AutoUniv Version 1 Reference Manual

1 Overview

AutoUniv (AU) is a tool for creating classification models which can then be used to generate classified examples for the evaluation of classification learning algorithms.

The motivation for AU is to provide an alternative to the data sets available from repositories such as [1]. Repositories provide data sets which are for the most part 'real' having been donated by businesses or by scientific researchers; a few are artificial such as [2]. These repositories have served the Machine Learning well for several decades. It is now the norm that a new algorithm be evaluated on a substantial number of these data sets before publication can be expected. There is a lot to be said for the evaluation of algorithms on real data. Not least, there is the fact that the data has not been manipulated to serve the purposes of the experimenter. This adds to confidence that the results would be replicated on real data encountered in the future. In contrast, with artificial data, there is a sense that its contrived nature renders results obtained less useful as an indicator of future performance. Often it is overly simplistic in structure, following some straightforward rule or mathematical function, possibly with noise added. It is certainly the case that evaluating an algorithm on only artificial data will draw a negative response from a reviewer of a paper.

Nevertheless, artificial data has a compelling advantage over real data: its structure, i.e. the underlying mapping of a description to a class (whether it is deterministic or probabilistic) is known. This allows for a more thorough white-box approach to evaluation. It also permits systematic variation of parameters or effects. The latter, it is true, is possible with real data. For example noise can be added systematically to real data.

The aim in AU is to marry the benefits of real and artificial data, that is to produce data, in as large a quantity as is required, which has the richness, variety and complexity of real data with the known properties of artificial data.

Repositories store actual data sets, or, in some cases for artificial data, the underlying model itself. With AU, a classification model is built and examples sets are run off from this, i.e. examples are generated at random in accordance with probabilities specified in the model. Thus it is conceived that there would a repository of models which could be accessed by researchers wishing to replicate the work of others.

2 What is a model?

Given several description attributes, discrete or continuous, and a number of mutually exclusive classes or categories, a *classification model* provides a mapping of each possible instance vector of the description attributes (an actual description) to a probability distribution over the classes. The interpretation is: 'if an object has this description then these are the probabilities that it will belong to a particular class'. These associations are referred to as *rules* with the description being the *rule condition*. Rules may be explicated, in small models, as a table enumerating every possible instance but are more typically represented using rule conditions which exhibit a degree of generality.

A class probability distribution having non-zero probability on more than one class is said to possess *noise*. If it is degenerate at a particular class (probability one) it is said to be *noise-free*. The class(es) in the class probability distribution having the largest probability are the *majority class(es)* and the corresponding probability is the *majority class probability*. The rule is said to be a *rule for that class*.

A *deterministic classifier* is obtained by mapping each rule condition to a majority class.

The description attributes are governed by a joint probability distribution which determines the frequency of occurrence of instances. The classification model can be regarded as conditional on this joint distribution. Together they specify a joint distribution on the description attributes and the class attribute.

From this overall joint distribution several important properties of the model may be calculated. The *default class distribution* is the expected class distribution over all rules in the model under the joint description attribute distribution. The class(es) having the largest probability in this distribution are the *default* class(es) and this probability is the *default* or *base rate*. The expected value of the majority class probability for each rule under the joint description attribute distribution is the *Bayes* rate of the model. The Bayes rate is the upper limit for the classification performance of any classifier presented with examples generated in accordance with the model[1]. To know this value is extremely useful when evaluating learning algorithms. The noise level for a model is *1-Bayes rate* (usually expressed as a percentage).The *lift* is defined as *Bayes rate – base rate*.

A characteristic of attributes in the real world is that they may be broken down, at least approximately, into sets of independent factors, i.e. there is probabilistic dependency amongst attributes within a factor but not across factors. Factorisation is implemented in AU. In addition to mirroring the real world, factorisation also produces the considerable computational savings that go with dimension reduction.

Another characteristic of real-world attributes is that they differ in the extent to which they influence the class associated with a description. Informally, if knowledge of an attribute's value can ever influence classification, i.e. alter the probabilities of classes, it is said to be *relevant*, else it is *irrelevant* [3]. An attribute becomes relevant by virtue of being instantiated in one or more rules[2] or, if not, by belonging to a factor in which another attribute in that factor is instantiated. In the latter case the attribute is said to be *redundant*, so-called because if the attribute's value were never known, the Bayes rate could still be achieved. An irrelevant attribute is often referred to as *pure noise*. It can be modelled as an attribute that is not instantiated in any rule and does not belong to a factor in which any other attribute in the factor is instantiated.

---

[1] It is, of course, possible to beat the Bayes rate using a small test set of examples. By luck of the draw, a classifier could classify all correctly.

