Which

1.0

Generated by Doxygen 1.5.5

Wed Feb 20 13:52:40 2008

Contents

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

Class Documentation

2.1 Data Class Reference

`#include <Data.h>`

Public Member Functions

- Data ()
- ~Data ()
- void read (std::string fName)
- void discretizeEqInt (int bins, Data *combine)
- void discretizeEqFreq (int bins, Data *combine=NULL)
- bool subsample (unsigned int desClass, float per)
- unsigned int microsample (unsigned int amount)
- void normalizeAttribute (int attIndex)
- Data * clone ()
- bool cover (Rule *rule)
- int compareListItems (ListItem l1, ListItem l2)
- void calcLift ()
- void calcPDPFEst (unsigned int LOC)
- void calcProbSupt ()
- float getLift ()
- int getTotLOC ()
- std::vector< int > getLOCs ()
- std::vector< std::vector< InstanceElement * > * > getInstanceSet ()
- unsigned int getNumAtts ()
- unsigned int getNumClasses ()
- unsigned int getClassIndex (std::vector< InstanceElement * > *instance)
- unsigned int getNumAttVals (std::string att)
- std::string getAttName (int index)
- unsigned int getAttIndex (std::string name)
- unsigned int getAttValIndex (std::string attName, std::string valName)
- std::string getAttValName (std::string att, int index)
- std::string getClassName (int index)
- std::vector< int > getClassFreqs ()

- const std::vector< std::vector< int * > > * getFrequencyTable ()
- void printAttributes ()
- void printDataSet (std::ostream &stream)
- void printClassDist ()
- void printInstance (int inst)
- void printFrequencyTable (std::ostream &stream)

Protected Member Functions

- void processAttribute (std::string line)
- void processInstance (std::string line)
- std::string preprocessString (std::string line)
- int find (std::string att, std::vector< std::string > &l)

2.1.1 Detailed Description

This class represents a data file. It facilitates the structures necessary to easily get infomormation from the data.

2.1.2 Constructor & Destructor Documentation

2.1.2.1 Data::Data ()

Empty Constructor.

2.1.2.2 Data::~Data ()

Destructor.

2.1.3 Member Function Documentation

2.1.3.1 void Data::read (std::string *fName*)

Reads in the training ARFF file and creates the Data instance.

Parameters:

fName The name of the file to read in.

Returns:

The file stream.

2.1.3.2 void Data::discretizeEqInt (int *bins*, Data * *combine*)

This method will discretize the attributes that are continuous using an equal interval discretization method.

Parameters:

bins The number of bins to use.

combine Another data set to combine with this one in the discretization.

2.1.3.3 void Data::discretizeEqFreq (int *bins*, Data * *combine* = NULL)

This method will discretize the attribtues that are continous using an equal frequency discretization method.

Parameters:

bins The number of bins to use.

combine Another data set to combine with this one in the discretization. If this is null, this is ignored.

2.1.3.4 bool Data::subsample (unsigned int *desClass*, float *per*)

This method will subsample the data. That is, remove instances of data that are not the desired class until the percentage of the desired class in the entire data set is met.

Parameters:

desClass The index of the desired class.

per The desired percent. If this is smaller than the percent makeup already, this method does nothing.

Returns:

true if the set has been altered, false otherwise.

2.1.3.5 unsigned int Data::microsample (unsigned int *amount*)

This method will microsample that data. This involves having an equal distribution of all classes and a total number of each class being equal to amount.

Parameters:

amount The number of each class to be left in the data set.

Returns:

The actual number of each class left in the data set. If amount > size(class) than it will only remove from other classes.

2.1.3.6 void Data::normalizeAttribute (int *attIndex*)

This method will normalize an attribute so that each value is between 0 and 1 and the greatest attribute is equal to 1.

Parameters:

attIndex The index of the attribute to normalize.

2.1.3.7 Data * Data::clone ()

Creates a copy of the Data with the attributes and instance information.

Returns:

The copied Data.

