AN EXPLORATION OF NAÏVE BAYESIAN CLASSIFICATION AUGMENTED WITH CONFIDENCE INTERVALS

By

PAUL ANTHONY MANCILL JR

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY
School of Engineering and Computer Science

August 2010

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of PAUL ANTHONY MANCILL JR find it satisfactory and recommend that it be accepted.

Date of Signature

Scott A. Wallace, Ph.D.
Assistant Professor of Computer Science
Thesis Adviser, Chair of Committee

Orest J. Pilskalns, Ph.D.
Assistant Professor of Computer Science

Wayne O. Cochran, Ph.D.
Assistant Professor of Computer Science

AN EXPLORATION OF NAÏVE BAYESIAN CLASSIFICATION AUGMENTED WITH CONFIDENCE INTERVALS

Abstract

by Paul Anthony Mancill Jr, MS
Washington State University

August 2010

Chair: Scott A. Wallace

Instance classification using machine learning techniques has numerous applications, from automation to medical diagnosis. In problem domains such as spam filtering, classification must be performed quickly across large datasets. In this paper we begin with machine learning techniques based on naïve Bayes and attempt to improve classification accuracy by taking into account attribute and class confidence intervals. Our classifiers operate over nominal datasets and retain the asymptotic time complexity of linear learning and prediction algorithms. We present results indicating a modest improvement over the naïve Bayes classifier alone across a range of multi-class nominal datasets.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Space

Machine learning surrounds us, even if we do not recognize it as such, comprising the core of every Internet search engine, spam filter, and data mining application. Conceived of as a general technology tool, supervised learning provides the computer scientist with the opportunity to apply the knowledge of subject matter experts to vast seas of data. Discussing attributes of rational agents in their text on artificial intelligence, Russell and Norvig list *machine learning* (ML) as a necessary capability in order to pass the Turing test, defining it as "the [ability] to adapt to new circumstances and to detect and extrapolate patterns" [1, p. 3]. Witten and Frank approach the subject from the perspective of a world in which we are inundated with potentially valuable data, provided we can make sense of it, and treat ML as synonymous with the term *data mining*. We focus on a specific problem in ML, that of *supervised learning* for *classification*, wherein the ML algorithm develops a statistical *model* by observing instances during *training*. An *instance* is the smallest quantum of a pattern to be learned by an algorithm; training consists of exposing the ML algorithm to example instances for which the correct answer (or *class*) is known and provided to the algorithm. Once trained, the algorithm is tasked

with predicting the correct class for subsequent instances, known as either *testing* or *prediction*. Machine learning is a subfield of artificial intelligence because when performed with a high level of accuracy, we might say that the computer has "learned" to recognize instances of a pattern and is acting rationally by consistently classifying them correctly.

Instances are comprised of one or more *attributes* or *features* (the terms are used interchangeably), each of which has a value. The values can be character strings, integers or real-valued numbers, Boolean flags, members of a discrete set of values (of any of the aforementioned types), and in some cases absent within an instance. Not all ML algorithms perform equally well in all problem domains, or are suited for particular attributes types or classes. Classification datasets for which all of the attribute and class values belong to a discrete set are referred to *nominal*. In order to classify real-valued data, either the ML algorithm must be equipped to construct a model from it, or the data can discretized into ranged buckets to create a nominal dataset. Nominal datasets can represent a wide variety of real-world classification problems. Email filtering is an example where headers and keywords can be represented as nominal attributes, and the classification is either spam or ham (non-spam). Also consider a medical diagnosis dataset where the attributes are a series questions of the form "the patient exhibits symptom x," and the resulting diagnosis (class) belongs to the set {immune, susceptible, infected}. The difference in possible classes between these examples illustrates the delineation between 2-class or *binary* classification and the *multi-class* medical diagnosis.

The nominal, multi-class classification problem in machine learning is the province in which our research takes place. We focus specifically on naïve Bayesian (NB) classifiers and how their performance, in terms of classification accuracy, might be improved through the use of confidence intervals. Confidence intervals represent the expected uncertainty the classifier may have in a specific attribute or class based on its prevalence in the dataset. We defer a more thorough introduction of confidence intervals to Section 2.2, and proceed

as follows.

In the remainder of the introductory chapter, we posit our problem statement and the contributions of this thesis. We then provide an introduction to our tool set, background on naïve Bayes classification, and methods of evaluation for machine learning classifiers. Next we discuss related work before delving into the background material and initial experiments with confidence intervals in Chapter 2. One particular CI-augmented classifier presented itself as a successful candidate during initial experimentation and is thus the subject of the remainder of the thesis. In Chapter 3 we evaluate our classifier's performance against both stock and real-world datasets and in the presence of noise and underflow, the latter suggesting a potentially useful heuristic based on the prediction confidence interval attribute. We conclude with a discussion of the run-time performance of NB-variants in Chapter 4 and future logical steps for our research in Chapter 5.

1.1.1 Problem Statement

With the volume of data to be classified growing exponentially, and despite Moore's Law and the steady progress in the speed of commodity hardware, an imperative remains for ML algorithms with low asymptotic run-time complexity. The idea of a classifier that enhances performance over naïve Bayes while maintaining its asymptotic complexity is therefore one component of our motivation. The other is to explore the utility of confidence intervals (that is, the frequency of occurrence of an attribute or class value relative to sample size) as an aspect of classification. These frequencies are gleaned during training, and so are in essence "free" in the sense of time complexity. Thus our task is to either lend credence to or dispel the notion of confidence intervals or derivatives thereof as a useful facet of the machine learning problem.

1.1.2 Contributions of the Thesis

This work extends upon prior work regarding confidence intervals in the area of nomogram visualization of naïve Bayesian classifiers by Možina *et al.* [2] by expanding upon their proposition that confidence is useful as an element of visualization through the conjecture that confidence is useful for classification itself. We first establish a correspondence between confidence intervals and NB prediction accuracy for the multi-class problem, and then proceed to investigate the viability of confidence intervals in the prediction calculation. We contribute a new variant to the family of naïve Bayesian classifiers that often out-performs standard NB for several stock and real-world datasets based on three separate evaluation metrics. Furthermore, we propose that our CI-augmented NB classifier can be used as a general replacement for NB, based on empirical observations that it "does no harm,"[1] meaning that the incorporation of CI does not significantly reduce classifier performance on datasets for which it does not improve it.

1.1.3 Glossary of Terms

We introduce several terms that will be used throughout this work which may have specific definitions in the field of machine learning or usage which we adopt by convention to reduce confusion.

attribute Attribute, feature, and attribute-value are all synonymous and refer to any datum other than class appearing in the training set and used during model creation or in the test set and evaluated during prediction.

conditional probability The probability of an attribute value, given the class.

conviction The term *conviction*, often qualified by either weak or strong, is used in lieu of confidence to denote the relative magnitude of a confidence interval in the sense

[1] Often attributed to the Hippocratic Oath, but originally from the Latin *primum non nocere*.

of smaller confidence intervals implying stronger conviction (lower variability) than larger ones. The term *confidence* is used to refer to both various *confidence intervals* (or CIs) and the confidence used when evaluating statistical relevance.

ensemble learning Machine learning techniques that train and test multiple classifiers, employing a (possibly weighted) voting algorithm to produce the final prediction.

normalize To scale the numbers in an array such that the sum of the normalized numbers is 1. This is often done to allow class predictions to be interpreted as percentages.

overfitting Term used to describe a ML algorithm that has learned its training set to the point that it fails to generalize well during testing.

prior Term referring to the probability of a class in the dataset.

1.2 Background Topics

In this section we introduce the Weka data-mining software and then review the mechanics of naïve Bayesian classification and the related topic of Laplace estimation before demonstrating a classification example.