[2] This definition assumes that not all class distributions are uniform. In the latter case, no attribute, regardless of its instantiation or not in the rule set, is relevant.

3 The model building procedure

In AU, the user specifies a number of properties of the model referred to as *build settings*. A model is then built by a process of constrained randomisation to meet these requirements. As a result of this process, further properties of the model will emerge.

There are several hard-coded limits in AU: a model can have at most 1000 attributes of which at most 500 can be relevant; a discrete attribute can have at most 10 values; the maximum number of attributes in a factor is 11; there are at most 10 classes; the number of rules in a model is at most 3000. These limits serve to contain the workload on AU with regard to memory requirements and execution time.

The main statistical properties, including base rate, Bayes rate and hence, lift are emergent. They can, however, be partially controlled. For example, the Bayes rate is influenced by limits set for noise levels in class distributions.

It is very much line with the philosophy of AU that the user has a limited degree of control. Re-generation of models under the same user specification will produce a variety of results. Trial and error to achieve a particular property, for example, lift, is all part of the process. There is a richness and heterogeneity within and across models which reflects that in real data. Within a model some rules may be more general than others, noise levels may vary from one rule to another and some classes may be more prominent than others.

The properties that are specified by the user are shown below. These are set in the *Build Classification Model* dialog.

Attribute Definitions	**Limits**
Number of attributes (N)	1 - 1000
Number of discrete attributes (ND)	0 - N
Number of nominal attributes discrete	0 – ND
Number of integer attributes discrete	0 - ND
Number of continuous attributes	0 - N
Minimum number of values of a discrete attribute	2 -10
Maximum number of values of a discrete attribute	2 -10
Rule Conditions	
Number of relevant attributes (NR)	0 – min(N,500)
Number of pure noise attributes	0 - min(N,500)
Minimum number of attributes instantiated in a rule	0 - NR
Maximum number of attributes instantiated in a rule	0 - NR
Stopping probability for condition specialisation	0 - 1

Class distributions	
Number of classes (NC)	2 - 10
Class weight (for each class)	0 - 1000
Minimum majority class probability (for each class)	First integer greater or equal to 1000/NC - 1000
Maximum majority class probability (for each class)	First integer greater or equal to 1000/NC - 1000

Constraints operate amongst these settings and are implemented in the dialog. For example the number of discrete plus continuous attributes must equal the overall number of attributes.

Given these settings, AU builds a model as outlined below:

---

define attributes
determine relevant and pure noise attributes
create attribute factors
define classes
build rule conditions
create a class distribution for each rule condition
create joint distribution for each attribute factor

Attributes are named in sequence as att1, att2, … . For discrete nominal attributes, the values are v1, v2, … and for discrete integer attributes they are 0, 1, 2 … .

For a continuous attribute, a set of contiguous ranges (up to 10) is randomly determined and a number of decimal places is chosen from 0 to 3. Thus initially, a continuous attribute is really discrete with its ranges acting as discrete values. Continuous attributes appear alongside discrete attributes in a factor. When examples are generated, a range will be selected as part of the instance of the factor. A random value in the range will then be generated with the required number of decimal places. Generating instances of multivariate continuous attributes is problematic and this method affords a simple and effective means of creating attributes with complex dependencies within a factor and relationship to class.

Attributes are divided randomly into relevant and pure noise sets. Within each set, attributes are grouped randomly into factors. Thus no factor contains both relevant and pure noise attributes.

A rule condition instantiates some attributes and leaves others wild-carded. For discrete attributes, an instantiation has the form *value*; for a continuous attribute it has the form *lower_value* < *upper_value* interpreted as *lower_value* ≤ *att* < *upper_value*.

The set of rule conditions partitions the attribute space. The partition is created by specialisation from the empty condition using a breadth-first search. A node at depth *d*, corresponds to a condition in which *d* attributes are instantiated as discussed above.

Nodes are expanded until the specified minimum depth is reached. At this point if the number of unexpanded nodes exceeds the maximum allowed number of rules (3000), the search fails. Else, while the maximum depth has not been reached, each unexpanded node is expanded, using an available attribute, with probability (1-stopping probability) or terminated (and becomes a rule condition) with the stopping probability. An expansion fails if the resulting number of unexpanded nodes together with the number of rules would exceed 3000. In this case the all remaining unexpanded nodes (including the one whose expansion just failed) are converted to rule conditions and the search finishes.

Only relevant attributes are selected for node expansion. At least one attribute from every relevant factor must be instantiated for the search to succeed.

Feedback on the progress of the search is provided in the *Build Log* window of the *Build Classification Model* dialog.