2.1.3.8 bool Data::cover (Rule * *rule*)

This method will remove all instances of data from the data set that are covered by a given rule.

Parameters:

rule The rule to check coverage.

Returns:

true if the set was altered, false otherwise.

2.1.3.9 int Data::compareListItems (ListItem *l1*, ListItem *l2*)

Compares two ListItems.

Parameters:

l1 The first ListItem.

l2 The second Listitem.

Returns:

0 if l1 = l2, -1 if l1 < l2, or 1 if l1 > l2.

2.1.3.10 void Data::calcLift ()

Calculates the base lift of the data.

2.1.3.11 void Data::calcPDPFEst (unsigned int *LOC*)

Calulates the base infomation needed for Effort scoring.

Parameters:

LOC The attribue that is the lines of code.

2.1.3.12 void Data::calcProbSupt ()

Calculates the frequency counts of each attribute-value pair. Assumes all data is discrete. Assumes only 2 ordered classes.(Best is second class)

2.1.3.13 float Data::getLift ()

Gets the base lift of the data.

Returns:

The base lift.

2.1.3.14 int Data::getTotLOC ()

Gets the total lines of code in this data instance.

Returns:

The total lines of code.

2.1.3.15 vector< int > Data::getLOCs ()

Gets the lines of code per instance.

Returns:

A vector containing the lines of code per instance.

2.1.3.16 vector< vector< InstanceElement * > * > Data::getInstanceSet ()

Gets the instance set.

Returns:

The instance set.

2.1.3.17 unsigned int Data::getNumAtts ()

Gets the number of attributes.

Returns:

The number of attributes.

2.1.3.18 unsigned int Data::getNumClasses ()

Gets the number of class values.

Returns:

The number of class values.

2.1.3.19 unsigned int Data::getClassIndex (std::vector< InstanceElement * > * *instance*)

Gets the class index for a given instance.

Parameters:

An instance of data.

Returns:

The class index.

2.1.3.20 unsigned int Data::getNumAttVals (std::string *att*)

Gets the number of values for a given attribute.

Parameters:

att The attribute.

Returns:

the number of values for att.

2.1.3.21 string Data::getAttName (int *index*)

Gets the attribute name of the index'th attribute.

Parameters:

index The name to return.

Returns:

The name of the attribute at index.

2.1.3.22 unsigned int Data::getAttIndex (std::string *name*)

Gets the index of an attribute if the string sent in matches it.

Parameters:

name The name of the attribute to find the index of.

Returns:

The index if found, number of attributes + 1 otherwise.

2.1.3.23 unsigned int Data::getAttValIndex (std::string *attName*, std::string *valName*)

Gets the index of an attribute value if the string sent in matches it.

Parameters:

attName The name of the attribute.

valName The name of the attribute value to match.

Returns:

The index if found, number of attribute values + 1 otherwise.

2.1.3.24 std::string Data::getAttValName (std::string *att*, int *index*)

Gets the name of the attribute value at the index'th value.

Parameters:

att The name of the attribute.

index the value to get.

Returns:

The name of the attribute value at the index.

2.1.3.25 string Data::getClassName (int *index*)

Gets the class name at the index'th location.

Parameters:

index The index of the class to get.

Returns:

The class name in string form.

2.1.3.26 vector< int > Data::getClassFreqs ()

Gets the class frequency vector.

Returns:

The class frequency vector.

2.1.3.27 const vector< vector< int * > > * Data::getFrequencyTable ()

Gets the frequency count table for best^2/(best+rest)

Returns:

A jagged array with each 2-dimensinal access containing a length two array with the first element being the rest count and the second element being the best count of this attribute-value pair.

2.1.3.28 void Data::printAttributes ()

This method will print the attributes.

2.1.3.29 void Data::printDataSet (std::ostream & *stream*)

This method will print the data set.

2.1.3.30 void Data::printClassDist ()

This method prints the class names and frequencies.

2.1.3.31 void Data::printInstance (int *inst*)

This method will print one instance of the data set.