1.2.1 Weka

Although not a direct subject of our research, the Weka machine learning and data-mining software [3] figures prominently in our work and should be afforded a brief introduction as we will refer to it throughout this thesis. Weka is an collection of machine learning algorithms and tools for data-mining tasks supported by the University of Waikato in New Zealand.[2] It includes tools for data pre-processing, visualization, and classification,

[2]It is also the name of a flightless bird endemic to New Zealand also known as a woodhen (*Gallirallus australis*).

in addition to other approaches to the ML problem including regression, clustering, and association rules. The toolkit reduces the barrier to entry to ML tasks by providing a large collection of implemented algorithms, integrated with GUI-based tools for interactive evaluation (the Explorer and KnowledgeFlow modules) and algorithm comparison (the Experimenter) within a Swing-based GUI. The Experimenter enables the evaluation and comparison of multiple ML algorithms as part of a single logical experiment, aggregating and comparing results from any number of individual classifiers. Command-line modes of interaction are also supported, including support for many ancillary tasks to ML research, *e.g.* attribute selection and filtering, discretization, and conversion between various data formats. The automation of the arduous and error-prone tasks of evaluation and statistical comparison (discussed further in Section 1.3) permits rapid investigation into strategies of classifier improvement, allowing us to cover a greater breadth of hypotheses. The software is written in Java and available as open source under the GNU General Public License, providing both an accessible source of information for implementation details of other researchers' code and a convenient and formidable library upon which to base our research. For these reasons, Weka is an ideal foundation for our work on confidence intervals.

1.2.2 Naïve Bayesian Classification

The basis of naïve Bayesian classification is Bayes' rule, shown in (1.1), which allows the computation of a conditional probability, $P(b|a)$, given the two class priors $P(a)$ and $P(b)$, and the conditional probability of $P(a|b)$.

$$P(b|a) = \frac{P(a|b)P(b)}{P(a)} \tag{1.1}$$

The rule is quite useful in a classification context where the datum sought is the probability of class c given a set of attribute values, X (an *instance*). The method

requires that the probability of each attribute value a_i relative to the class be known or estimated, and employs the product rule—that is, assumes conditional independence amongst the attribute values $P(a_i|c)$—resulting in the following formulation:

$$P(c|X) = \frac{P(c)\prod_i P(a_i|c)}{P(X)} \tag{1.2}$$

Because the machine learning classification task is supervised, both the probabilities $P(c)$, or *priors*, and the attribute probabilities $P(a_i|c)$ are easily determined simply by counting occurrences in the training set. For a NB classifier to make a prediction, the probability $P(c|X)$ for all possible classifications, $c \in C$, is calculated in order to find the maximum-likelihood classification hypothesis, often referred to as *maximum a posteriori* or *MAP* [1, p. 718].

For this computation the $P(X)$ terms are constant and therefore typically ignored. Thus the prediction task is reduced to computing Equation 1.3 for each instance in the test set. For more information about naïve Bayesian classification, Elkan provides concise overview in [4].

$$pred_X(c) = \underset{c}{argmax}\ P(c)\ \prod_i P(a_i|c) \tag{1.3}$$

1.2.3 Laplace Estimators

One potential complication when computing class probabilities is that one or more of the attribute probability terms, $P(a_i|c)$, is zero. This occurs whenever a specific attribute value does not appear in for the class c during the training of the classifier. The effect is to cause the predicted class probability of the instance, $P(c|X)$, to become zero, irrespective of either the class prior or other attribute probability terms. An approach frequently employed is to initialize all attribute value counts to a minimum of one (or another small constant), referred to in the literature as a *Laplace estimator* [3, p. 91]. In actual

implementations of NB, such as the `NaiveBayes` classifier found in Weka, the estimator follows slightly more complex rules for nominal datasets to ensure that probabilities are strictly in the range (0.0–1.0), to wit:

- When calculating the class priors for a given class ($P(c)$ in Equation (1.3)), the number of instance of the given class (the dividend) is taken as the actual number of instances of that class plus 1. For the overall count of instances in a dataset (the divisor), the estimator uses the number of instances plus the number of classes.

$$P(c) = \frac{n_c + 1}{n + |C|} \tag{1.4}$$

- When calculating the conditional probabilities for a given class, $P(a_i|c)$, the numerator is the count of instances of a specific attribute value increased by 1, divided by the count of all instances of that attribute (given the class) plus the number of values the attribute can assume given the class.

$$P(a_i|c) = \frac{n_{a_i}(c) + 1}{n_a(c) + |A_{i_c}|} \tag{1.5}$$

Viewed in the limit as $n \to \infty$, the estimator terms are insignificant. However, for datasets with few instances relative to the number of attributes or classes, the use of a Laplace estimator has a potentially disproportionate effect on the class priors and conditional probabilities. Specifically, when $n_c = 0$ or $n_{a_i(c)} = 0$, the probability of such features is assumed to be uniformly the smallest representable probability in the model. In practice the values of n are sufficiently large relative to the class and attribute value counts so as not to pose a practical issue, and teaching machines to learn very small problems is not a worthwhile pursuit anyway. However, the use of the estimator may detract somewhat from our intuitive sense of the probabilities when working out small illustrative problems by hand, as we will observe in the following example.

1.2.4 Example NB Calculation

We perform a simple multi-class prediction using a naïve Bayesian classifier with a Laplace estimator to illustrate the procedure. We train our classifier with four instances of the Boolean logic functions XOR, AND, and NOR as per Table 1.1 and observe the state of the model after training in Table 1.2. The effect of the Laplace estimator on such a small training set is evident. For example, we would expect the class prior of XOR to be 2/4 (2 of 4 instances), but instead it is 3/7 as per Equation (1.4). The conditional probabilities for each attribute value are affected by the estimator as well; attribute values such as $p(x = 0|AND)$, which we would compute to be 0 from the training data becomes 1/3. By Equation (1.5), the numerator contains 0 instances where the class is AND and $x = 0$ plus the constant 1, and the denominator enumerates 2 instances for the class AND plus 1 possible value of x given the class AND.

x	**y**	**Class**
0	1	xor
1	0	xor
1	1	and
0	0	nor

Table 1.1: Naïve Bayes Example Dataset

	Priors	**Conditional Probabilities**			
		x=0	x=1	y=0	y=1
xor	3/7	2/4	2/4	2/4	2/4
and	2/7	1/3	2/3	1/3	2/3
nor	2/7	2/3	1/3	2/3	1/3

Table 1.2: Naïve Bayes Example Model After Training

We can now take an instance, say $x = 1, y = 0$, and apply Equation (1.2) by looking up the values in the model (Table 1.2). For each class, we compute the predicted probability

as demonstrated in Table 1.3. We then apply Equation (1.3) to the values in the *raw result* column to make a prediction of the correct class XOR. Predictions for the other possible values of x and y also yield the expected class.

Prediction	$p(c) \times p_{x=1}(c) \times p_{y=0}(c)$	**Raw Result**	**Normalized Result**
$p(xor\|x = 1, y = 0)$	$3/7 \times 2/4 \times 2/4$	0.1071	0.4576
$p(and\|x = 1, y = 0)$	$2/7 \times 2/3 \times 1/3$	0.0634	0.2712
$p(nor\|x = 1, y = 0)$	$2/7 \times 1/3 \times 2/3$	0.0634	0.2712

Table 1.3: Naïve Bayes Example Prediction

1.3 Performance Evaluation

If we were to test our NB classifier model with an instance of each of our 3 functions, we would observe that it is able to predict instances of each correctly, *i.e.* it would have an accuracy of 100%. Admittedly, this is not overly impressive as the training set and test set are identical, and we have done a moderate amount of counting and floating point math to replicate the functions of a few basic logic gates. However, we use this opportunity to introduce the topic of how classifiers are evaluated in the supervised learning context. The first item to address is the notion that we are not really learning anything if we are testing using the same data used to train the model. After all, we could have implemented a lookup table just as easily (although we would not be able classify instances with unobserved attribute values). The technique of *cross-validation*, in conjunction with the Student's t-test, addresses this concern.

