USER'S MANUAL
Part 4:
The aiNet DLL Library

The aiNet DLL Library

Introduction

A lot of people around the World who are using aiNet, expressed their need for an additional tool, which would give them an opportunity to customise the aiNet neural network algorithm according to their specific needs. We have considered several different methods to make the algorithm available to meet our customer needs and finally we decided to provide one solution in two slightly different forms. Basically, the solution is a library of functions, which may be called from within a customer's application. The first form of this solution is a library of C functions encapsulated in a DLL library. The second form is a library of C++ classes which will be made available in future releases of aiNet. This solution is simple, compact, adaptable and efficient. We also found that people who required the additional tools tended to be knowledgeable about computer programming and therefore the libraries could be put to immediate use.

The C library is limited to the prediction functions only. The C++ library will consist of the full set of methods and classes which are used in aiNet.

Notes about 16 and 32 bit versions of aiNet DLL

The aiNet DLL library comes in two forms: as a 16 bit DLL and as a 32 bit DLL, named as AINET16.DLL and AINET32.DLL, respectively. If your compiler can produce 32 bit code, than we recommend that you to use the 32 bit library, since the 32 bit code runs significantly faster.

How to use the aiNet DLL library

The aiNet DLL library is intended to be used together with aiNet. The global problem analysis and modelling should be done with aiNet, as described in previous sections of the manual. When a good model is found, the user may then write his/her own application which will be based on the resulting model. The library is given as a DLL (dynamic link library) hence any computer language which can call functions in a DLL may be used. Examples using C and Visual Basic are provided.

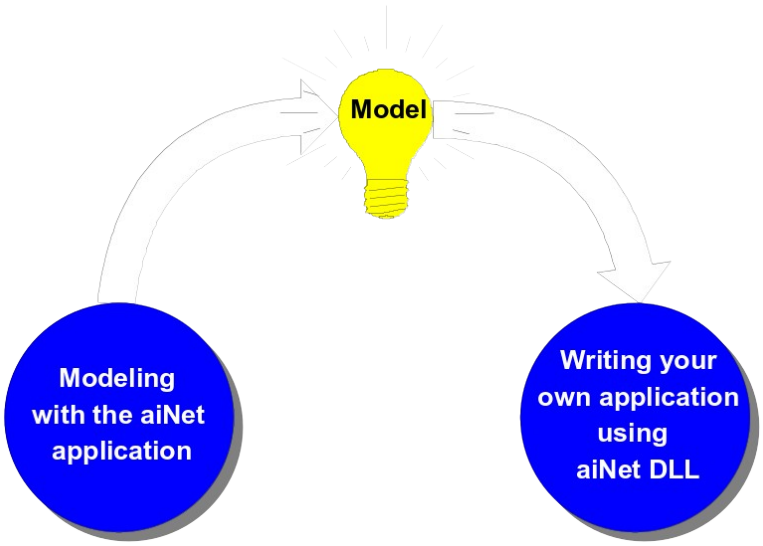

Figure 4.1: How to use the aiNet DLL library

Of course, you may develop your own strategies for using the aiNet DLL.

aiNet DLL functions overview

We have already told that the aiNet DLL library is limited to the prediction task only. These functions should be used in similar fashion to the aiNet application after successful modelling has been completed. The flowchart of actions in the prediction process is shown in Figure 4.2.

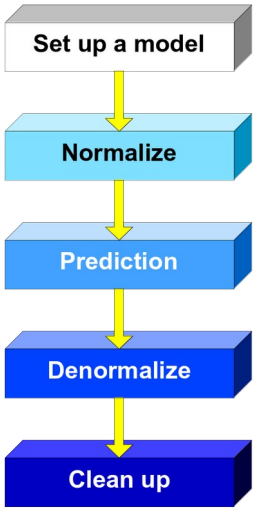

Figure 4.2: Flowchart of typical usage of the aiNet DLL functions

Let us assume that we found a good model with aiNet and now we would like to write an application which will calculate predictions based on the model. We will also assume that we are using C and a compiler which can link Windows based DLLs.

The major steps in the process may be seen from the flowchart and there follows a brief explanation.

First we should allocate memory for the model. The *aiCreateModel* function is used for this purpose. Than data is loaded into the model either directly in the computer code, or simply by reading a CSV model file produced by aiNet. To use a CSV file, use the *aiCreateModelFromCSVFile* instead *aiCreateModel* function. Once the model is defined, it may be normalized using the *aiNormalize* function. Everything is now ready for the prediction to take place. The prediction process is run with the *aiPrediction* function. The prediction may be run repeatedly. When all predictions are complete, it is good practice to denormalize the model, although this is not necessary. *aiDenormalize* function completes this task. Finally, if memory for the model was allocated dynamically, it should be freed using the *aiDeleteModel* function.

aiNet DLL Library and C (C++) compilers

The best way to see how to use the aiNet DLL with a C compiler, is to look at the examples in the \AINET\DLL\C directory. There three examples. All these perform the same task, but in different ways. Study these simple examples and also look at The aiNet DLL Library Reference chapter for more details about each aiXX function used in examples.

Note, although the examples are functionally very similar they produce different results. This is not en error, because each example uses different normalization and penalty settings.

To build examples correctly, AINETXX.LIB needs to be added to the project files. This is a virtual library which holds function signatures needed to link the aiNet DLL library correctly. For 32 bit applications, it is suggested that AINET32.LIB is used. For 16 bit applications, AINET16.LIB must be used. Please see your compiler documentation for more specific details.