Parameters:

inst The instance number to print.

2.1.3.32 void Data::printFrequencyTable (std::ostream & *stream*)

This method will print all of the attribute value best and rest frequencies.

Parameters:

stream The stream to print to.

2.1.3.33 void Data::processAttribute (std::string *line*) [protected]

Processes a string of text and converts that to a new attribute with values in the mAtts and mAttVals lists.

Parameters:

line The line of text to process.

2.1.3.34 void Data::processInstance (std::string *line*) [protected]

Processes a string of text and converts that to a new instance of a data set. Inserts that instance into the mInstances list.

Parameters:

line The line of text to convert.

2.1.3.35 std::string Data::preprocessString (std::string *line*) [protected]

Removes any trailing whitespace from a line. Makes every letter lower case. This allows for easier matching in later stages of the program.

Parameters:

line A line of text.

Returns:

A processed line of text.

2.1.3.36 int Data::find (std::string *att*, std::vector< std::string > & *l*) [protected]

Attempts to find a string in a list of strings.

Parameters:

att The string to find.

l The list to search.

Returns:

The index of att in l. If it is not found, returns -1.

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

- My Documents/Zach/School/Research/Which/which/Data.h
- My Documents/Zach/School/Research/Which/which/Data.cpp

2.2 DisjunctionSet Struct Reference

`#include <Rule.h>`

2.2.1 Detailed Description

Represents a set of disjunctions for a given attribute.

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

- My Documents/Zach/School/Research/Which/which/Rule.h

2.3 Rule Class Reference

`#include <Rule.h>`

Collaboration diagram for Rule:

Public Member Functions

- Rule ()
- Rule (int attribute, int value, Data *d, RULE_TYPE type, float(*scoreFnc)(Rule *)=NULL)
- ~Rule ()
- void * getComponent ()
- void setComponent (void *com)
- std::vector< DisjunctionSet * > * getRuleSet ()
- void setData (Data *d)
- void setWeights (float alpha, float beta, float gamma)
- float getScore ()
- float getSupport ()
- float getPD ()
- float getPF ()
- float getEffort ()
- int getSize ()
- Rule * clone ()
- bool createFromFile (std::string fName, Data *d, RULE_TYPE type, float(*scoreFcn)(Rule *)=NULL)
- float compare (Rule *r)
- bool isSatisfied (std::vector< InstanceElement * > *instance)
- void score ()
- int findAttribute (int attribute)
- int findAttributeValue (int attribute, int value)
- bool hasComponent (int attribute, int value)
- bool isEqualTo (Rule *r)
- bool addComponent (int attribute, int value)
- Rule * combine (Rule *r)
- Rule * backSelect ()
- void printGotWant (std::ostream &stream, Data *eData)
- void print (std::ostream &stream)
- void printRule (std::ostream &stream)

2.3.1 Detailed Description

Represents a Rule for Which. Contains a series of attributes and the ranges they can have for a rule to fire. Rules in Which are a series of conjunctions of disjunctions. EG: (a = 3 + a = 2)(b = 9 + b = 5)(...)

2.3.2 Constructor & Destructor Documentation

2.3.2.1 Rule::Rule ()

Creates an empty Rule. After this constructer the Rule will have no attributes to fire on.

2.3.2.2 Rule::Rule (int *attribute*, int *value*, Data * *d*, RULE_TYPE *type*, float(*)(Rule *) *scoreFnc* = NULL)

Creates a Rule with one conjunction.(if attribute = attributeValue then...)

Parameters:

attribute The numerical index of the attribute in the data set.

value The numerical index of the attribute's value in the data set.

d A pointer to the data set to score this Rule with.

type The type of Rule this is.

the method to use to score this rule. Only needed for type = SPECIAL

2.3.2.3 Rule::~Rule ()

Destructor.

2.3.3 Member Function Documentation

2.3.3.1 void * Rule::getComponent ()

Returns a pointer to the user defined component.

Returns:

The pointer.