Next we then turn to metrics of evaluation to the end of comparing one classifier to another. In [5], Ferri, Hernández and Modroiu compare 18 classification performance measures and group them into 3 separate families: those based on a threshold and a *qualitative* understanding of error, those based on a *probabilistic* understanding of error,

and those based on how well the model *ranks* instances. Their paper provides a good overview of the metrics available and then goes on to discuss correlation between metrics within families. In our work, we select a metric from each family, *accuracy*, *root mean-squared error* (*RMSE*), and *area under the curve*, or *AUC*. These are introduced in turn following the discussion of cross-fold validation.

1.3.1 Cross-fold Validation

"The standard way of predicting the error rate of a learning technique given a single, fixed sample of data is to use stratified 10-fold cross-validation" [3, p. 150]. *Cross-validation* (CV) is the technique of partitioning the dataset into n approximately equally sized partitions, or *folds*. Each fold of becomes a test set, and the remaining instances comprise its corresponding training set. The machine learning algorithm is trained and evaluated for each fold and the results averaged to determine the overall accuracy of the algorithm. *Stratification* is a variant upon partitioning where the relative class frequencies present in the dataset are maintained within each fold. Another variant of cross-validation is *leave one out*, often seen in the literature as *LOOV*, in which the number of folds equals the number of instances. For a dataset with k instances, there are k folds, each comprised of a test set of one instance and a training set of $k - 1$ instances. This technique can be computationally expensive for large datasets, and according to the evaluation by Kohavi [6], standard deviation of NB classification accuracy does not improve significantly in k-fold validation once $k > 10$. 10-fold cross-validation is used prevalently for algorithm evaluation in machine learning research and is used in our work unless noted.

An alternative to cross-validation is *bootstrapping* or *0.632 bootstrap*, wherein a training set of size n is constructed from a dataset (also of size n) by random sampling with replacement. Due to the random sampling, there will be some repetition in the training set; any instance not selected for training is added to the test set. The probability of an in-

stance not being selected for the training set is $(1-1/n)^n \approx e^{-1} = 0.368$, leaving $(1-1/e)$ (or 63.2%) of the instances to appear in the training set) [3, p. 152]. This process is done multiple times and the results of each evaluation averaged. "The bootstrap procedure is good at estimating [classifier] error for very small datasets. However, like [LOOV] it has disadvantages" that in some situations make it "misleadingly optimistic" [3, p. 153].

Another standard technique is to evaluate a learner's performance across multiple cross-fold validations, each initiated with a different random seed, in order to minimize the potential random bias of any one of the cross-fold validation runs [3, p. 151]. The mean of those runs, again typically numbering 10, is taken as the performance of the learning method. The *Student's t-test*, or simply *t-test* is used to determine whether the mean of a set of samples is significantly greater or less than the mean of another [3, p. 154]. Specifically, we use Weka's paired corrected t-test unless otherwise noted to determine whether a classifier's performance is statistically different than another's after 10 times 10-fold cross validation. The significance threshold used for the t-test is 0.05.

Despite describing a procedure for constructing and evaluating 100 individual models per learning method, we have not yet declared what value it is we are averaging—*i.e.* what metrics do we use to compare the relative merit of learners to each other, and how do we visualize classifier results?

1.3.2 Accuracy and the Confusion Matrix

The *confusion matrix* is a tabular representation of the predictions of a nominal machine learning classifier. It has one row and one column for each class in a nominal classification problem; typically rows represent the predicted classes while columns represent the actual classes. At the intersection of each row and column are the counts of test instances that meet the row and column criteria when the model is evaluated (*i.e.* predicted) by the classifier. For a 2-class problem, the general form of the matrix is shown in Table 1.4.

		Predicted Class	
		a	**b**
actual class	**a**	true positive (TN)	false negative (FN)
	b	false positive (FP)	true negative (TN)

Table 1.4: Confusion Matrix for 2-class Problem

Couched in the nomenclature of the confusion matrix, the *accuracy* for a 2-class learning problem is calculated via Equation 1.6, although it is equally valid to think of accuracy as simply the ratio of the correct predictions over the total number of instances.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1.6}$$

Therefore accuracy is a unit-less scalar that ranges between 0–1 that can be used to compare ML classifiers directly. It benefits from an immediately accessible interpretation as the percentage of correctly identified instances in the test dataset—*i.e.* as a grade—and is very often expressed as a percentage between 0 and 100%.

Multi-class problems extend the dimensions of the matrix to $k \times k$ for a k-class problem. The 4×4 confusion matrix output for a 4-class problem appears in Table 1.5. The layout varies somewhat from the example 2-class problem merely as a matter of convention; the layout in Table 1.5 is similar to the layout used by Weka. In this example the classifier was unable to correctly classify any of the test instances of the class d (*secondary hypothyroid*), instead predicting these instances as belonging to class a (*negative*).

a	**b**	**c**	**d**	← **classified as**
3470	2	9	0	**a** = negative
14	180	0	0	**b** = compensated hypothyroid
5	0	90	0	**c** = primary hypothyroid
2	0	0	0	**d** = secondary hypothyroid

Table 1.5: Example Confusion Matrix for 4-class Problem

Accuracy in the multi-class case is calculated by taking the ratio of the sum of the diagonal of the confusion matrix to the sum of all entries in the matrix. For this example, *hypothyroid* from the UCI dataset [7], the accuracy is 3740 over $3740 + 32$ (the diagonal plus the non-diagonal terms), for a result of 99.15%. Referring back to Equation (1.6), obviously the concepts of true and false positives and negatives must be grounded in an evaluation of the predictor *for a particular class*, echoing back to the MAP evaluation for NB discussed in Section 1.2.2. That is, true positives do not appear along the entire diagonal of the confusion matrix; they only appear in a single cell along the diagonal for class being evaluated. Similarly, it is nonsensical to consider false positives in any other column save this same column. Evaluating accuracy for a multi-class problem is thus equivalent to weighting the accuracy for each of the k classes by the number of instances in that class, and then dividing by the total number instances.

The qualitative nature of accuracy is quite evident in this formulation. Any correct prediction is a good prediction, or true; any incorrect prediction is false. However, recall that the NB classifier not only predicts the class. The individual class predictions in Equation (1.3) each have a real-valued number associated with them. Another common practice is to take the predicted class probabilities and scale them such that they sum to 1. This is referred to as *normalizing* the predictions so they can be interpreted as percentages, and the task is used for other values in our work as well.

1.3.3 Root Mean-Squared Error (RMSE)

Root mean-squared error is an other scalar metric from the probabilistic family as per Ferri, Hernández and Modroiu [5] which measures the deviation from the true probability. "These measures are especially useful when we want an assessment of the reliability of the classifiers, not measuring when they fail but whether they have selected the wrong class with high or low probability." Witten and Frank [3] point out that RMSE is typically used

to evaluate numeric predictions (*regression*), but include it in their evaluation of nominal classifiers. The standard equation is given in Equation (1.7), where p_i is the predicted value, a_i the actual value, for n instances. Equation (1.8) extends this for k classes. In the context of a nominal predictions, the actual value is taken to be 1 for the actual class of the instance and 0 for all other classes. The p_i terms are the normalized prediction probabilities introduced in Section 1.3.2.

$$RMSE = \sqrt{\frac{\sum_i (p_i - a_i)^2}{n}} \tag{1.7}$$

$$RMSE_{nominal} = \sqrt{\frac{\sum_i \sum_k (p_{i_k} - a_{i_k})^2}{nk}} \tag{1.8}$$

In order for a nominal classifier to achieve a perfect RMSE of 0, it would not only have to be 100% accurate, but also assign a prediction probability of 100% to the correct class for each instance and 0% to any other classes. We know from the Laplace estimator discussion that 0% predictions will not occur (except in the case of underflow or rounding due to limited precision), so clearly the RMSE can only asymptotically approach the ideal. However, we may consider a classifier that assigns a probability of 95% to a true positive to perform better than one that assigns a probability of 51%, hence we include the metric. We note however that for the multi-class problem the situation is more pronounced because the "winning" prediction probability may only slightly exceed $1/k$ for the k-class problem. For this reason we take care to only compare RMSE metrics between classifiers evaluating the same datasets.