Here is an example of project file for Borland C(++), version 4.5

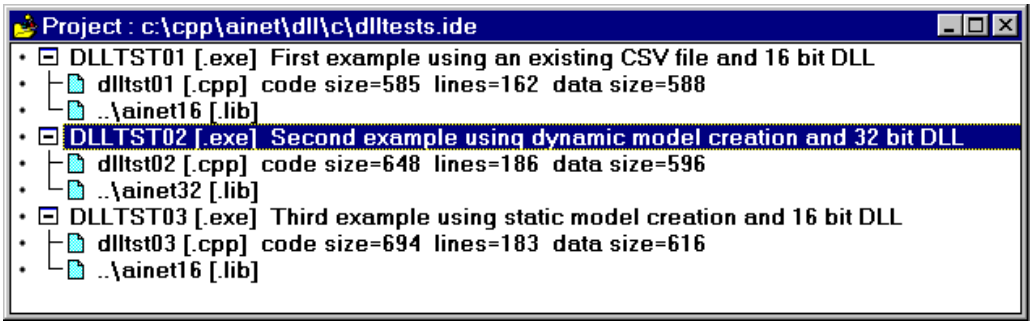

Figure 4.3: Sample of project file for BC++ 4.5

Troubleshooting: If the example will not compile, then verify that the header file AINETDLL.H is included in the project search path. This file may be found in the..AINET\DLL directory. Check also that proper LIB files are added to the project file. Finally, ensure that the correct DLL files are either in the working directory, in the search path, or in WINDOWS\SYSTEM directory.

AINETXX.DLL files will be found will be found in several aiNet subdirectories. Only one is required in the WINDOWS\SYSTEM directory. The others may be deleted.

aiNet DLL Library and Visual Basic

Before using aiNet with Visual Basic, it is recommend that the example projects located in ..AINET\DLL\VB directory are examined. There are two sub-directories named TEST1 and TEST2. Also refer to 'The aiNet DLL Library Reference' chapter, for more details about each aiXX function used in the examples.

The description in The aiNet DLL Library Reference uses the C language syntax. If you are not familiar with it, then the following table may be useful. This table shows the conversion of some variable types from C to Visual Basic. In AINET.BAS, all of the function calls are converted to Visual Basic form. Please, refer to the Visual Basic documentation for more details.

C	Visual Basic	Comment
int	Integer	
BOOL	Integer	
float	Single	
long	Long	
aiModel*	Long	VB does not recognise pointers. A pointer is converted in aiModel to a Long variable which are both 32 bit long.
aiModel		Not accessible in Visual Basic

Table 4.1: C to Visual Basic variable type conversion

To build you applications properly, AINET.BAS must be included in the make (project) files. This file holds all of signatures of an aiNet DLL function and enables calling of the aiNet DLL library. AINET16.DLL must be either in your working directory, on the search path or in WINDOWS\SYSTEM directory.

AINET16.DLL exists in several AINET subdirectories. It is suggested that a single copy is placed in the WINDOWS\SYSTEM directory and the others deleted.

The aiNet DLL Library Reference

This chapter shows how aiNet DLL functions used. Function calls are explained, together with function arguments, return values and error conditions.

The aiModel structure

Before looking at individual functions, it is advisable to examine the structure of aiModel. This structure encapsulates all the variables which are needed for the prediction: it holds the model data and also has some working variables.

There is an important difference between the model organisation in the aiNet application and model organisation in the aiNet DLL. The aiNet DLL will not compute "excluded" variables. Individual variables may not be switched from input to output and vice versa as may be done with the aiNet application. The input and output variables in the aiNet DLL can not be mixed in any order. Input variables come first and the output variables follow. These "drawbacks" are rewarded by greater simplicity and calculation speed, since no extra mapping is needed.

Definition

The *aiModel* structure is defined by:

```
typedef struct aiTagModel{
   float **data;        /* model (vector of model vectors) */
   int nMV;             /* number of model vectors         */
   int nVar;            /* total number of variables       */
   int ni;              /* number of input variables             */
   int* discrete;       /* array of flags for discrete variables */
   aiVector n1;         /* noramlization / denormalization       */
   aiVector n2;         /* ...                                   */
NEW int capacity;       /* data array size                       */
NEW unsigned char flag; /* tells if model vector is excluded     */
} aiModel;
```

There is no need to directly access any of the variables in the *aiModel* structure - the functions do that, but here is an explanation of the purposes of individual variables.

*float **data* is a pointer to pointers (a vector of vectors). It holds the model vector data. It is actually an array of size at least *nMV* (number of model vectors) by exactly *nVar* (number of variables). The data in the *data* field can be accessed using *aiGetVariable* and *aiSetVariable*. NOTE: The *data* variable works in conjunction with the *capacity* variable.

int nMV returns the number model vectors that may be stored in the model. This variable must be set to (it may be set indirectly by the function *aiCreateModel*) exactly the number of model vectors that are required. If more model vectors are specified than the actual number of model vectors, an error condition may result.

int nVar returns the total number of variables in the model. As previously mentioned, the aiNet DLL does not support excluded variables.

int ni returns the number of input variables in the model. It is used as a differentiator between the input and output variables. Variables in a model vector which have smaller C[1] index than is value of *ni* are treated as input variables and variables which have C index greater or equal to value of ni are treated as output variables.