2.3.3.2 void Rule::setComponent (void * *com*)

Allows the user to create a class or struct that gets coupled with a Rule. Using this method allows the user to set a component of the Rule.

Parameters:

com A pointer to the component in memory.

2.3.3.3 vector< DisjunctionSet * > * Rule::getRuleSet ()

Returns the Rule in toe form of a jagged array of integers. See DisjuctionSet to see how to use this array.

Returns:

A pointer to the Rule in a jagged array from.

2.3.3.4 void Rule::setData (Data * *d*)

Allows the user to dynamically change the Data this Rule is evaluated on.

Parameters:

d The new Data.

2.3.3.5 void Rule::setWeights (float *alpha*, float *beta*, float *gamma*)

Allows the user to alter how the standard metrics of PD, PF, and Effort are weighted in any pre-defined scoring method that uses the above metrics.

Parameters:

alpha The weight of PD.

beta The weight of PF.

gamma The weight of Effort.

2.3.3.6 float Rule::getScore ()

Gets the current score of a Rule.

Returns:

The score.

2.3.3.7 float Rule::getSupport ()

Gets the current support of a Rule.

Returns:

The support.

2.3.3.8 float Rule::getPD ()

Gets the PD of a Rule. This value is defined for all pre-defined scoring types.

Returns:

The PD.

2.3.3.9 float Rule::getPF ()

Gets the PF of a Rule. This value is defined for all pre-defined scoring types.

Returns:

The PF.

2.3.3.10 float Rule::getEffort ()

Gets the effort of a Rule. This value is defined for all pre-defined scoring types.

Returns:

The effort.

2.3.3.11 int Rule::getSize ()

Gets the size of the Rule in terms of the sum of conjunctions and disjunctions. For instance: a=[1 OR 4 OR 5] AND b = [2 OR 5] has a size of 5

Returns:

The size.

2.3.3.12 Rule * Rule::clone ()

Creates a cloned version of this Rule. The new Rule is completely seperate of this Rule.

Returns:

The newly cloned Rule.

2.3.3.13 bool Rule::createFromFile (std::string *fName*, Data * *d*, RULE_TYPE *type*, float(*)(Rule *) *scoreFcn* = NULL)

Creates a Rule from reading in a file.

Parameters:

fName The name of the file.

Data The data set to score and create this Rule with.

type The type of Rule this is.

the method to use to score this rule. Only needed for type = SPECIAL Returns false if no Rule was created. True otherwise.

2.3.3.14 float Rule::compare (Rule * *r*)

Compares two Rules' scores.

Parameters:

r The Rule to compare to this one.

Returns:

>0 if r is greater than this Rule, 0 if r is equal to this Rule, <0 if r is less than this Rule.

2.3.3.15 bool Rule::isSatisfied (std::vector< InstanceElement * > * *instance*)

Checks to see if an instance of data is satisfied by this Rule.

Parameters:

instance The instance of data.

2.3.3.16 void Rule::score ()

Scores the Rule based on which type of Rule it is.

2.3.3.17 int Rule::findAttribute (int *attribute*)

Checks to see if an attribute index is already in the Rule.

Parameters:

attribute The attribute index to search for.

Returns:

-1 if the attribute does not exist, the index of the attribute otherwise.

2.3.3.18 int Rule::findAttributeValue (int *attribute*, int *value*)

Checks to see if an attribute value is already in the Rule.

Parameters:

attribute The attribute whose value is to be searched for.

value The value of the attribute to search.

Returns:

-1 if the value is not found, the index of the value otherwise.

2.3.3.19 bool Rule::hasComponent (int *attribute*, int *value*)

Checks to see if a certain attribute value is in the Rule.

Parameters:

attribute The attribute index to check the value of.

value The value index of the attribute.

Returns:

True if it attribute = value is in this Rule, false otherwise.

2.3.3.20 bool Rule::isEqualTo (Rule * *r*)

Checks to see if two Rules have the same component sets.

Parameters:

r The Rule to compare to this one.