When the rule conditions have been created, a class distribution is built for each condition. First, a majority class is chosen at random using the class weights assigned in the *Class distribution Settings* dialog. The probability that a class will be chosen as majority is *weight/1000*. Next, a probability for this majority class is assigned randomly within the range, *minimum – maximum*, specified for the class in the *Class distribution Settings* dialog. Finally the residual probability *1- majority class probability* is assigned to the remaining (residual) classes. This is achieved by first determining the minimum number of classes which must receive non-zero probability. For example, if there are five classes and the majority class probability is 0.3 then at least three other classes must have non-zero probability. The actual number of residual classes to receive non-zero probability is chosen at random between the minimum and the number of residual classes. The residual probability is then randomly allocated across this number of classes[3].

At this point, the creation of the rule set, i.e. the classification model, is complete. The user may rebuild the rule set (or try again if the build failed). Under the *Rebuild and Drift* section of the *Build Classification Model* dialog there is the option, from the first two checkboxes, to keep the attributes and the factorisation. Note that keeping the factorisation requires also keeping the attributes. It is possible to change some of the build settings before a rebuild. The choices available will depend on what has been kept. For example, if the factorisation is kept, it is not possible to change the number of relevant attributes as this is an intrinsic part of the factorisation. Further keep options are available for concept drift and are discussed below.

When satisfied with the rule set, the user can save the model. It is now that the joint distribution of the description attributes is created and also saved. This

[3] Zero probability might be assigned to one or more of these classes. Thus the number of residual classes actually receiving positive probability might be less than that initially selected.

distribution consists of separate (independent) randomly built distributions, one for each factor.

4 The saved model

When saving, a model is named and three text files are created which share this name as a file stem. The files are:

<*model_name.aupl*>	A Prolog source code definition of the model.
<*model_name.aurules*>	A text representation of the rules.
<*model_name.auprops*>	A statistical summary of the main properties of the model including base and Bayes rates and lift.

The *.aupl* file provides a complete definition of the model as a collection of Prolog predicates. The details of these are given in the Appendix. In addition to the definition there is a probability calculator for rule conditions, **cpx_prob/2**, and a example generator, **random_select/1**.

The rules are represented in the predicate

rule(Number, Condition, Majority_Class, Class_Distribution)

Details of these arguments are given in the appendix. A rule condition can be difficult to read and interpret especially when there are a large number of attributes and for this reason the *.aurules* file provides a more user-friendly version in a familiar attribute-value format and which is more condensed since wild-carded attributes do not appear.

5 Applying drift to a model

Once a model is saved, the four checkboxes are enabled under the *Rebuild and Drift* section. A new model may be built or drift may be applied to the current model.

To build a new model, leave all boxes unchecked. New build settings for attributes and rules may be applied. Creating a new model completely overwrites the old model.

Concept drift involves changes to the rules while population drift involves changes to the joint attribute distribution. To apply drift, check *Keep attributes definitions*: a drifted model must have the same attribute definitions as the original model (but the number of classes may be changed). Other check boxes then alter the drift type:

No further checks	Concept and population drift
Keep attribute factors	Concept and population drift (but with the same factors)
Keep attribute factor distributions	Concept drift only
Keep rules	Population drift only

Note that to keep attribute factors, attribute factor distributions or rules, the attributes themselves must be kept. Also, to keep the factor distributions or the rules, the factors must be kept. These constraints are implemented in the dialog, preventing illegitimate combinations from being selected.

If *Keep rules* is checked, i.e. only the factor distributions are to be changed, the latter will be re-generated upon saving the model.

The attribute build settings cannot be changed for drift, but other settings may be may be changed depending on drift options selected. Thus for concept drift, the number of classes, the complexity of the drifted rule set and its noise level may be altered from the original.

A sequence of drifted models, i.e. each drifting from the previous, may be built while *Keep attribute definitions* remains checked.

The *.auprops* file for a drifted model calculates, for small models only, the cross-classification rate (*XCR*) for the drift. For this purpose, small means models with up to 50 attributes and original and drifted rule sets of up to 250 rules each. The *XCR* was defined in [4] and is the classification rate of the original rule set that would be obtained if it were used as a deterministic classifier under the new model, i.e. it is the expected value of the majority class probability in the original rules calculated using the attribute distribution of the drifted model. The *XCR* is at most the Bayes rate for the drifted model and its shortfall from this is an indicator of the extent of drift.

6 Practicalities and tips for model building

A certain amount of trial and error will be needed to produce a model that meets the user's needs. In building a rule set with a large number of relevant attributes, sufficient depth must be provided. Even then the challenge is to stay within the 3000 rule limit and to have used all relevant factors. For more than 250 relevant attributes, it is advisable to keep the maximum number of values of a discrete attribute to at most 5. The feedback from the *Build Log* window can be useful for adjusting settings for a rebuild.