*int *discrete* is an array of size *nVar*. This discrete array indicates if a variable with index *i* is a discrete variable. In this case where the variable *i* is discrete, the array must be set to a non-zero value. Remember, only input variables can have a discrete attribute; this attribute is ignored for output variables.

float n1, n2* are internally used working arrays and they are not important for understanding the aiNet DLL.

NEW *int capacity* is an integer used for internal bookkeeping of the current length of *data* variable. This new variable enables that new model vectors can be added and removed dynamically using

[1] C here stands for C programming language. C index means an index which starts counting at zero element - as it is done in C programming language

a couple of new functions: *aiSetCapacity, aiGetCapacity, aiGetFreeEntries, aiInsertModelVector, aiOverwriteModelVector, aiAppendModelVector, aiDeleteModelVector*. The *capacity* variable works in conjunction with the *data* variable. Initially the array size (*capacity*) equals to *nMV*. This can be later changed by using *aiSetCapacity* function. If the *capacity* is greater than *nMV*, then pointers to model vectors beyond *nMV* are set to *NULL* (they point to nothing). This is very useful, if new model vectors will be added to the model later, dynamically. Totally (`capacity - nMV`) model vectors can be added later. If *capacity* equals to *nMV*, no new model vectors can be added. Capacity will never be smaller than *nMV*. If we force the *capacity* to a smaller value than the current *nMV*, all model vectors between *capacity* and *nMV* will be deleted in order to obtain *nMV* = *capacity*.

NEW *unsigned char* flag* is an array of unsigned chars. The array size is maintained internally and has always the same size as the *data* variable. The *flag* array is used to exclude individual model vectors from the prediction process. This can be very useful in time series analysis where model vectors are sorted according to their time stamp. The *flag* variable is manipulated by *aiExcludeModelVector*, *aiExcludeModelVectorRange* and *aiIsModelVectorExcluded* functions.

aiVector

Besides the *aiModel* structure there is also one definition (typedef) which is used in the aiNet DLL. Here it is it's C definition:

```
typedef float*  aiVector;
```

It is a pointer to an array of floats, which may be treated as a vector.

aiCreateModel

Syntax

aiModel* aiCreateModel(int nModelVectors, int nVariables, int nInpVariables)

Description

This function creates an *aiModel* structure and allocates memory for it's working variables. It allocates as much memory as is required to hold *nMV* model vectors, where each vector has *nVar* variables. All working variables and flags of the aiModel structure are also initialised. The number of variables or number of model vectors may not be changed after this function call has been made. The *aiModel* structure created by this function should be always freed using the function *aiDeleteModel*, so that all allocated memory is properly returned to the system.

Arguments

int nModelVectors is number of model vectors that will be put into the model. This argument determines the value of the *nMV* variable in the *aiModel* structure. The value of *nMV* variable should not be changed after this function has been called.

int nVariables is the total number of variables in the model (input and output). The variable *nVar* in the *aiModel* structure is set to the value held by argument *nVariables*. The value of *nVar* should not be changed after this function has been called.

int nInpVariables is the number of input variables used in the model. The variable *ni* in the *aiModel* structure will be set to the value held by *nInpVariables*.

Return value

The function returns a pointer to the structure *aiModel*. If anything goes wrong during model creation, the function frees all of the allocated memory and returns NULL pointer indicating an error condition.

See also

aiModel, aiDeleteModel, aiCreateModelFromCSVFile

aiCreateModelFromCSVFile

Syntax

aiModel* aiCreateModelFromCSVFile(const char* fileName)

Description

This function creates a model on the basis of a CSV file. The CSV file should be created using aiNet. The function first checks how many model vectors and variables are in the CSV. This is done using the *aiGetCSVFileModelSize* function. The function then sorts the variables according their status. Input variables are mapped to the left, output variables to the right and excluded variables are deleted. This means that if there are excluded variables in the CSV file, the final number of variables in the model will be reduced by the number of excluded variables in the file.

Example: The following model was created using aiNet:

Variable name	**Result1**	**A**	**B**	**C**	**Result2**	**D**
Status	Output	Input	Excluded	Excluded	Output	Input
Discrete	No	Yes	No	No	No	No
1	100	20	15	-10	70	5
...	...	...	...	...	...	...
nMV	30	25	76	-23	44	3

By using the *aiCreateModelFromCSVFile* function the data is redefined as per the following table:

Variable name	A	D	Result1	Result2
Status	Input	Input	Output	Output
Discrete	Yes	No	No	No
1	20	5	100	70
...	...	...	...	...
nMV	25	3	30	44

Variables B and C are missing, because they were excluded. We can also notice that variable D was shifted to left and Result1 was shifted to right. In the CSV file there were 6 variables and in the model, created by the *aiCreateModelFromCSVFile* function, there are 4 variables.

The *aiModel* structure created by this function should be always freed using the *aiDeleteModel* function, so that all of the allocated memory is properly returned to the system.

See aiNet User's Guide, chapter 2.4: aiNet's File Formats for a detailed CSV file format explanation.

Arguments

const char fileName* points to a full path and file name of the CSV file used for model creation.

Return value

The function returns a pointer to structure *aiModel*. If anything goes wrong during model creation the function frees all allocated memory and returns NULL pointer indicating an error condition.

