An Investigation into the Interaction between Feature Selection and Discretization: Learning How and When to Read Numbers

Sumukh Ghodke[1] and Timothy Baldwin[1,2]

[1] Department of Computer Science and Software Engineering
University of Melbourne, VIC 3010, Australia

[2] NICTA Victoria Laboratories
University of Melbourne, VIC 3010, Australia

sumukh.ghodke@gmail.com, tim@csse.unimelb.edu.au

Abstract. Pre-processing is an important part of machine learning, and has been shown to significantly improve the performance of classifiers. In this paper, we take a selection of pre-processing methods—focusing specifically on discretization and feature selection—and empirically examine their combined effect on classifier performance. In our experiments, we take 11 standard datasets and a selection of standard machine learning algorithms, namely one-R, ID3, naive Bayes, and IB1, and explore the impact of different forms of preprocessing on each combination of dataset and algorithm. We find that in general the combination of wrapper-based forward selection and naive supervised methods of discretization yield consistently above-baseline results.

1 Introduction

Machine learning is an experimental science encompassing the areas of probability theory, statistics, information theory and theoretical computer science. It deals with the automatic extraction of useful and comprehensible knowledge from data. Much of machine learning is focused on classification and other predictive tasks, where the value of an attribute is predicted based on prior knowledge.

In classification, a model is built based on known data which is then used to predict the discrete values of new data, in the form of a class label. The input to a classifier is called the training data, while the unseen data to which results of the learning process are applied is called the test data. The classification process can be modelled as a pipeline of one or more components, obligatorily containing a classifier module, and optionally including a pre-processor module. When a pre-processor is present it precedes the classifier and transforms the input data into a form that (ideally) enhances classifier performance.

Data is the crux of all classification tasks. The way in which data is presented to a learning algorithm often has a significant impact on its performance. In classification tasks, data is usually represented in the form of instances, i.e. single data records characterised by one or more attributes (also known as features) and a unique class label. For our purposes, we assume that there are two types

Ghodke, Sumukh and Timothy Baldwin (2007) An Investigation into the Interaction between Feature Selection and Discretization: Learning How and When to Read Numbers, In Proceedings of the 20th Australian Joint Conference on Artificial Intelligence (AI07), Gold Coast, Australia, pp. 48–57.

of attributes: **discrete** attributes, where there is a finite set of values with no explicit or implicit order (e.g. HIGH, MEDIUM or LOW as values for a TEMPERATURE feature); and **continuous** attributes, which take real numeric values (e.g. a temperature value in degrees Celcius).

There are many ways to pre-process data when building a classifier, but in this paper we focus exclusively on discretization and feature selection. The aim of discretization is to convert continuous attributes into meaningful discrete attributes. Feature selection is the process of pre-selecting a subset of features to train the classifier over, generally with the intention of removing redundant and/or irrelevant attributes from the dataset.

1.1 Case study

Due to the importance of pre-processing in machine learning, there is a considerable volume of literature proposing new methods or comparing existing ones. Examples of such studies include [5], [3], [9] and [10] where the effects of discretization and feature selection on classifier performance were studied independently. As many learners prefer to operate over discrete attributes, it is important to select the best discretization method for a given dataset. Things are complicated further when feature selection is included as a second form of preprocessing.

The task of finding the best combination of feature selection and discretization has largely been based on heuristics or prior knowledge with similar datasets and exhaustive experimentation. Our study attempts to unearth general trends in the interaction between feature selection and discretization by applying various combinations of methods to a range of datasets and evaluating their output relative to a sample of learner algorithms.

1.2 A new framework for machine learning experimentation

While the machine learning community has generally been good at sharing datasets and toolkits, and developing fully automated evaluation methodologies, it would be a stretch to say that full reproducibility is the norm in machine learning research. This is due to effects such as a lack of documentation of data splits/sampling, variability in the preprocessing of data, unpublished parameter settings and scope for interpretation in the implementation of learning algorithms. As part of this research, we have developed an open-source machine learning toolkit from scratch which includes all code used to perform these experiments, all datasets in exactly the form used to carry out our experimentation, and a script to automatically run all experiments in their entirety, for maximum transparency. All the above features were built into a new machine learning component of the NLTK-Lite toolkit [2], which is available from `nltk.sourceforge.net`.