In Table 1.6 we demonstrate a sample calculation of RMSE from our 3-class NB example (Table 1.3) for the first (XOR) prediction. As you can see, there is error despite predicting the class correctly. We interpret lower error values as indicators of greater certainty in the classifier's predictions.

Predicted (normalized)	**Actual**	$\lvert actual - predicted \rvert$
0.4576	1.0	0.5424
0.2712	0.0	0.2712
0.2712	0.0	0.2712
$RMSE_{x=1,y=0} = \sqrt{\frac{0.5424^2+0.2712^2+0.2712^2}{1\times 3}} = 0.3835$		

Table 1.6: RMSE Example Calculation

1.3.4 Area Under the ROC Curve (AUC)

AUC (area under the curve) of the *Receiver Operating Characteristic* (ROC) graph is another metric frequently used to compare machine learning algorithms. The ROC curve plots the rate of true positives versus false positives by evaluating the ranked predictions of the classifier. The plotting is performed on a standard Cartesian $n \times n$ graph, where n is the number of test instances, although the x and y axes are typically labelled as percentages of the total number of attributes. Ranking involves sorting predictions by their predicted class probabilities in descending order and then evaluating each prediction to determine whether it is correct. Correct predictions increment the y-coordinate whereas incorrect predictions increment the x-coordinate.

A classifier that ranks well will accumulate true positives "more quickly" (as a rate with respect to the number of instances) than false positives. When expressed as percentages, the rates and therefore the axes range between 0–1. The AUC metric is the area under the curve constructed when the points on the graph are connected with line segments; by convention a line segment connects the last prediction with the point 1,1. Thus the AUC also ranges between 0–1, where higher values indicate better classifier performance. The ideal graph is simply a vertical line from the origin and forms a unit square with an AUC of 1.

These rates are affected by the order in which instances are evaluated (hence the reason for using ranked predictions), and they also depend on the training and test mixture.

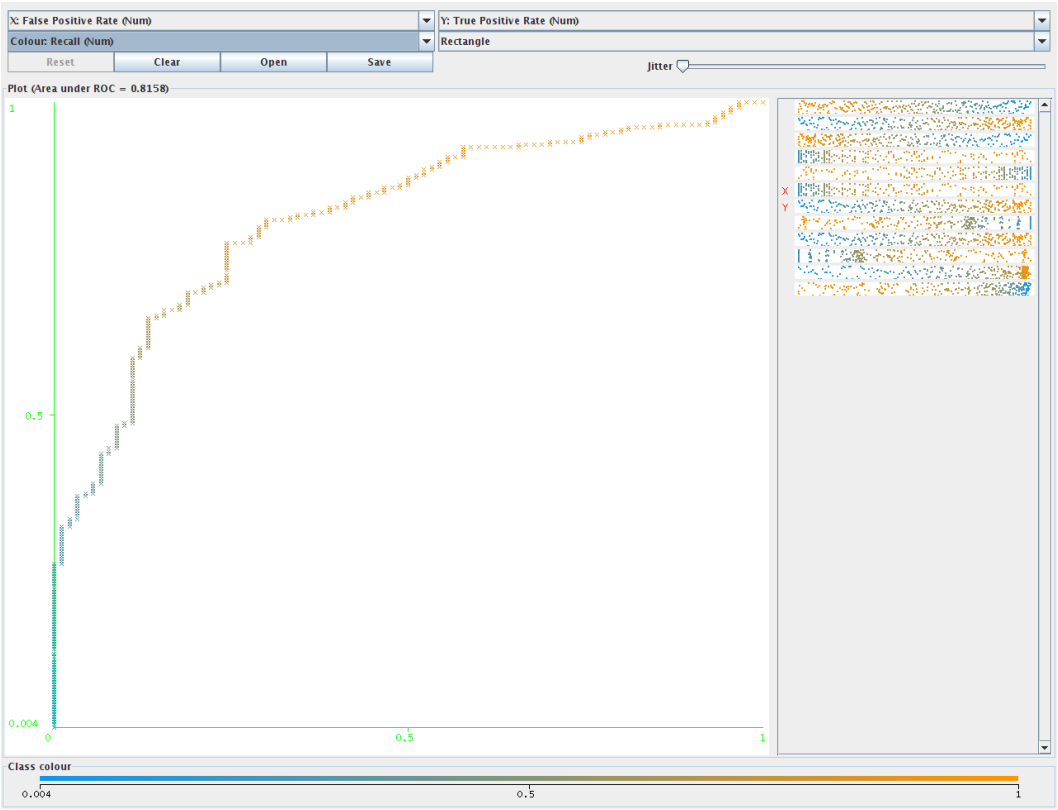


Figure 1.1: ROC Curve for Credit-G Dataset

For this reason, the ROC curve is typically generated taking the average of the ranked prediction after several rounds of cross-validation [3, p. 169]. Because the AUC is a scalar, it can be used like accuracy and RMSE to assess relative classifier performance. An example depicting an ROC curve is show in Figure 1.1. The AUC for this classifier is 0.8158.

For multi-class problems, the definition of AUC must be expanded because each class has its own concept of true or false positive/negative classification results. To depict this in 2 dimensions requires k AUC curves, one for each class c_k, where TP and TN indicate the that instance truly belongs to class c_k, and instances of all other classes are treated as negatives. In the 3-class case one intuitive approach is to extend to 3 dimensions and calculate the volume under the surface (VUS), which like AUC is unit-scaled and provides

a metric for comparison. The general k-class case can only be visualized piecewise using projections and loses its intuitive interpretation. Weka employs the weighted average approach taken by Provost and Domingos in [8] and depicted in Equation (1.9), and hence our work uses this formulation of AUC as well. In [9] Fawcett notes that Provost and Domingo's approach has the disadvantage of being sensitive to class distributions—*i.e.* the AUC of classes with higher prior probabilities are weighted more heavily. An alternative proposed by Hand and Till [10] found in the literature is to compute the multi-class AUC by calculating the AUC for each class pairwise with respect to every other class and sum these terms and scale by $2/(k(k-1))$ to give a measure they call M. The advantage of this approach is that it results in the pairwise discriminability between classes, but the disadvantage is that it is quadratic in the number of classes (there $k(k-1)/2$ such pairs of classes) and, according to Fawcett, there is no easy way to visualize the surface whose area is being calculated.

$$AUC_{total} = \sum_{c_i \in C} AUC(c_i)\dot{p}(c_i) \tag{1.9}$$

We note that performance evaluation and AUC in particular are areas of active research [11, 12]. For binary classification problems there are 4 different types of AUC according to Vanderlooy and Hüllermeier [13]. The binary AUC we first introduced is equivalent to the Wilcoxon-Mann-Whitney statistic. According to the work in [14, 15], AUC is a better indicator of classifier performance than accuracy. This claim is based on an increased sensitivity to in ANOVA (analysis of variance), independence with respect to the decision threshold (meaning that any correct prediction is preferred), and is invariant to *a priori* class distribution in the dataset (although this only holds true for binary class problems). Finally, Fawcett provides a good academic overview of both binary and multi-class ROC analysis in [9].