See also

aiModel, aiDeleteModel, aiCreateModel, aiGetCSVFileModelSize

aiDeleteModel

Syntax

int aiDeleteModel(aiModel* model)

Description

This function frees all memory which was allocated during *aiCreateModel* or *aiCreateModelFromCSVFile* function call and returns the freed memory to the system. When the *aiDeleteModel* function finishes, the structure *model* becomes invalid and any further reference to this structure will result in an invalid pointer assignment error.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

Return value

The function returns AIERR_INVALID_POINTER if the *model* argument was bad. If successful, then the function returns AIERR_NO_ERROR.

See also

aiModel, aiCreateModel, aiCreateModelFromCSVFile

aiNormalize

Syntax

int aiNormalize(aiModel model, int method)

Description

Before the *aiPrediction* function can be used on a model, the model must be normalized. Two different methods of normalization may be used: regular and statistical. See the aiNet User's Guide - section 2.2.3 for more detailed explanation of these two methods.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or *aiCreateModelFromCSVFile* function.

int method defines which method to be used during the normalization process. Two different methods of normalization may be used and they are denoted with following constants:

METHOD_REGULAR	for regular normalization
METHOD_STATISTICAL	for statistical normalization

Return value

On success, the *aiNormalize* function returns the AIERR_NO_ERROR constant (which equals to zero). If an error has occurred, than the function returns a negative value. The following errors can occur: AIERR_NO_IO_VARIABLES, AIERR_EMPTY_ROW, AIERR_EMPTY_COLUMN, AIERR_EQUAL_COLUMN.

See also

aiModel, aiPrediction, aiDenormalize, list of error constants

aiDenormalize

Syntax

int aiDenormalize(aiModel model)

Description

The *aiDenormalize* function is an inverse function of the *aiNormalize* function. It initialises the model. Because of the float (4 byte) precision, used in the aiNet model, minor rounding errors may occur.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using *aiCreateModel* or *aiCreateModelFromCSVFile* function.

Return value:

There are no errors produced during denormalization and therefore this function always returns AIERR_NO_ERROR.

See also

aiModel, aiNormalize

aiRegistration

Syntax

int aiRegistration(const char* user, const char* code)

Description

By registering the aiNet DLL, a fixed User Name and User Code will be provided. See 'All About the Registration' in the aiNet User's Manual for further details. By using *aiRegistration,* it will prevent the registration dialogue box from appearing.

Arguments

const char user* is a pointer to an array of characters which represents a registered user name.

const char code* is a pointer to an array of characters which represent a 10 character code string which belongs to the user name.

Return value

The function returns non-zero if the registration was successful and zero if not.

aiPrediction

Syntax

int aiPrediction(aiModel model, aiVector toPredict, float penalty, int method)

Description

The *aiPrediction* function is at the heart of the aiNet DLL. It calculates a prediction based on the *model* argument and on the input part of the *toPredict* argument. Of course, the *penalty* and *method* arguments are also important. The prediction is calculated in an almost identical way as

is done in aiNet itself. The underlying algorithm is the same, differences come from different coding used in C++ language and from the simpler representation of input and output variables in the aiNet DLL. However, the results obtained using the DLL should be identical to those obtained from aiNet. For the mathematical description of the algorithm see the Appendix.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using *aiCreateModel* or *aiCreateModelFromCSVFile* function.

aiVector toPredict it is an array of floats. The input part (first *model->ni* variables) must be set to values for which a prediction (last *model->nVar-model->ni* variables) is required. The size of this array must be equal to (or larger than) the value in *model->nVar* variable. When the *aiPrediction* function is finished and no errors have occurred, then the output part of the *toPredict* vector (last *model->nVar-model->ni* variables) holds the prediction result.

float penalty holds the value of the penalty coefficient. This coefficient plays the major role in the aiNet neural network algorithm. Other sections of the manual describes this in detail. See aiNet User's Guide and Basics About Modelling with the aiNet for more details.

int method selects the penalty method to be used. There are three different methods denoted by following constants: METHOD_DYNAMIC, METHOD_STATIC and METHOD_NEAREST. Note that each method requires a different optimal penalty coefficient value.

Return value

On success, the *aiPrediction* function returns the AIERR_NO_ERROR constant. If an error occurs, the input and output part of the *aiVector toPredict* argument holds invalid values and function returns with one of the following constants:

AIERR_PENALTY_TOO_SMALL, AIERR_PENALTY_ZERO, AIERR_NO_IO_VARIABLES.

See also

aiPredictionEx, aiExcludeModelVector, aiExcludeModelVectorRange, aiModel, aiVector, list of error constants

aiGetVariable

Syntax

float aiGetVariable(aiModel* model, int mv, int v)

Description

The *aiGetVariable* function is used to get individual values in the *model* at random positions. The position is limited to the range [1 ... *nMV*] for the model vector index and to the range [1 ... *nVar*] for the variable index. If the model was normalized before this function call, then the returned value holds the normalized value.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

int mv is an index of a model vector in the *model* structure, where the variable required is located. The first model vector has an index of 1. This means that the valid index range is within [1 ... *nMV*] and not [0 ... *nMV*-1] as it would be in classic C-notation.

int v is the index of a variable in the selected model vector. The first variable has an index of 1, meaning that the valid index range is [1 ... nVar].

Return value

The function returns the value at selected position., if the position is valid. If the position is invalid, the function returns the -MAXFLOAT constant, which indicates an error condition.

Important note:

This function does not work well if it is called from a Visual Basic application. For Visual Basic applications, an alternative function *aiGetVariableVB*, is provided.