An object-oriented modular implementation of the algorithms targeted in this research enhances the readability and understandability of the implemented algorithms. Additionally, the choice of python as the programming language makes the implementations much closer to pseudocode than is the case with existing machine learning toolkits. It also has a fair coverage of algorithms required for experimentation, bundled with 11 datasets from the UCI machine

learning repository. It provides tools and utilities which enhance productivity and experimental reproducibility like the presence of a batch testing utility.

2 Methodology

This section outlines the algorithms used in our case study, then goes on to describe the discretization and feature selection techniques. A brief description of the datasets used in our experiments is then followed by findings from past research.

2.1 Classification Algorithms

Our toolkit currently implements five classification algorithms: (1) zero-R, as a baseline classifier; (2) the one-R decision stump induction algorithm; (3) the ID3 decision tree induction algorithm; (4) the naive Bayes algorithm; and (5) the IB1 instance-based learning algorithm. For full descriptions of these algorithms, see [7], [6] and [1]. Note that of these algorithms, naive Bayes and IB1 have an in-built handling of continuous attributes (in the form of a Gaussian PDF, and continuous distance metric, respectively). The remainder of the algorithms are implemented to ignore all continuous attributes, i.e. to operate only over the discrete attributes in a given dataset.

2.2 Discretization Methods

Discretization is the process of converting continuous attributes into discrete ones. Some classification algorithms like C4.5 [6] have the facility to perform localized discretization at a given node in a decision tree. For our purposes, however, we focus exclusively on global discretization, in discretizing the training dataset once to produce discrete intervals for each attribute. The intervals are then used to map continuous values to discrete values for both the training and test datasets.

Discretizers can be categorised into **supervised** and **unsupervised** methods, based on whether they take the labels of the training instances into account in determining the breakpoints between discrete intervals. We focus on two unsupervised discretization algorithms, namely unsupervised equal width and unsupervised equal frequency, and two supervised methods of discretization, the first of which has two modified variants.

Unsupervised Equal Width (UEW) The unsupervised equal width method is the simplest of all discretization algorithms. In this algorithm, the lowest and highest continuous attributes are used to define the input range. This range is then divided into a fixed number of intervals of equal width. If the attribute values range from x_{min} to x_{max} and the number of intervals is given by k, the width of each interval δ is given by $\delta = \frac{x_{max} - x_{min}}{k}$. k intervals are created with the first interval starting from x_{min} by adding δ to the end of each previous interval. In our experiments, k was set to 10.

The main disadvantage of equal width discretization is that it is severely affected by outliers. The intervals are created solely based on the extreme values, which may be erroneous data.

Unsupervised Equal Frequency (UEF) The unsupervised equal frequency discretization (a.k.a. histogram equalization) method attempts to improve on equal width discretization by focussing on individual continuous values, rather than on extreme values in an attribute. A breakpoint is inserted in the ordered sequence of continuous values after every k^{th} instance in the sequence ($k = 10$ in all our experiments). If the values on either side of a breakpoint are the same the breakpoint is shifted until it separates two distinct values. This process thus creates unevenly sized intervals and gives a closer representation to the actual distribution of continuous values.

Despite its simplicity, this method has been shown to give excellent results when combined with some algorithms, e.g. with k set to the square root of the number of instances [8]. K-means clustering can also be used to find the value of k, but, in general, the task of calculating and specifying k still rests with the user, which is a disadvantage of this method.

Naive Supervised Discretization (NS) The naive supervised discretization method is supervised. When discretizing using this algorithm, continuous attribute values are evaluated along with their corresponding class labels. The list of continuous values is sorted to form an ordered sequence and breakpoints are placed at all positions where the class membership changes. The breakpoints are then used to form intervals, where each interval represents a discrete value.