1.3.5 Other Metrics

For accuracy, RMSE and AUC, the relative merit of a correct or incorrect prediction is considered equal—that is, they carry the same weight, and this across all classes. In some domains, the cost of an incorrect prediction (false positive or false negative) differs from the cost of a correct prediction (true positive or true negative). Examples include the field of information retrieval, where emphasis is placed on the true positive rate (as compared to the true negative rate; after all, knowing the number of documents that truly do not match a query is not very useful to a search engine user), and medicine, where the predictive power of a diagnostic test is relative to the overall number of cases that go either detected or undetected, not necessarily the overall patient population. Several classifier metrics frequently encountered in domain-specific literature are introduced here briefly.

precision The ratio of relevant results to the overall number of results classified as true.

$$precision = \frac{TP}{TP + FP} \tag{1.10}$$

recall The ratio of relevant results to the number of relevant instances in the corpus.

$$recall = \frac{TP}{TP + FN} \tag{1.11}$$

F-measure (also known as *F-score* or F_1) The harmonic mean of precision and recall, this weights precision and recall equally in terms of importance (cost).

$$\text{F-measure} = \frac{2 * precision * recall}{precision + recall} \tag{1.12}$$

sensitivity Common terminology in the medical field equivalent to recall.

specificity In medicine, the ratio of patients without a condition who receive a negative test result to the overall number of patients without the condition.

$$specificity = \frac{TN}{TN + FP} \tag{1.13}$$

Although the examples above are typically used for 2-class problems (either a document is relevant or irrelevant to a search; either a patient diagnosis is correct or incorrect), the concept of cost is readily extended to multi-class domains. For an overview of cost-based classifier evaluation, see [3, Section 5.7]. In our work, we compare the accuracy, RMSE, and AUC of classifiers using 10 times 10-fold cross-validation for the general case, irrespective of cost.

1.4 Related Work

Although not universally accepted as detrimental [16, 17], one of the primary criticisms of naïve Bayes is that the assumption of truly independent conditional attributes probabilities is often incorrect (even "grossly violated") in real-world datasets, and that this harms performance [18, 19, 20]. For this reason, a classifier should somehow account for feature interaction. Various approaches exist for improving the classification performance of naïve Bayes based on this assumption; these can generally be categorized into two groups. The first is feature selection, where irrelevant, noisy, or redundant attributes are removed from the dataset, improving the classifier's performance and reducing its runtime and memory requirements. The second is the combination of NB with other machine learning algorithms, which we term *hybrid* or *structural* variants of NB.

In [21] Jiang *et al.* posit four general categorizations, the two above, and a third utilizing local learning (citing NBTree as an example) and a fourth referred to as *data expansion*. We assert that NBTree [22] can be considered a structural approach, blending

the structure of a decision tree with a forest of NB classifiers, each trained on attributes remaining after traversal of the decision tree. Their fourth approach describes a lazy learner algorithm that expands the training set with clones of existing instances based on the similarity of the test instance to training instances. This can also be considered a hybrid approach in which a nearest neighbor algorithm is used to detect similarity, and then bootstrapping or another commonly used ensemble learning technique adjusts the instance counts in the NB model. We briefly examine each approach to establish a categorization for our algorithms based on attribute and instance confidence intervals.

1.4.1 Feature Selection

Feature selection comes in two forms, *filters* and *wrappers* [23]. At their core, both appeal to Ockham's razor in that equivalent or improved classifier performance on the training set using fewer attributes should result in less overfitting and therefore increased generality of the learned hypothesis when the model is applied [24]. Features may be filtered based on metrics such as the covariance of attribute values. Filters select attributes prior to construction of the model and so are independent of the algorithm used for machine learning. Wrappers, by contrast, select attributes based on their effect on the machine learning algorithm's performance. Attributes are added (or removed) and then the model is built and evaluated. Each evaluation is compared to those prior, resulting in a search for the optimal feature set. Both filters and wrappers can be computationally intensive, the former because it involves search across all features in the training set (and its complexity therefore expressed in terms of the number of attributes), and the latter because it requires search, each step of which necessitates construction and evaluation of the model.

Related to feature section (and an idea that we will attempt to incorporate into our CI-based classifiers later) is emphasizing and deemphasizing features so that the relative influence of an attribute on the prediction depends upon its weight. Zhang and Sheng's

work on weighted naïve Bayes [25] modifies attribute weights by exponentiation. Their task is to learn an optimal (or at least improved) set of attribute weights during training that result in improved performance.

1.4.2 Hybrid Variants

Structural variants have received considerable research attention, many with implementations available directly within Weka. In addition to NBTree, in which decision trees employ NB classifiers as leaves, we introduce several others here. Many of them apply to Bayesian networks, which are a generalization of the flat naïve Bayesian classifier that is our primary focus. As discussed in [26], learning an optimal Bayesian network classifier is NP-Hard, hence the research interest in heuristic approaches.

- Hidden Naïve Bayes (HNB) modifies the structure of the learner by creating a (hidden) parent for each attribute to construct a Bayesian network that accounts for the influences of other attributes [27].

- DMNBtext [28] (Discriminative Multinominal Naïve Bayes) is another Bayesian network learner geared towards text classification that employs a discriminative learning technique to establish and refine parameters (frequency estimates of the conditional probabilities of attribute parents) during model creation.

- AODE (Averaged One-Dependence Estimators) "achieves highly accurate classification by averaging over all of a small space of alternative naïve Bayes-like models that have weaker (and hence less detrimental) independence assumptions than naïve Bayes. The resulting algorithm is computationally efficient while delivering highly accurate classification on many learning tasks" [29].

- Classified as a rules-based classifier in Weka, Hall and Frank's DTNB [30] (decision tree/naïve Bayes hybrid) is an example of a classifier that incorporates a number of

techniques into one algorithm, including filtering, a wrapper, and a hybrid structure. The authors describe their classifier as follows:

> At each point in the search, the algorithm evaluates the merit of dividing the attributes into two disjoint subsets: one for the decision table, the other for naïve Bayes. A forward selection search is used, where at each step, selected attributes are modeled by naïve Bayes and the remainder by the decision table, and all attributes are modelled by the decision table initially. At each step, the algorithm also considers dropping an attribute entirely from the model.

1.4.3 The m-Estimate

Through the course of our research we did not encounter other works where confidence intervals are used directly as an attribute for NB classification. However we note the potential similarity between the m-estimate, which is a parameterized alternative to the Laplace estimator, and confidence intervals. Updating the equations in Section 1.2.3, we include the parameter m and a term p which is the prior estimate of the probability being calculated yields:

$$P(c) = \frac{n_c + mp}{n + m} \tag{1.14}$$

$$P(a_i|c) = \frac{n_{a_i}(c) + mp}{n_a(c) + mp} \tag{1.15}$$

In [31] Džeroski *et al.* offer three interpretations of m. First, it is described as "inversely proportional to the initial variance... meaning the higher the value of m the lower the initial variance and [thus] the initial expectation of p. Second, it controls balance between the relative frequency and prior probability. Finally, m can be set to correspond to the

level of noise in the data." It is the connection to variance we believe relates to our work. Their work varies m from 0–999 and then compares the performance of an algorithm using Laplace estimator with the best value found for m.

In [19] Jiang *et al.* fix $m = 1$ and apply the m-estimate to four NB classifiers and compare them on the basis of accuracy. They observe improvements across a subset of the UCI datasets; their work differs from ours in that m is parameterized and is applicable to both binary and multi-class problems, whereas our confidence interval metrics only affect prediction in the multi-class context.