See also

aiGetVariableVB, aiModel, aiSetVariable

aiGetVariableVB

Syntax

int aiGetVariableVB(aiModel* model, int mv, int v, float* value)

Description

The *aiGetVariableVB* function is used to get individual values in the *model* at random positions. The position is limited to the range [1 ... *nMV*] for the model vector index and to the range [1 ... *nVar*] for the variable index. If the model was normalized before this function call was made then the returned value holds the normalized value.

Use this function with Visual Basic applications.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using *aiCreateModel* or *aiCreateModelFromCSVFile* function.

int mv is an index of a model vector in the *mode,* where the variable required is located. The first model vector has an index 1. This means that the valid index range is [1 ... *nMV*] and not [0 ... *nMV*-1] as it would be in classic C-notation.

int v is an index of a variable in the selected model vector. The first variable has an index 1 and this means that the valid index range is [1 ... *nVar*].

float value is a value which will be read from the selected position and passed back to the variable in the calling function.

Return value

If the position is a valid, the function returns the AIERR_NO_ERROR constant. If the position is invalid than the function returns the AIERR_INVALID_INDEX constant.

See also

aiModel, aiGetVariable, aiSetVariable

aiSetVariable

Syntax

int aiSetVariable(aiModel* model, int mv, int v, float value)

Description

The *aiSetVariable* function is used to setup the *model* before the model is normalized. It will set the value in the *value* argument to the position selected by *mv* and *v* arguments. The position must be in the range [1 ... *nMV*] for the model vector index and in the range [1 ... *nVar*] for the variable index. If the position is invalid, the function ends and returns an error condition. Note this function may be used at any time, but it will be invalid for normalized models.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

int mv is an index of a model vector in the *model,* where the variable required is located. The first model vector has an index 1. This means that the valid index range is [1 ... *nMV*] and not [0 ... *nMV*-1] as it would be in classic C-notation.

int v is index of a variable in the selected model vector. The first variable has an index 1 and this means that the valid index range is [1 ... *nVar*].

float value is a value which will be set to the selected position.

Return value

If the position is valid, the function returns the AIERR_NO_ERROR constant. If the position is invalid, then the function returns the AIERR_INVALID_INDEX constant.

See also

aiModel, aiGetVariable

aiGetCSVFileModelSize

Syntax

DWORD aiGetCSVFileModelSize(const char* fileName)

Description

This function opens the *fileName* CSV file and reads number of model vectors and number of variables in the file. The result is packed into a DWORD (unsigned long).

See aiNet User's Guide, chapter 2.4: aiNet's File Formats for a detailed CSV file format description.

Arguments

const char fileName* points to a full path and file name of the CSV file.

Return value

This function returns a DWORD. The DWORD is composed from two WORDs. The Hi WORD holds number of model vectors and the low WORD holds number of variables. If an error occurs, then the function returns either the AIERR_CSV_OPEN or the AIERR_CSV_READ constant.

The following code may be used to split the returned DWORD into two WORDs or ints:

```
int mv, var;
DWORD result = aiGetCSVFileModelSize(fileName);
if( ((int)result) == AIERR_CSV_OPEN || ((int)result) == AIERR_CSV_READ )
  { report error here ... }
mv  = (int)LOWORD(result);
var = (int)HIWORD(result);
```

aiGetVersion

Syntax

int aiGetVersion(void)

Description

The aiGetVersion function returns the version of the aiNet dynamic link library

Return value

The function returns an int which holds the major and minor version number of the aiNet DLL. The return value is defined as: 100*major+minor.

Example:

The current version of the aiNet library is 1.20 - major version is 1 and minor version is 20. The return value in this case is 100*1+20 = 120.

aiGetNumberOfVariables

Syntax

int aiGetNumberOfVariables(aiModel* model)

Description

This is a simple function which retrieves number of variables used in the *model*. The number of variables in the *model* is set during the *aiCreateModel* function or during the *aiCreateModelFromCSVFile* function call.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

ReturnValue

This is the number of variables in the *model* structure. If the pointer *model* was invalid then the function returns the AIERR_INVALID_POINTER constant.

See also

aiModel, aiCreateModel, aiCreateModelFromCSVFile

aiGetNumberOfModelVectors

Syntax

```
int aiGetNumberOfModelVectors(aiModel* model)
```

Description

This function returns the number of model vectors used in the *model*. The number of model vectors was set during the *aiCreateModel* function or during the *aiCreateModelFromCSVFile* function call.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* function, or using the *aiCreateModelFromCSVFile* function.

ReturnValue

The function returns the number of model vectors in the *model* structure. If the pointer *model* was invalid, then the function returns the AIERR_INVALID_POINTER constant.

See also

aiModel, aiCreateModel, aiCreateModelFromCSVFile

aiGetNumberOfInputVariables

Syntax

```
int aiGetNumberOfInputVariables(aiModel* model)
```

Description

This function returns the number of input variables used in the *model*. The number of input variables is set by the *aiCreateModel* function or by the *aiCreateModelFromCSVFile* function call.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

ReturnValue

The function returns number of input variables in the *model* structure. If the pointer *model* was invalid, then the function returns the AIERR_INVALID_POINTER constant.

See also

aiModel, aiCreateModel, aiCreateModelFromCSVFile

aiSetDiscreteFlag

Syntax

int aiSetDiscreteFlag(aiModel* model, int v, BOOL f)

Description

The *aiSetDicreteFlag* function is used to set or clear flags in the *discrete* variable of the *model*. See Basics About Modelling with the aiNet, Chapter 2 for details about variable types.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

int v is a field index of the *discrete* variable in the *model*. The valid range for this index is [1...*model->ni*].