To maximise the number of redundant relevant attributes, use as small a maximum depth as possible.

To create rules which vary considerably in the number of attributes instantiated, use a low minimum depth, high maximum depth and a large stopping probability so that the search is forced down rather than across.

To create a substantial lift, use high minimum values for majority class probabilities in the *Class Distributions Settings* dialog. To achieve a fairly uniform default class distribution, use nearly equal class weights and similar minimum and maximum values for the majority class probabilities of each class.

To build a rare class model, first decide whether there are to be rules for the rare class(es), i.e. a rare class is ever a majority. In the real world this can be the case but the majority probabilities are often lower than for common classes. Then apply low weights to the rare classes. If there are no rules for rare classes, then setting a wide range for majority class probabilities can produce an interesting model for learning rare class contours [5].

7 Generating examples from models

Generation of classified examples is undertaken from the *Generate Data from Model* dialog. Selecting a model compiles the *.aupl* file for that model. AU supports generation of examples in csv, ARFF and c4.5 formats. Examples may be written with or without a header. For csv format the header is just a row of attribute names. The *Append without header* option allows for appending of examples to a previously created example set. This is particularly useful for creating data sets containing drift. If a drifted model contains additional classes, then it is the user's responsibility to manually update the header in ARFF and c4.5 formats to reflect this.

References

[1] UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/ .

[2] LED Display Domain Data Set. http://archive.ics.uci.edu/ml/datasets/LED+Display+Domain .

[3] Hickey RJ (1996) Noise modelling and evaluating learning from examples, Artificial Intelligence 82 : 157–179.

[4] Black M and Hickey RJ (1999) Maintaining the performance of a learned classifier under concept drift, Intelligent Data Analysis 3 : 453-474.

[5] Hickey RJ (2003) Learning Rare Class Footprints: the REFLEX Algorithm, Proceedings of Proceedings of the Second International Workshop on Learning with Imbalanced Data Sets, International Conference on Machine Learning, Washington, D.C., 21-24 August 2003 :89-96.

Appendix: Prolog predicates defined in the *.aupl* file

univ/1

univ(Model)

Model : model name

parameters/1

parameters([K, KDisc+KCts, KDiscNom+KDiscInt, Krel+Kpn, M],
[MinV-MaxV, MinDepth - MaxDepth : StopP, Ranges*Ps])

K : # attributes
KDisc : # discrete attributes
KCts : # continuous attributes
KDiscNom : # discrete nominal attributes
KDiscInt : # discrete integer attributes
Krel : # relevant attributes
Kpn : # pure noise attributes
M : # classes
MinV : minimum number of values for a discrete attribute
MaxV : maximum number of values for a discrete attribute
MinDepth : minimum # attributes instantiated in a rule
MaxDepth: maximum # attributes instantiated in a rule
StopP : stopping probability
Ranges : list of (MinMajP – MaxMajP) where MinMajP is the minimum and MaxMajP the maximum majority class probability (* 1000) for a class
Ps : list of class weights

attributes/1

attributes (Atts)

Atts : list of attribute names

attributes_discrete/1

attributes_discrete(DiscAtts)

DiscAtts : list of discrete attribute names

attributes_continuous/1

attributes_continuous(CtsAtts)

CtsAtts : list of continuous attribute names in format Att/NDec where Att is the attribute name and NDec is the number of decimal places (0 – 3) when an instance is generated

att_values/2 att_values(Att, Vs) Att : attribute name Vs : list of values
classes/1 classes(Classes) Classes : list of class names
attribute_relevance/2 attribute_relevance(Type, Atts) Type : relevant | relevant_redundant | pure_noise Atts : list of attributes names of that type
attribute_factors/1 attribute_factors(Factors) Factors : list of Fac where Fac is a list of attribute names for that factor
number_of_rules/1 number_of_rules(N) N : # of rules
rule/4 rule(N, Cond, MajClass, CD) N : rule number Cond : list of attributes instantiated or wild-carded representing rule condition MajClass : majority class CD : class distribution

att_distn/2 att_distn(Factor, AttD) Factor : list of attribute names for factor AttD : list of FacInst * P where FacInst is a list representing a factor instance and P is the probability of that instance
cpx_prob/2 [4] cpx_prob(Cond, P) Cond: list of attributes each of which may be a variable or instantiated P : probability of Cond
random_select/1 random_select(e(Description, Class)) Description : fully instantiated list of attribute instances Class : class assigned to Description

[4] cpx_prob/2 and random_select/1 are rules which depend on internally defined predicates.