Chapter 2

Confidence Intervals

2.1 Nomograms and Orange

Before delving into the mathematical definition of confidence intervals, let us first relate the inspiration for their use in the context of classification. It begins with the *nomogram*, a type of graph first attributed to the French mathematician Maurice d'Ocagne in 1899[1] and which was used during World War I to direct anti-aircraft fire [32]. The nomogram is essentially a two-dimensional plot of a function where both the function's range and the interrelation between its parameters and their domains are depicted such that the result of the function calculation can be read directly from the graph. Unlike standard Cartesian coordinates, nomograms can represent functions of more than two variables, and provide approximate answers very quickly, similar to the operation of a slide rule. Nomograms are quite prevalent in the medical profession and also in engineering, with the most common example being the Smith chart used to calculate impedance over transmission lines. In addition to allowing the user to calculate an answer, they also aid in the visualization of the function. To illustrate the nomogram, we refer the reader to Figure 2.1.[2]

[1] *Traité de nomographie: Théorie des abaques, applications pratiques* (Paris: Gauthier-Villars, 1899)
[2] Source: PyNomo.org

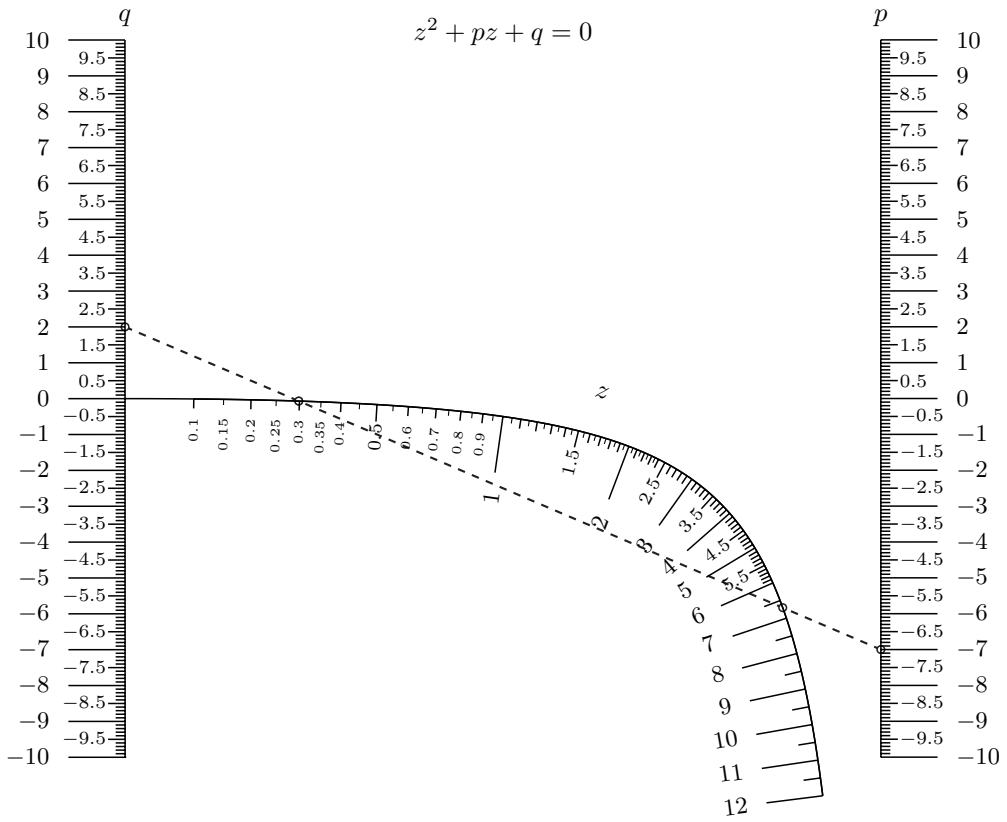


Figure 2.1: Nomogram Depicting a Second Order Equation

Enter Orange, a open source component-based machine learning library for Python developed by the Laboratory of Artificial Intelligence, Faculty of Computer and Information Science, University of Ljubljana, Slovenia [33]. Orange's authors set out to create software similar in purpose to Weka, but also to demonstrate their research [2, 34] on the topic of visualization of classification problems based on nomograms.[3] Specifically, their graphical representations depict attribute values using either a log-odds ratio or points scale and allow one to visualize the prediction directly from the nomogram. Furthermore, their nomograms include confidence intervals expressed as error bars, and it is those error bars that led us to ponder the question of whether these confidence intervals could be used as part of the classification process. To wit, the error bars potentially represent

[3]Strictly speaking, Orange does not produce traditional nomograms, but its interface captures the spirit and expressivity of the nomogram in a context not given to visualization.

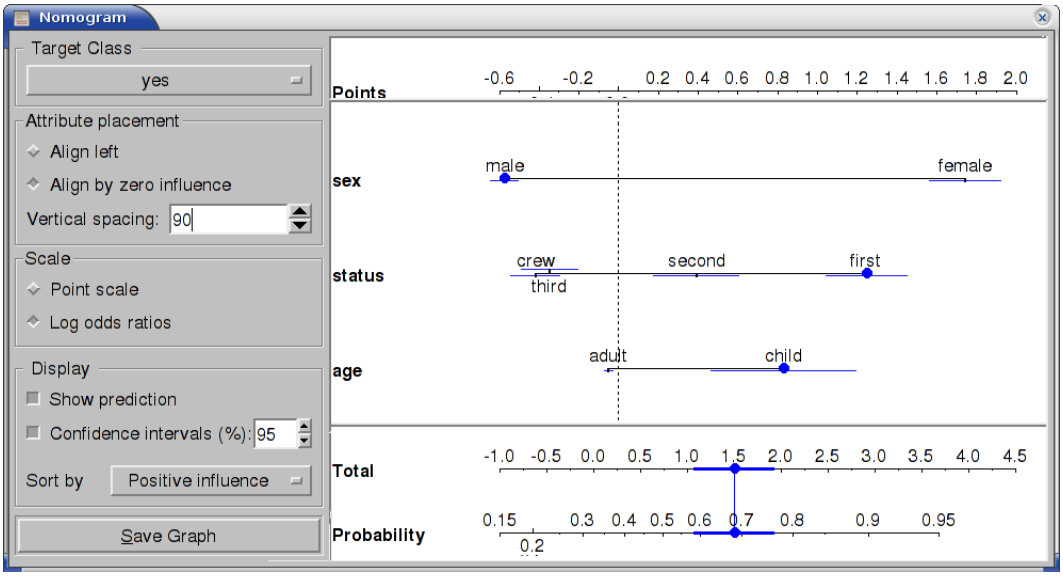


Figure 2.2: Nomogram of NB Classification and Confidence Intervals in Orange

information that could be incorporated into the ML process. Figure 2.2 is Orange's representation of a binary classification problem of three attributes. You can "read" the nomogram by adding up the points assigned to an attribute and then use the probability scale to ascertain the (normalized percentage) result of the class prediction.

2.2 Confidence Interval Calculations

We now turn from nomograms to confidence intervals and how to calculate them. As noted previously, the probabilities used in Equation (1.3) are derived by counting occurrences that occur in the training set, and as such are necessarily merely estimates of the true probabilities. Because counting in involved, there is an estimated error, or *confidence interval*, associated with each probability. The smaller the sample size, the larger the confidence interval, which we interpret as the greater the likelihood that the estimated probability varies from the calculated value.

Related to (but not to be confused with) this calculation is the *confidence*, sometimes referred to in statistics as *confidence level* or simply α, with which the confidence interval

is calculated. This term assumes a normal distribution for the confidence interval and has the effect of scaling it. For our experiments we use a confidence level of 95%, which corresponds to a scaling factor of $z_{\alpha/2} = 1.96$ [35, p. 613]. A higher confidence, for example 99%, would result in a scaling factor of 2.576. One can think of the $z_{\alpha/2}$ as the coefficient in order for the area under the normal distribution curve at α to be 1. The confidence will also appear as $z_{(1-\alpha)/2}$, which is equal to $z_{\alpha/2}$.

This term appears in Equation (2.3), taken from the section on confidence intervals in the Možina *et al.* paper regarding nomograms for NB classification [2]. We vary our nomenclature slightly from that paper in favor of more mnemonic variable names with which to refer to the various confidence interval calculations, but acknowledge their work as the source of these formulae.