ReturnValue

The function returns one of the three possible constants: AIERR_NO_ERROR indicates a success, AIERR_INVALID_INDEX and AIERR_INVALID_POINTER indicate failure.

See also

aiModel, aiGetDiscreteFlag

aiGetDiscreteFlag

Syntax

int aiGetDiscreteFlag(aiModel* model, int i)

Description

This function returns the discrete status of selected variable.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

int v is a field index of the *discrete* variable in the *model*. The valid range for this index is [1...*model->ni*].

ReturnValue

On successful completion, the function returns zero if the variable *v* is not a discrete variable and 1 if the variable *v* is a discrete variable. In the event of error it returns either an AIERR_INVALID_INDEX or an AIERR_INVALID_POINTER constant.

See also

aiModel, aiSetDiscreteFlag

aiSetCapacity NEW

Syntax

int aiSetCapacity(aiModel* model, int newCapacity)

Description

This function resizes the *data* and the flag fields of an *aiModel* structure. If the new capacity is smaller than the current one, which means that the model will be reduced, than the redundant model vectors will be deleted automatically. (`model->capacity - newCapacity` model vectors will be deleted).

If the new capacity is greater than current one, new free entries will be added to the *data* field of an *aiModel* structure. All new entries will point to NULL – to non-existing model vectors. By ensuring new free entries, new model vectors can be added to the model using *aiInsertModelVector, aiOverwriteModelVector, aiAppendModelVector* function.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

int newCapacity is a new value of the total numbers of model vectors that can be put in the model.

ReturnValue

On successful completion, the function returns value of new capacity. In the event of error it returns AIERR_MEMORY_ALLOCATION constant. NOTE: All error constants are negative integers.

See also

aiModel, aiGetCapacity, aiGetFreeEntries

aiGetCapacity NEW

Syntax

int aiGetCapacity(aiModel* model)

Description

This function returns current size of the *data* array of an *aiModel* structure. NOTE: This does not equal to number of model vectors *nMV* in the *aiModel* structure. If you would like to get number of model vectors in the structure than you should use *aiGetNumberOfModelVectors* function.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

ReturnValue

On successful completion, the function returns current value the capacity field.

See also

aiModel, aiSetCapacity, aiGetFreeEntries

aiGetFreeEntries NEW

Syntax

int aiGetFreeEntries(aiModel* model)

Description

This function returns a number of model vectors that can be added to a model. The return value is actually the difference between *capacity* and *nMV*. You can use this function to see how many model vectors can you add to a model without changing the current *capacity*.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

ReturnValue

It returns a number of free entries – number of model vectors that can be added to a *model* at current *capacity*.

See also

aiModel, aiSetCapacity, aiGetCapacity, aiGetNumberOfModelVectors

aiInsertModelVector NEW

Syntax

int aiInsertModelVector(aiModel* model, int index, aiVector newModelVector)

Description

This function inserts a new model vector to position defined by the *index* argument. The call to this function can be successful only if there is at least one free entry left in a model. (The *capacity* must be greater than *nMV*.) New model vector is inserted at *index* position – all model vectors above the *index* position are shifted for one place toward end.

If *index* is greater than *nMV* and smaller or equal to *capacity* (*index* points to the NULL area), than the new model vector is appended at the position next to the last valid model vector. The actual *index* position is ignored – it is set internally to *nMV+1*.

If index refers beyond the capacity, the function returns AIERR_INVALID_INDEX.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

int index is a position where the new model vector shall be inserted. If *index* is greater than *nMV*, than *index* is set to *nMV+1* and *nMV* is incremented by one.

aiVector newModelVector is a vector (an array) of floats which represent model vector values that will be copied to model at *index* position. The *newModelVector* argument must have at least *nVar* elements. The caller is responsible to pass the correct size of *newModelVector*. If *newModelVector* is NULL pointer, than a zero model vector will be inserted into a model - all values will have value 0.

ReturnValue

On successful completion, the function returns a positive value – the actual position of inserted model vector. On failure, it returns AIERR_NO_FREE_ENTRIES if there are no free entries left or AIERR_MEMORY_ALLOCATION if memory for new model vector could not be allocated or AIERR_INVALID_INDEX if *index* is invalid.

See also

aiGetFreeEntries, aiAppendModelVector, aiOverwriteModelVector, aiDeleteModelVector.

aiOverwriteModelVector NEW

Syntax

int aiOverwriteModelVector(aiModel* model, int index, aiVector newModelVector)

Description

This function overwrites an existing model vector at position defined by the *index* argument. If index is greater than *nMV* (it points to an empty area) then model vector will be appended at the

first available position, which is exactly at *nMV+1*. The call to this function can be successful only if *index* points to an existing model vector or if *index* points to a free area and there is at least one free entry left in a model. (In the later case, *capacity* must be greater than *nMV*.)

If index refers beyond the *capacity*, the function returns AIERR_INVALID_INDEX.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

int index is a position where the new model vector shall overwrite the old one. If *index* is greater than *nMV*, than *index* is set to *nMV+1* and *nMV* is incremented by one.

aiVector newModelVector is a vector (an array) of floats which represent model vector values that will overwrite an existing model vector at *index* position. The *newModelVector* argument must have at least *nVar* elements. The caller is responsible to pass the correct size of *newModelVector*. If *newModelVector* is NULL pointer, than a zero model vector will be inserted into a model - all values will have value 0.