This algorithm is simple to implement and guarantees that the classes of instances within an interval are the same in almost all cases. As breakpoints are placed on the basis of change in class membership, it lacks any sense of ordering. In the worst scenario, if the class membership changes for every instance there will be as many discrete values as there are instances. The other disadvantage of this algorithm is that it can lead to overfitting. Two modified versions of this algorithm are discussed in the following subsections, which change the way breakpoints are inserted in order to overcome the issue of overfitting.

Naive Supervised Modified Version 1 (NS1) In this modified version of naive supervised discretization, the breakpoints are not inserted every time the class membership changes. Instead, they are inserted only after a minimum of N instances of the majority class are contained in each interval. The threshold number of instances is a user defined value, and should be computed based on the type of data. For our experiments, we set N to one-fifteenth the number of training instances.

Naive Supervised Modified Version 2 (NS2) The second modified version tries to avoid overfitting by merging two previously formed intervals until they contain at least N instances of the majority class. This can lead to results which are quite different from the first version, depending on how frequently the class membership changes and the value of N. Once again, we set N to one-fifteenth the number of training instances in our experiments.

Entropy-based Supervised Discretization (ES) In the final form of supervised discretization, we identify the interval with the highest entropy and select

the binary breakpoint which minimizes the mean information of the two generated intervals. This is carried out recursively until N partitions are produced.

2.3 Feature Selection

The task of feature selection is to remove low-utility attributes such that the resulting attribute set generates a simpler classifier without compromising the classifier's performance. To be completely sure of the attribute selection, we would ideally have to test all the enumerations of attribute subsets, which is infeasible in most cases as it will result in 2^n subsets of n attributes. The two main types of feature selection methods are filter and wrapper methods, each of which use greedy search strategies to reduce the search space.

Filter feature selection methods evaluate attributes prior to the learning process, and without specific reference to the classification algorithm that will be used to generate the final classifier. The filtered dataset may then be used by any classification algorithms. Rank-based feature selection is a commonly-used filter method and is discussed in the following section.

Wrapper methods use a controlled enumeration strategy and apply the classification algorithm that will be used to generate the final classifier to test the performance of each attribute subset. The ways in which the attribute sets are incrementally updated results in two types of wrapper models: forward selection and backward elimination, as detailed below.

Rank based Feature Selection (Rank) Rank based feature selection is one of the simplest and most efficient means of filter-based feature selection. In this process each feature is evaluated in turn to assign a relative figure of merit, based on which the features are ranked. The rank can be calculated using information gain (**Rank-IG**) or gain ratio (**Rank-GR**), for example. Once the attributes are ranked, the top N ranked attributes are selected. In our experiments, we set N to two-thirds the number of attributes for datasets with less than 10 attributes, half the number of attributes for datasets with between 10 and 20 attributes, and 20 attributes for larger datasets (up to 100 attributes).

A disadvantage of this algorithm is that it does not capture the interaction of features or detect feature redundancy. The algorithm used to rank attributes also tends to have biases, and requires that the user specifies a value for N.

Forward Selection (FS) In forward selection, we start with a seed set of zero attributes and add in one attribute at a time to the seed set [4]. Each feature subset thus formed is evaluated using stratified cross validation, and a greedy selection strategy is used to select the attribute set which results in the highest performance at each level. This attribute is then added to the seed set, and each of the remaining attributes is experimentally added to the expanded feature set on the next iteration of the algorithm. This process continues until all attributes are included, or the increment in classifier performance falls below a δ value.

Backward Elimination (BE) Backward elimination is similar to forward selection in operation, except that the starting state is all attributes, from which

Dataset name	No. of instances	No. of discrete attributes	No. of continuous attributes	No. of classes
annealing	798	32	6	6
australian	690	8	6	2
breast-cancer	286	9	0	2
diabetes	768	0	8	2
german	1000	13	7	2
glass	214	0	10	7
heart	270	8	5	2
iris	150	0	4	3
monks-1	124	7	0	2
monks-2	169	7	0	2
monks-3	122	7	0	2
vehicle	846	0	18	4
votes	435	16	0	2

Table 1. Outline of the UCI datasets used in this research

one attribute is removed at a time until the increment in classifier performance from one iteration to the next fails to grow. The backward elimination algorithm tends to favour a higher number of attributes as compared to forward selection.