The first of these is the confidence interval calculation for an individual attribute value, which we refer to as aci_{a_i}, or *attribute CI*. Shown in (2.1), this value is calculated once for every attribute value and class found in the training set:

$$aci_{a_i}(c) = \sqrt{\frac{1}{N_{a_i} P_{a_i}(c) P_{a_i}(\bar{c})} - (ci_c)^2} \tag{2.1}$$

The subtrahend under the square root, ci_c, is the confidence interval associated with the class itself (and corresponds to $\widehat{Var}(logit\widehat{P}(c))$ in [2]). It is calculated once per class from the size of the training set and priors, as shown in (2.2):

$$ci_c(c) = \sqrt{\frac{1}{NP(c)P(\bar{c})}} \tag{2.2}$$

With these values calculated for a training set and stored in our classifier model, we can calculate yet a third confidence interval that is associated with the attribute values of a test set instance. The expected error, or *prediction CI* for a training instance X is depicted in (2.3). This calculation is taken from the portion of the Orange [33] source code that calculates the width of the confidence interval error bars for the nomogram.

$$pci_X(c) = \pm z_{1-\alpha/2} \sqrt{\sum_{i=1}^{|A|} aci_i(c)^2 + ci_c(c)^2} \tag{2.3}$$

Please refer back to Figure 2.2 to observe an example of the confidence intervals associated with attribute values and with the overall prediction. This nomogram represents the probability of survival of a male child in the first class on the Titanic being predicted using naïve Bayes. The error bars can be seen just below the attribute values. The wider the error bar, the higher the $aci_{a_i}(c)$, *i.e.* the less confident we may choose to be about the attribute, which we refer to as weak conviction. In this particular dataset, there are far fewer children than adults, hence the difference in width of their error bars. At the bottom of the graph, the confidence interval associated with the prediction itself, $pci_X(c)$ can be seen to range between 59–77%, or $68 \pm 8\%$. Because the dataset is a binary classification problem, either passengers are predicted to survive or not. Therefore the confidence intervals associated with the attributes and prediction for class *yes* (survived) are the same as those for class *no* (perished). This follows from equations (2.1) and (2.2) because it is necessarily the case in binary class datasets that $P(yes) = P(\bar{no})$. Multi-class class datasets do not exhibit this property, and so their confidence intervals become potentially more interesting from a classification perspective.

One potential source of consternation is that the confidence interval formulas used in Orange do not match up with those found in other resources for standard margin or error or confidence interval calculations. Because our exploration of the utility of confidence intervals is done using the CI found in Možina's work we do not dwell on this topic further at this point. Instead, we launch into our investigation of confidence interval and its correlation to naïve Bayesian prediction.

2.3 Experiment I: CI Correlation to Prediction

The notion of a confidence interval associated with dataset features and classifier prediction is intriguing, at least intuitively, because it would seem to offer some indication as to how much a prediction made using those features might be trusted. The question then becomes whether that intuition has any basis, and if so are there any practical applications of this "extra" feature? Our initial experiment is thus divided into 2 related parts. First, is there any relationship between prediction confidence intervals and prediction accuracy? That is, if we compute prediction CIs for each class and then compare those CI for correct predictions (*hits*) to incorrect predictions (*misses*), is there any observable pattern? The remainder of this section addresses this question, describing our extensions to Weka to compute CI and the subsequent analysis. In Section 2.4 we begin to tackle the follow-on question of whether any such patterns can be put to use.

2.3.1 Extending Weka to Incorporate CI

Our initial approach was to implement a log-odds ratio based NB classifier in Weka, taking the same approach to classification taken in Orange. Such an implementation computes the sum of log-odds ratios instead of the product of probabilities, and then exponentiates to calculate the prediction probability. Elkan covers this formulation in his introduction to naïve Bayes [4]. This implementation was completed, but abandoned due to minor discrepancies between prediction results for this new module, `NBLor`, and Weka's `NaiveBayesSimple` classifier, which we use as our baseline. That is, we felt that it was vital to ensure that comparisons were being made between like quantities. (In hindsight, we suspect that the inconsistencies stem from differences in the implementation of the Laplace estimator.) Instead, we modified Weka's `NaiveBayesSimple` classifier to calculate the attribute and prediction confidence intervals discussed in Section 2.2 during training, and then extended the output format to emit the prediction CI along with the class

probability during classification.

This new classifier, `NBCi`, was evaluated using the Experimenter to verify that no changes to model generation or prediction were introduced through the modifications. Similarly, the CI terms for attributes and predictions were compared with the values generated by an instrumented version of the Orange software to ensure correct operation. This manual verification step was done for 3 standard machine learning datasets: *titanic* (binary), *iris* (3 classes), and *zoo* (7 classes); introduced in Section 2.3.2. During the verification of prediction CI, we discovered another aspect of the CI calculations preformed in Orange, that being the notion of an "infinite attribute CI." When the value of the CI for an attribute cannot be calculated using Equation (2.1), either because there are no instances of the attribute-value in the training set or because the attribute probability is 0 or 1, the attribute CI is arbitrarily set to 99.99. Unlike the calculation of class and attribute prior probabilities, no Laplace estimator is used for the attribute CI calculations.

2.3.2 UCI Datasets

The evaluation was conducted using 37 classification datasets from the UCI Machine Learning Repository [7], which are prevalent in the ML literature. 17 of these are binary classification problems, while the 20 multi-class sets range from 3 to 26 classes, with an average number of classes of 8.65. The source datasets contain a mixture of nominal, integer, and real-valued attributes. Because our confidence interval calculations are defined only for counts of nominal attributes, all datasets are discretized using Weka's discretization class.[4] The datasets range from as few as 24 instances for *contact-lenses* to 20000 for *letter*, with an average of slightly more than 1700 instances. The UCI datasets are listed in Table 2.1 for reference. The table is divided into sections for multi-class and binary-class datasets; the latter are used in our early experiments that employ class and

[4] `weka.filters.supervised.attribute.Discretize`

attribute CI—prediction CI only varies for multi-class problems.

2.3.3 Generating NBCiDisplay Output

With `NBCiDisplay`, a classifier extended from our `NBCi` class, we generated predictions with confidence intervals for all 37 datasets. The predictions use 10-fold cross-validation (CV), *i.e.* 90% of the dataset was available for training and 10% for test. By default, Weka performs stratification during CV, so except in the case of datasets with very few instances, training and test sets retain ratios of the class and attribute values consistent with those found in the full dataset. The output format of the extended classifier is tabular and lists the index of the actual class, predicted class, number of classes, and then a list of class probability and CI tuples for each class prediction. Class probabilities are normalized between 0–1, as is customary within Weka, while the CI values are the raw prediction CI as calculated by Equation (2.3). Sample output for 5 test instances in the *iris* dataset are shown in Table 2.2. Note that the first 2 predictions are correct because the classes in the first 2 columns are equal, but in the third prediction, the classifier failed to predict the actual class b, instead predicted c with a probability of over 61%. The confidence interval for this prediction was slightly higher (weaker conviction) than for the actual class, but both of these were small relative to the CI for the first class prediction (to which the classifier assigned a predicted probability of nearly 0).

2.3.4 Correlation of CI with Prediction Accuracy

Once these raw data had been prepared, the next question was how to compare prediction CIs for correctly and misclassified instances, which we refer to *hits* and *misses*, respectively. We travailed several false starts in this area of our research. The goal of locating a correlation led us to perform several "what if" analyses employing spreadsheets. We document the results of two of those in this section.

Dataset	Classes	Attributes	Instances
anneal	6	39	898
audiology	24	70	226
autos	7	26	205
balance-scale	3	5	625
contact-lenses	3	5	24
glass	7	10	214
heart-c	5	14	303
heart-h	5	14	294
hypothyroid	4	30	3772
iris	3	5	150
letter	26	17	20000
lymphography	4	19	148
primary-tumor	22	18	339
segment	7	20	2310
soybean	19	36	683
splice	3	62	3190
vehicle	4	19	846
vowel	11	14	990
waveform	3	41	5000
zoo	7	18	101
breast-cancer	2	10	286
breast-w	2	10	699
colic	2	23	368
credit-a	2	16	690
credit-g	2	21	1000
diabetes	2	9	768
heart-statlog	2	14	270
hepatitis	2	20	155
ionosphere	2	35	351
kr-vs-kp	2	37	3196
labor	2	17	57
mushroom	2	23	8124
schizo	2	15	340
sick	2	30	3772
sonar	2	61	208
titanic	2	4	2201
vote	2	17	435