ReturnValue

On successful completion, the function returns a positive value – the actual position of inserted model vector. On failure, it returns AIERR_NO_FREE_ENTRIES if there are no free entries left or AIERR_MEMORY_ALLOCATION if memory for new model vector could not be allocated or AIERR_INVALID_INDEX if *index* is invalid.

See also

aiGetFreeEntries, aiAppendModelVector, aiInsertModelVector, aiDeleteModelVector.

aiAppendModelVector NEW

Syntax

int aiAppendModelVector(aiModel* model, int index, aiVector newModelVector)

Description

This function appends the newModelVector at the first available position, which is exactly at *nMV+1*. The call to this function can be successful only if there is at least one free entry left in a model. (In the later case, *capacity* must be greater than *nMV*.) After the function completion, *nMV* is incremented by one.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

aiVector newModelVector is a vector (an array) of floats which represent model vector values that will overwrite an existing model vector at *index* position. The *newModelVector* argument must have at least *nVar* elements. The caller is responsible to pass the correct size of *newModelVector*. If *newModelVector* is NULL pointer, than a zero model vector will be inserted into a model - all values will have value 0.

ReturnValue

On successful completion, the function returns a positive value – the actual position of inserted model vector. On failure, it returns AIERR_NO_FREE_ENTRIES if there are no free entries left or AIERR_MEMORY_ALLOCATION if memory for new model vector could not be allocated.

See also

aiGetFreeEntries, aiOverwriteModelVector, aiInsertModelVector, aiDeleteModelVector.

aiDeleteModelVector NEW

Syntax

int aiDeleteModelVector(aiModel* model, int index)

Description

This function deletes the model vector at the position defined by *index* argument. Here, *index* must point to an existing model vector. The model vector is removed from the model and memory allocated by this model vector is freed. All model vectors in the range of [*index+1 ... nMV*] are shifted by one place toward beginning. After the function completion, *nMV* is decremented by one.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

int index points to a model vector which will be deleted and removed from the *model*. *Index* must be a valid index - in the range [*1...nMV*].

ReturnValue

On successful completion, the function returns AIERR_NO_ERROR if function call was successful. If index was not valid it returns AIERR_INVALID_INDEX

See also

aiAppendModelVector, aiInsertModelVector, aiOverwriteModelVector, aiExcludeModelVector, aiExcludeModelVectorRange

aiPredictionEx NEW

Syntax

int aiPredictionEx(aiModel model, aiVector toPredict, float penalty, int method, int* list,
int listSize, BOOL mostInfluent)

Description

This function is an extension of the *aiPrediction* function. It has exactly the same behaviour as the aiPrediction function. The aiPredictionEx function additionally returns the indexes of model

vectors in the *model*, which had the greatest (smallest) influence on the result of the prediction. In many cases this can be very valuable information. The model vector indexes are stored in the *list* argument, sorted by their influence. Theoretically, the *list* argument can have up to *nMV* elements. When this is the case, list will hold indexes of all model vectors any you will be able to judge every individual model vector. However, in this case prediction will take much more time to finish the calculation since it has to maintain an internal sorted list of intermediate results - which can be time consuming if there is a lot of model vectors in the *model*.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using *aiCreateModel* or *aiCreateModelFromCSVFile* function.

aiVector toPredict it is an array of floats. The input part (first *model->ni* variables) must be set to values for which a prediction (last *model->nVar-model->ni* variables) is required. The size of this array must be equal to (or larger than) the value in *model->nVar* variable. When the *aiPrediction* function is finished and no errors have occurred, then the output part of the *toPredict* vector (last *model->nVar-model->ni* variables) holds the prediction result.

float penalty holds the value of the penalty coefficient. This coefficient plays the major role in the aiNet neural network algorithm. Other sections of the manual describe this in detail. See aiNet User's Guide and Basics About Modelling with the aiNet for more details.

int method selects the penalty method to be used. There are three different methods denoted by following constants: METHOD_DYNAMIC, METHOD_STATIC and METHOD_NEAREST. Note that each method requires a different optimal penalty coefficient value.

int list* is an array of integers where indexes of model vectors that had the greatest influence on the result of prediction are stored. Indexes in the list are sorted according to the influence. A model vector at list[0] had the greatest influence, a model vector at list[1] had the second greatest influence, etc.

int listSize tells how many indexes will be stored in the list argument by the *aiPredictionEx* function. The user must take care that *listSize* does not exceed the maximal number of elements that can be stored in the *list* argument.

BOOL mostInfluent tells how the elements in the *list* are sorted. If *mostInfluent* is *TRUE* then most influent model vectors are stored in the *list* in their ascending order. If *mostInfluent* is *FALSE* then least influent model vectors are stored in the *list* in their descending order.

Return value

On success, the *aiPrediction* function returns the AIERR_NO_ERROR constant. If an error occurs, the input and output part of the *aiVector toPredict* argument holds invalid values and function returns with one of the following constants:

AIERR_PENALTY_TOO_SMALL, AIERR_PENALTY_ZERO, AIERR_NO_IO_VARIABLES.