2.4 Datasets

Datasets from the UCI machine learning repository are commonly used to benchmark machine learning research. The data used in our experiment was imported into NLTK-Lite (including establishing canonical data splits where they weren't predefined) to facilitate full reproducibility when conducting future experiments using our toolkit. Table 1 lists the datasets imported along with instance, attribute and class composition.

3 Past research

Effect of discretization Dougherty et al. [3] compared the effects of unsupervised and supervised methods of discretization. They identified the characteristics of individual methods and performed an empirical evaluation by experimenting with different discretization methods and classifiers.

The results obtained from their study show that the performance of classifiers almost always improves with global discretization. It is also found that the discretized version of the naive Bayes algorithm slightly outperforms the C4.5 algorithm on these datasets. A combination of naive Bayes with entropy based supervised discretization is said to have resulted in marked improvements, although even the binning method of discretization improved results for the naive Bayes classifiers. The supervised version of discretization is claimed to be generally better than the unsupervised method. The relatively small effects of supervised methods on C4.5 in general was attributed to either the induction algorithm not being able to fully utilize the local discretization options or the local discretization not helping with the induction algorithm on the given algorithms.

These general results were independently verified by Yang and Webb [10] with respect to the naive Bayes algorithm, at the same time as looking at a wider variety of supervised discretization methods and proposing a novel method.

Effect of feature selection John et al. [5] conducted experiments with the ID3 and C4.5 algorithms on 6 artificial and real datasets, and combinations of filter and wrapper methods of feature selection, to identify general trends. They observed that the wrapper method of feature selection does not generally improve the performance of classifiers significantly. The size of decision trees constructed using these datasets was smaller than ones built using other data sets, although there was an increase in accuracy in a few cases.

4 Experiments

To analyse the effects of pre-processing on classifier performance, we take a range of datasets and classification algorithms, and apply a selection of combinations of pre-processing methods to each. In this, we first discretize each dataset using one of 6 different discretization algorithms (see above). We then optionally perform feature selection (including on non-discretized data) based on one of four methods (see above). In sum, pre-processing results in 4 different types of dataset: unchanged datasets (without discretization or feature selection), datasets with discretized features, datasets with a subset of attributes based on feature selection, and discretized, feature-selected datasets. We evaluate the performance of each classification algorithm over the different pre-processed versions of a given dataset. In all cases, we use simple classification accuracy to evaluate our classifiers.

For all our experiments, we use a zero-R classifier without any preprocessing as our baseline classifier. All results are presented in terms of error rate reduction (e.r.r.) relative to the classification accuracy for this baseline.

In the experiments related to wrapper based methods, 25-fold stratified cross validation is used internally to find the accuracy at each level, and a delta value of 0.1 on classification accuracy is used as the stopping criterion.

Classification is performed based on 5-fold stratified cross validation on each data set and the reported accuracies are the average of results obtained in each trial.

As mentioned above, the complete code and suite of datasets used to run these experiments is contained in the NLTK-Lite distribution.

5 Results

For easier analysis, the classification accuracy of each classifier is calculated with respect to the baseline performance. Averages and variations of accuracy improvements (or degradations) over all datasets are given in Table 2.

