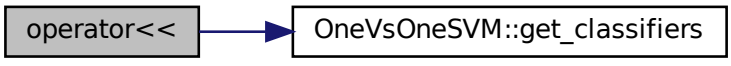
5.21.1.2 operator<<() [2/2]

```
std::ostream& operator<< (
            std::ostream & os,
            const OneVsOneSVM & one_vs_one_svm )
```

Definition at line 84 of file one_vs_one_svm.cpp.

```
84                                                                          {
85     // A for_each is not used here because the size of the final string can be really long
86     os « "One Vs One Linear SVM classifiers:" « std::endl;
87     std::size_t i = 0;
88     for (const LinearClassifier &c: one_vs_one_svm.get_classifiers()) {
89         os « "\t[" « i « "] " « c « std::endl;
90         i++;
91     }
92     return os;
93 }
```

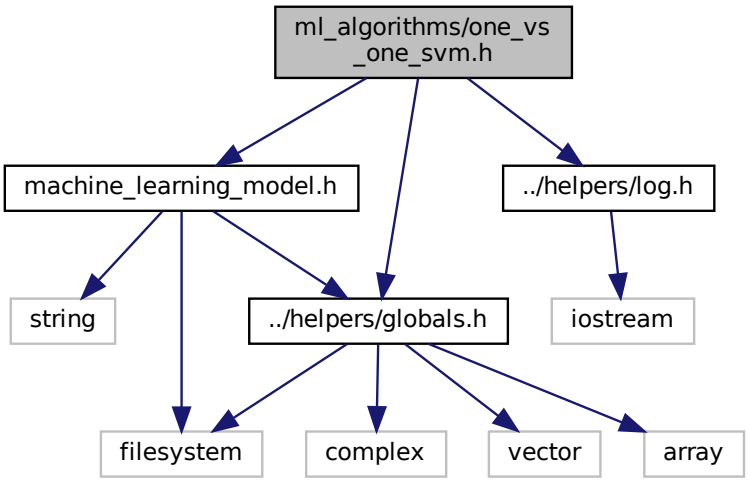
Here is the call graph for this function:



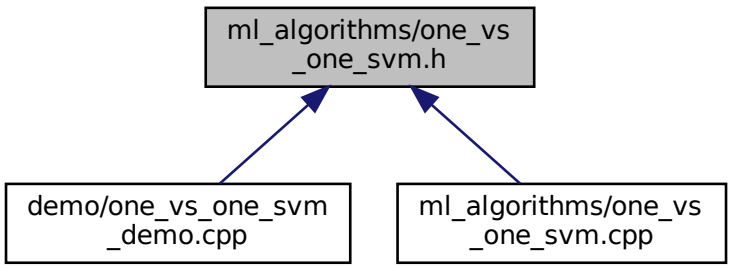
5.22 ml_algorithms/one_vs_one_svm.h File Reference

```
#include "machine_learning_model.h"
#include "../helpers/globals.h"
#include "../helpers/log.h"
```

Include dependency graph for one_vs_one_svm.h:



This graph shows which files directly or indirectly include this file:



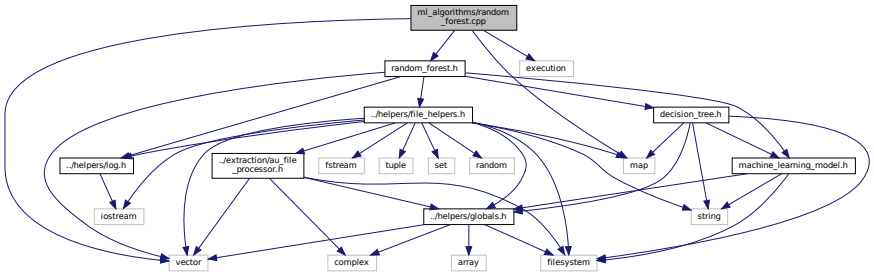
Classes

- class LinearClassifier
- class OneVsOneSVM

5.23 ml_algorithms/random_forest.cpp File Reference

```
#include <vector>
#include <map>
#include <execution>
#include "random_forest.h"
```

Include dependency graph for random_forest.cpp:



Functions

- std::ostream & operator<< (std::ostream &os, const RandomForest &random_forest)

5.23.1 Function Documentation

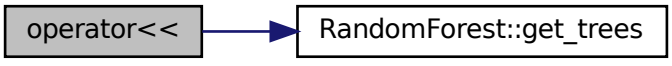
5.23.1.1 operator<<()

```
std::ostream& operator<< (
            std::ostream & os,
            const RandomForest & random_forest )
```

Definition at line 25 of file random_forest.cpp.

```
25                                                                          {
26     for (const DecisionTree &tree: random_forest.get_trees()) {
27         os « "\n - tree[depth: " « tree.get_depth() « ", number_of_nodes: " « tree.get_number_of_nodes()
       « "]";
28     }
29     return os;
30 }
```

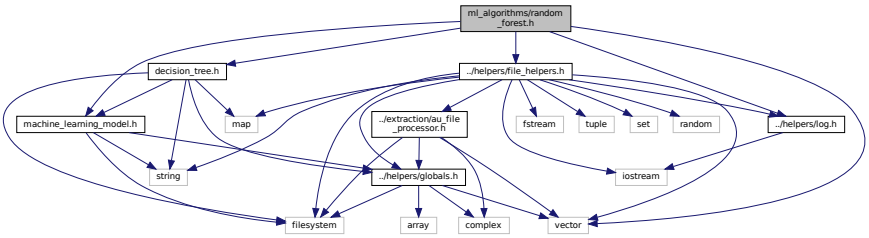
Here is the call graph for this function:



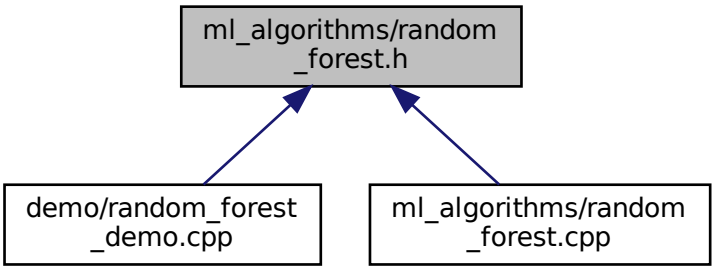
5.24 ml_algorithms/random_forest.h File Reference

```
#include <vector>
#include "decision_tree.h"
#include "machine_learning_model.h"
#include "../helpers/log.h"
#include "../helpers/file_helpers.h"
```

Include dependency graph for random_forest.h:



This graph shows which files directly or indirectly include this file:



Classes

- class RandomForest