Returns:

True if r == this, false otherwise.

2.3.3.21 bool Rule::addComponent (int *attribute*, int *value*)

Adds a new component to the Rule.

Parameters:

atribute The attribute to add.

vale The value of the attribute to add.

Returns:

True if it was added, false if it already was in the rule.

2.3.3.22 Rule * Rule::combine (Rule * *r*)

Combines two Rules by adding together their disjunctions and conjunctions. If this Rule and r are equivalent, the new Rule is just a clone of the first rule.

Parameters:

r The Rule to add to this one.

Returns:

A pointer to a new Rule that is created from this one.

2.3.3.23 Rule * Rule::backSelect ()

Attempts to create a better, smaller rule.

Returns:

The smaller rule.

2.3.3.24 void Rule::printGotWant (std::ostream & *stream*, Data * *eData*)

Prints a Weka-like got want matrix to allow for evaluation alongside the Weka.

Parameters:

stream The stream to print o.

eData The data containing the proper line of code information.

2.3.3.25 void Rule::print (std::ostream & *stream*)

Prints the Rule in the format: A = [1 OR 2] AND B = [2] AND C = [1 OR 4] Score: ### <optional scoring="" metrics>="">

Parameters:

stream The stream to print the Rule to.

2.3.3.26 void Rule::printRule (std::ostream & *stream*)

Prints just the Rule portion in the format. A = [1 OR 2] AND B = [2] AND C = [1 OR 4] => class

Parameters:

stream The stream to print tte Rule to.

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

- My Documents/Zach/School/Research/Which/which/Rule.h
- My Documents/Zach/School/Research/Which/which/Rule.cpp

2.4 RuleSet Class Reference

`#include <RuleSet.h>`

Collaboration diagram for RuleSet:

Public Member Functions

- RuleSet ()
- RuleSet (Data *train, Data *test, RULE_TYPE type)
- ~RuleSet ()
- void create ()
- void score ()
- void print (std::ostream &stream)

2.4.1 Detailed Description

This class represents a collection of Rules that are meant to cover an entire data set.

2.4.2 Constructor & Destructor Documentation

2.4.2.1 RuleSet::RuleSet ()

Empty Constructor.

2.4.2.2 RuleSet::RuleSet (Data * *train*, Data * *test*, RULE_TYPE *type*)

Creates a basic RuleSet. train and test MUST be discrete already.

Parameters:

train The data set this RuleSet will train on.

test The data set this RuleSet will evaulate on. Passing NULL will result in the RuleSet being evaulated on the test data.

type The type of Rules this RuleSet contains.

2.4.2.3 RuleSet::~RuleSet ()

Destructor.

2.4.3 Member Function Documentation

2.4.3.1 void RuleSet::create ()

Creates a series of Rules that attempt to best cover the Data.

2.4.3.2 void RuleSet::score ()

Scores the RuleSet.

2.4.3.3 void RuleSet::print (std::ostream & *stream*)

Prints the RuleSet.

Parameters:

stream The stream to print to.

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

- My Documents/Zach/School/Research/Which/which/RuleSet.h
- My Documents/Zach/School/Research/Which/which/RuleSet.cpp

2.5 WhichStack Class Reference

`#include <WhichStack.h>`

Public Member Functions

- WhichStack ()
- WhichStack (int maxSize)
- ~WhichStack ()
- void create (Data *data, int type, float alpha=1, float beta=1, float gamma=1)
- bool push (Rule *r)
- int select (int count=2000, int check=200, float improve=0.2)
- bool pickTwo ()
- unsigned int size ()
- Rule * getBest ()
- Rule * getRule (int index)
- std::ostream & report (std::ostream &stream, int n)
- void print (std::ostream &stream)
- bool push (int[ ] attVal, float(*scoreFcn)(Rule *))

Protected Member Functions

- bool contains (Rule *r)
- int pick (std::vector< float > scores, float sum)

2.5.1 Detailed Description

Represents a Which-specific stack. It is sorted and has special facilities geared towards Which.