Algorithm	Feature selection	Discretization						
		None	UEW	UEF	NS	NS1	NS2	ES
1-R	None	.1130 (±.1988)	.3024 (±.2506)	.3042 (±.2548)	.3029 (±.2428)	.2980 (±.2578)	.3038 (±.2617)	.2765 (±.2541)
	Rank-IG	.1130 (±.1988)	.3024 (±.2506)	.3042 (±.2548)	.3029 (±.2428)	.2980 (±.2578)	.3038 (±.2617)	.2765 (±.2541)
	Rank-GR	.1130 (±.1988)	.3024 (±.2506)	.3042 (±.2548)	.3029 (±.2428)	.2980 (±.2578)	.3038 (±.2617)	.2765 (±.2541)
	FS	.2033 (±.1827)	.3083 (±.2461)	.3101 (±.2504)	**.3209** (±.2469)	.3106 (±.2518)	.3148 (±.2538)	.2908 (±.2511)
	BE	.2033 (±.1827)	.3024 (±.2506)	.3042 (±.2548)	.3029 (±.2428)	.2980 (±.2578)	.3038 (±.2617)	.2765 (±.2541)
ID3	None	.1041 (±.2050)	.3134 (±.2502)	.3149 (±.2519)	.2750 (±.2568)	.3098 (±.2475)	.3078 (±.2460)	.2946 (±.2389)
	Rank-IG	.1014 (±.1972)	.3169 (±.2493)	.3117 (±.2511)	.2776 (±.2620)	.3160 (±.2492)	.3046 (±.2520)	.2956 (±.2399)
	Rank-GR	.1038 (±.1950)	.3109 (±.2538)	.3037 (±.2581)	.2847 (±.2564)	.3054 (±.2551)	.3036 (±.2516)	.2868 (±.2450)
	FS	.2552 (±.2280)	.3083 (±.2461)	**.3222** (±.2654)	.3209 (±.2469)	.3106 (±.2518)	.3148 (±.2538)	.2908 (±.2511)
	BE	.2244 (±.2188)	.3134 (±.2502)	.3149 (±.2519)	.3065 (±.2344)	.3098 (±.2475)	.3078 (±.2460)	.2946 (±.2389)
NB	None	.2756 (±.2366)	.2662 (±.2678)	.2743 (±.2687)	.2902 (±.2646)	.2758 (±.2673)	.2753 (±.2660)	.2283 (±.2399)
	Rank-IG	.2411 (±.2187)	.2841 (±.2682)	.2848 (±.2580)	.3120 (±.2684)	.2909 (±.2647)	.2990 (±.2706)	.2596 (±.2423)
	Rank-GR	.2411 (±.2187)	.2880 (±.2680)	.2859 (±.2573)	.3084 (±.2619)	.2856 (±.2609)	.2920 (±.2649)	.2587 (±.2423)
	FS	.2875 (±.2325)	.3031 (±.2539)	.3055 (±.2555)	**.3157** (±.2515)	.2991 (±.2531)	.3081 (±.2610)	.2855 (±.2564)
	BE	.2756 (±.2366)	.2662 (±.2678)	.2743 (±.2687)	.2902 (±.2646)	.2758 (±.2673)	.2753 (±.2660)	.2283 (±.2399)
IB1	None	.2444 (±.2895)	.3225 (±.2649)	.3249 (±.2626)	.3439 (±.2598)	.3252 (±.2611)	.3388 (±.2604)	.3081 (±.2341)
	Rank-IG	.2297 (±.2382)	.3288 (±.2632)	.3251 (±.2657)	.3487 (±.2472)	.3279 (±.2549)	.3436 (±.2490)	.3094 (±.2325)
	Rank-GR	.2448 (±.2354)	.3250 (±.2706)	.3183 (±.2673)	**.3509** (±.2556)	.3176 (±.2590)	.3330 (±.2585)	.3011 (±.2378)
	FS	.2939 (±.2397)	.3083 (±.2461)	.3227 (±.2660)	.3209 (±.2469)	.3106 (±.2518)	.3148 (±.2538)	.2908 (±.2511)
	BE	.2444 (±.2895)	.3225 (±.2649)	.3249 (±.2626)	.3439 (±.2598)	.3252 (±.2611)	.3388 (±.2604)	.2785 (±.2362)

Table 2. Mean error rate reduction and standard deviation (in parentheses) over the 11 datasets, relative to a Zero-R baseline (the highest mean e.r.r. for each algorithm is presented in **bold face**)

5.1 Observations

From Table 2, we observe that wrapper-based feature selection almost always results in improved or equal performance relative to no feature selection, for a given discretization method. Of the two wrapper-based methods, forward selection (FS) is in most cases superior to backward elimination (BE). There are some interesting incompatibilities with the wrapper-based feature selection methods: forward selection is the worst of the feature selection methods (including no feature selection) in the majority of cases when combined with the IB1 algorithm; backward selection, on the other hand, has almost no impact when combined with each of the supervised discretization methods, and has no impact whatsoever in combination with naive Bayes (NB) and IB1 in all but one case. In general, forward selection tends to select a very small number of attributes (often just a single attribute!), whereas backward selection *eliminates* a relatively small number of attributes.