See also

aiPrediction, aiExcludeModelVector, aiExcludeModelVectorRange, aiModel, aiVector, list of error constants

aiExcludeModelVector NEW

Syntax

int aiExcludeModelVector(aiModel* model, int index, BOOL exclude)

Description

This function excludes/includes the model vector at the position defined by *index* argument. Here, *index* must point to an existing model vector. The model vector is excluded from the model if the *exclude* argument is *TRUE* (non-zero) and included back into the model if the exclude argument is set to *FALSE* (zero). This function does not delete the model vector. Exclusion of model vector effects only aiPrediction and aiPredictionEx functions. All excluded model vectors are skipped in the prediction process. Initially - during *model* construction, all model vectors are marked as included.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

int index points to a model vector which will be excluded/included from/in the *model*. *Index* must be a valid index - in the range [*1…nMV*].

BOOL exclude is a flag which tells whether a model vector will be excluded or included. If *exclude* is set to TRUE than model vector will be excluded. If *exclude* is set to FALSE than model vector will be included.

ReturnValue

On successful completion, the function returns AIERR_NO_ERROR if function call was successful. If index was not valid it returns AIERR_INVALID_INDEX

See also

aiExcludeModelVectorRange, aiDeleteModelVector

aiExcludeModelVectorRange NEW

Syntax

int aiExcludeModelVectorRange(aiModel* model, int start, int end, BOOL exclude)

Description

This function excludes/includes model vector in the range defined by the *start* and the *end* argument. Selected range must be a sub-range of [1 … nMV]. The model vectors are excluded from the model if the *exclude* argument is *TRUE* (non-zero) and included back into the model if the exclude argument is set to *FALSE* (zero). This function does not delete the model vectors. Exclusion of model vector effects only aiPrediction and aiPredictionEx functions. All excluded model vectors are skipped in the prediction process. Initially - during *model* construction, all model vectors are marked as included.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

int start points to the first model vector in the range that will be excluded/included from/in the *model*. *start* must be a valid index - in the range [*1...nMV*] and *start* must be smaller than *end*.

int end points to the last model vector in the range that will be excluded/included from/in the *model*. *end* must be a valid index - in the range [*1...nMV*] and *start* must be smaller than *end*.

BOOL exclude is a flag which tells whether a model vector will be excluded or included. If *exclude* is set to TRUE than model vector will be excluded. If *exclude* is set to FALSE than model vector will be included.

ReturnValue

On successful completion, the function returns AIERR_NO_ERROR if function call was successful. If index was not valid it returns AIERR_INVALID_INDEX

See also

aiExcludeModelVector, aiDeleteModelVector

aiIsModelVectorExcluded NEW

Syntax

BOOL aiIsModelVectorExcluded(aiModel* model, int index)

Description

This function tells whether a model vector selected by the *index* argument is excluded from the *model*. The *index* argument must be a valid index in the range of [*1...nMV*].

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

int index points to the model vector of our interest. It must be in the range [*1...nMV*].

ReturnValue

It returns TRUE (1) if selected model vector is excluded from the model and FALSE (0) if model vector is included in the model. On error - invalid index, the result is unpredictable.

See also

aiExcludeModelVector, aiExcludeModelVectorRange

aiSaveCSVFile NEW

Syntax

int aiSaveCSVFile(aiModel* model, const char* fileName)

Description

This function saves the current model to a file in a CSV format. This format is fully compatible with the CSV format used by the aiNet application versions 1.24 and later. Prior version of the aiNet application can not read files produced by this function. You should denormalize the model before you call this function.

Arguments

aiModel model* is a pointer to an *aiModel* structure. This structure has to be created using the *aiCreateModel* or the *aiCreateModelFromCSVFile* function.

const char fileName* is the name of the text file produced by this function. If file does not exist, it will be created, if it already exists, it will be overwritten.

ReturnValue

On successful completion, the function returns AIERR_NO_ERROR.

See also

aiCreateModelFromCSVFile, aiDenormalize

List of error constants

Here is the complete list of error constants used in the aiNet DLL. All constants have a negative value except AIERR_NO_ERROR constant which equals to zero and indicates that no errors have occurred. Error constants are listed in following table:

Error Name	Value	Description
AIERR_NO_ERROR	0	Indicates OK - NO ERROR condition
AIERR_PENALTY_ZERO	-1	Penalty coefficient was set to zero (or negative) value, which is invalid.
AIERR_NO_IO_VARIABLES	-2	There is no input or output variables in the model. At least one input and one output is required.
AIERR_PENALTY_TOO_SMALL	-3	The specified penalty coefficient was too small and calculation of prediction was completed. Specify a larger value.
AIERR_EMPTY_ROW	-4	There is an empty row (model vector) in the model. This is not allowed in the aiNet DLL. (It is allowed in the aiNet application.)
AIERR_EMPTY_COLUMN	-5	There is an empty column (variable) in the model. At least two different values for a given variable must be

		in the model.
AIERR_EQUAL_COLUMN	-6	All values for a given variable are the same. There must be at least two different values for a given variable in the model.
AIERR_CSV_OPEN	-7	A CSV file could not be opened.
AIERR_CSV_READ	-8	A CSV file was opened successfully but an error occurred during reading.
AIERR_MEMORY_ALLOCATION	-9	Memory was not allocated properly or it was not allocated at all.
AIERR_INVALID_POINTER	-10	The specified pointer is invalid.
AIERR_INVALID_INDEX	-11	Variable or model vector index is out of range.
AIERR_NO_FREE_ENTRY	-12	There are no free entries for model vectors.

Table 4.2: Error Constants in the aiNet DLL library.