2.5.2 Constructor & Destructor Documentation

2.5.2.1 WhichStack::WhichStack ()

Empty Constructor.

2.5.2.2 WhichStack::WhichStack (int *maxSize*)

Creates a WhichStack that has a maximum size.

Parameters:

maxSize The maximum size this WhichStack can be. If it is -1, the size if infinite.

2.5.2.3 WhichStack::~WhichStack ()

Destructor.

2.5.3 Member Function Documentation

2.5.3.1 void WhichStack::create (Data * *data*, int *type*, float *alpha* = 1, float *beta* = 1, float *gamma* = 1)

Creates a WhichStack having Rules of single attribute-value pairs.

Parameters:

data The data file to use to create the Rules from.

type The type of Rules to create.

alpha The weight for pd.

beta The weight of pf.

gamma The weight for effort.

2.5.3.2 bool WhichStack::push (Rule * *r*)

Pushes a Rule onto the WhichStack in the position based on the Rule's score. If the Rule would be last on a WhichStack of finite size that is full, the item will not be pushed. — THE RULE WILL NOT BE DELETED IF IT IS NOT ADDED —

Parameters:

r The Rule to push onto the WhichStack.

Returns:

True if the Rule made it onto the WhichStack, false otherwise.

2.5.3.3 int WhichStack::select (int *count* = 2000, int *check* = 200, float *improve* = 0.2)

Calls pickTwo a series of times to attempt to create a "best" Rules.

Parameters:

count The number of times to call PickTwo total.

check How many pickTwo calls to allow to pass before a check is made to make sure improvement is still happening.

improve A decimal number representing the percentage of increase in score a current "best" Rule must have since the last check in order to continue calling pickTwo.

Returns:

The true number of times pickTwo was called. A number <= count.

2.5.3.4 bool WhichStack::pickTwo ()

Based on a weighted distribution, picks two Rules from the WhichStack and combines them.

Returns:

True if the new Rule made it onto the WhichStack, false otherwise.

2.5.3.5 unsigned int WhichStack::size ()

Gets the number of Rules in the WhichStack.

Returns:

The number of Rules in the WhichStack.

2.5.3.6 Rule * WhichStack::getBest ()

Gets the best Rule in the WhichStack.

Returns:

The top of the WhichStack.

2.5.3.7 Rule * WhichStack::getRule (int *index*)

Gets the Rule indexed by index.

Parameters:

index The index of the Rule to get(0 is the same as calling getBest());

Returns:

The Rule at index index.

2.5.3.8 std::ostream& WhichStack::report (std::ostream & *stream*, int *n*)

Prints the first n Rules in the WhichStack.

Parameters:

stream The stream to print to.

n The number of Rules to print.

Returns:

The stream;

2.5.3.9 void WhichStack::print (std::ostream & *stream*)

Outputs the WhichStack to a stream.

Parameters:

stream The stream to output to.

2.5.3.10 bool WhichStack::push (int[ ] *attVal*, float(∗)(Rule ∗) *scoreFcn*)

This is a special method for using the WhichStack as an API. This allows a user to push items in isolation into the WhickStack, much like the standard construction does.

Parameters:

attVal A length 1x2 array that stores an attribute index and its value index to be added to the WhichStack.

scoreFcn A pointer to the method of evaluating a Rule.

2.5.3.11 bool WhichStack::contains (Rule ∗ *r*) [protected]

Checks to see if a Rule is already in this WhichStack.

Parameters:

r The Rulet to look for.

Returns:

True if r is in this WhichStack, false otherwise.

2.5.3.12 int WhichStack::pick (std::vector< float > *scores*, float *sum*) [protected]

Picks a Rule from the WhichStack.

Parameters:

scores The vector of scores for the rule.

max The maximum number to select.

Returns:

The position in the WhichStack of the chosen rule.

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

- My Documents/Zach/School/Research/Which/which/WhichStack.h
- My Documents/Zach/School/Research/Which/which/WhichStack.cpp

Index