There is little separating the two rank-based filter feature selection methods (Rank-IG and Rank-GR), perhaps unsurprisingly given that they are both based on mean information. Both have absolutely no impact on the performance of one-R, as it uses largely the same means of selecting a single attribute to form a decision stump as the rank-based feature selection methods, meaning that the preferred attribute for one-R is always included in the selected subset of attributes.

It is interesting to note that both one-R and ID3 carry out implicit feature selection in the process of generating a decision stump and decision tree, respectively, and yet both benefit from explicit feature selection in pre-processing. That is, it appears to be the case that feature selection optimizes classifier performance even in cases where the classification algorithm carries out its own feature selection.

Supervised discretization generally outperforms unsupervised discretization. Naive supervised discretization (NS) is the pick of the supervised methods, surprisingly given its inherent naivety, and the two modified variants of the basic method generate a marked improvement in results only in combination with ID3. Entropy based supervised discretization (ES) is the weakest of the supervised methods. Of the two unsupervised discretization methods, unsupervised equal frequency (UEF) performs the best, and in fact consistently outperforms the supervised discretization methods in combination with ID3.

Recall that naive Bayes and IB1 both have in-built mechanisms for handling continuous attributes, such that it is meaningful to compare the results of the different discretization methods with the native algorithms. There is a remarkable increment in discretizing the data and effectively bypassing this facility in both cases, with naive supervised discretization being the pick of the discretization methods in almost all cases. That is, despite the ability of these algorithms to deal natively with continuous attributes, better results are achieved through explicit pre-discretization of the data.

Importantly, feature selection tends to complement discretization, i.e. any gains in performance tend to be additive when the two forms of pre-processing are combined.

To summarise, from Table 2 we can conclude that the combination of naive supervised discretization and forward selection wrapper based feature selection is optimal across different datasets and classification algorithms.

6 Conclusions

We have surveyed a range of pre-existing discretization and feature selection methods, and investigated their performance in combination with a selection of classification algorithms when used to preprocess a range of datasets. The results of our experiments show that the forward selection wrapper based method is the pick of the feature selection algorithms, and complements even classification algorithms which include implicit feature selection of their own. Among the discretization methods, the naive supervised method was found to perform best in most cases, and that explicitly discretizing the data is superior to relying on native handling of continuous attributes within naive Bayes and IB1. Additionally, these two forms of preprocessing complement each other to often improve the overall performance of a given classification algorithm over a given dataset.

References

1. D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.
2. S. Bird. NLTK-Lite: Efficient scripting for natural language processing. In *Proc. of the 4th International Conference on Natural Language Processing (ICON)*, pages 11–18, Kanpur, India, 2005.
3. J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Proc. of the 12th International Conference on Machine Learning*, pages 194–202, 1995.
4. N. R. Draper and H. Smith. *Applied Regression Analysis*. Wiley-Interscience, New York, USA, 1998.
5. G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Proc. of the 11th International Conference on Machine Learning*, pages 121–129, 1994.
6. J. R. Quinlan. Induction of decision trees. In J. W. Shavlik and T. G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann, 1990.
7. P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, Boston, MA, USA, 2006.
8. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, USA, 2005.
9. Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. In *Proc. of the 14th International Conference on Machine Learning*, 1997.
10. Y. Yang and G. I. Webb. On why discretization works for Naive-Bayes classifiers. In *Proc. of the 16th Australian Conference on Artificial Intelligence (AI 03)*, pages 440–52, Perth, Australia, 2003.