Table 2.1: UCI Datasets

Actual	Predicted	pred_a		pred_b		pred_c	
Class	**Class**	**pct**	**pci**	**pct**	**pci**	**pct**	**pci**
c	c	0.00017	277.18878	0.00960	2.27939	0.99023	2.27727
b	b	0.00001	277.19177	0.99781	2.28350	0.00218	2.29994
b	c	0.00001	339.48489	0.38835	1.42062	0.61164	1.42411
b	c	0.00008	339.48248	0.26468	1.52322	0.73524	1.50541
a	a	0.99999	277.18790	0.00001	277.19371	0.00000	277.19433

Table 2.2: Comparison of Prediction CI for Classifier Predictions

Average Prediction CI

Our initial approach was to average the prediction CIs for each class prediction in a test instance. We then averaged that value across all instances, yielding the *average instance CI*. This value was then compared to two other aggregates, the average of the hit prediction CIs (*i.e.* the prediction CI for each instance where the prediction matched the true class) and the average of the prediction CIs for misses. The result was 3 scalars per dataset which can be compared directly. Our hypothesis was that we would observe a higher missed instance CI than average CI, indicating that when conviction is weak (prediction CI is high), it is more likely that the NB classifier would make a classification error. Recall that for binary datasets the prediction CI is the same for both class outcomes (refer to Equation (2.3)). Therefore, we only conduct this test on the multi-class datasets and present the results in Table 2.3.

The 14 of 20 (70%) of datasets marked with a • are those for which the average missed CIs exceeds the average CI. We interpret these results as a positive indication that there may be a correlation between missed predictions and weak conviction (high prediction CI). Otherwise, we would hope to observe no connection whatsoever. Nonetheless, we had some concerns about the use of averages across all instances of a datasets. For one, once the model has been constructed each prediction is episodic, and therefore unrelated to the prediction of other (distinct) instances. The use of an average lumps the instances

Dataset	Classes	Accuracy	Avg CI	Hit CI	Miss CI	
anneal	6	0.967	101.129	99.398	151.207	•
audiology	24	0.735	112.166	71.316	225.185	•
autos	7	0.717	101.231	78.858	157.934	•
balance-scale	3	0.707	0.409	0.406	0.416	•
contact-lenses	3	0.708	99.392	139.188	2.745	
glass	7	0.743	9.434	7.401	15.312	•
heart-c	5	0.845	4.746	2.652	16.149	•
heart-h	5	0.840	16.355	16.886	13.563	
hypothyroid	4	0.986	45.954	44.756	131.624	•
iris	3	0.940	108.081	112.126	44.719	
letter	26	0.740	7.973	6.458	12.293	•
lymphography	4	0.838	34.513	31.857	48.232	•
primary-tumor	22	0.502	14.528	11.748	17.324	•
segment	7	0.915	96.545	97.606	85.109	
soybean	19	0.930	52.164	50.258	77.378	•
splice	3	0.953	196.711	196.645	198.043	•
vehicle	4	0.627	19.014	22.777	12.702	
vowel	11	0.671	14.797	16.097	12.150	
waveform	3	0.807	0.662	0.621	0.835	•
zoo	7	0.931	259.618	255.963	308.700	•

Table 2.3: Average vs. Missed Prediction CI for Multi-class Datasets

of all classes into a single metric. (However, we avoid a per-class analysis in the interest of not needlessly widening our search space.) Furthermore, if there is in fact a strong correlation, why wouldn't the miss CI be consistently higher than the average CI or the hit CI?

Over/Under Ratios

Starting from the same source data like that in Table 2.2, we attempt to locate a correlation between NB prediction misses and prediction CIs through another approach. In the place of averages we opt for a more instance-centric view of CI. We modify our analysis scripts to count the following two occurrences. When the NB classifier misses the predic-

tion, we note the number of instances where the prediction CI for the actual class is over (greater than) the prediction CI for the predicted class. This is interpreted as the pool of misses that could potentially be improved by the taking prediction CI into account. When the NB classifier correctly predicts the actual class for an instance, we are faced with small dilemma because the prediction CI for the predicted class matches the *pci* of the actual class. In this case, we compare the prediction CI of the actual class with the other prediction CI values to determine whether it is the prediction of weakest conviction (highest CI) in the set. If it is the CI with weakest conviction, then the prediction CI might have been used to dissuade the NB classifier from choosing the correct class. For hits we count the positive cases, meaning those where the actual class prediction CI was not the weakest conviction prediction CI.

Table 2.4 depicts the results from the analysis. For the misses, we observe that the prediction CI for the actual class is of stronger conviction (lower CI) than the predicted class in 20% of classifier misses for the *anneal* dataset yet over in 83% for the *vehicle* and *vowel* datasets. These data imply that there is a correlation between the classifier missing a prediction and the prediction CI for the actual class having a CI implying stronger conviction. For 13 of the 20 datasets, the majority of misses exhibit this property. We are concerned with a majority because we would like to develop an algorithm that does not require a (potentially expensive) heuristic to assess whether to utilize the prediction CI in the prediction. That is to say, we would like to believe *a priori* that applying the prediction CI will augment the actual class prediction more often than it detracts from it. Also bear in mind that the pools of hits and misses are distinct, apportioned as per the classifier accuracy. In the case of a dataset like *anneal* the 20% ratio applies to the 3.3% of instances classified incorrectly. The hits column shows that the *pci* for the predicted class is only very rarely the weakest conviction prediction CI in the set. These numbers would appear overwhelmingly positive, but the statement being made about the actual

Dataset	Classes	Accuracy	Miss Over	Hit Under
anneal	6	0.967	0.200	1.000
audiology	24	0.735	0.817	1.000
autos	7	0.717	0.431	1.000
balance-scale	3	0.707	0.885	1.000
contact-lenses	3	0.708	0.714	1.000
glass	7	0.743	0.873	1.000
heart-c	5	0.845	0.404	1.000
heart-h	5	0.840	0.404	1.000
hypothyroid	4	0.986	0.365	1.000
iris	3	0.940	0.667	0.986
letter	26	0.740	0.715	1.000
lymphography	4	0.838	0.667	1.000
primary-tumor	22	0.502	0.710	1.000
segment	7	0.915	0.495	1.000
soybean	19	0.930	0.583	1.000
splice	3	0.953	0.520	0.995
vehicle	4	0.627	0.839	0.998
vowel	11	0.671	0.831	1.000
waveform	3	0.807	0.604	0.997
zoo	7	0.931	0.429	1.000

Table 2.4: Over/Under Ratios of Prediction CI Ratios w.r.t. Misses and Hits

class prediction CI not being that of least conviction in the set is considerably weaker than saying it is the strongest conviction CI of the set. The percentages where the hit prediction CI is in fact the best in the set are found in Table 2.5, where we observe that, as expected, the prediction CI alone is not a strong indicator of the actual class. In the next section we present our findings as we set out to attempt to use this correlation to improve NB classifier performance.

2.4 Experiment II: Classifiers and CI

Armed with a notion of prediction CI's relation to NB class predictions from Section 2.3.4, we attempt to develop new classifiers based on NB that take into account the confidence

Dataset	Classes	Accuracy	Hit Under (Strict)
anneal	6	0.967	0.158
audiology	24	0.736	0.115
autos	7	0.717	0.191
balance-scale	3	0.707	0.000
contact-lenses	3	0.708	0.294
glass	7	0.743	0.120
heart-c	5	0.845	0.406
heart-h	5	0.840	0.360
hypothyroid	4	0.986	0.020
iris	3	0.940	0.255
letter	26	0.740	0.018
lymph	4	0.838	0.161
primary-tumor	22	0.501	0.418
segment	7	0.915	0.030
soybean	19	0.930	0.038
splice	3	0.953	0.475
vehicle	4	0.626	0.036
vowel	11	0.671	0.241
waveform	3	0.807	0.007
zoo	7	0.931	0.021

Table 2.5: Prediction CI Ratios where Hit CI is of Strongest Conviction

intervals calculated during training. These classifiers are described in Section 2.4.1 and their evaluation in Section 2.4.2. Once again appealing to the intuitive sense of confidence, we assume we should be able to temper NB predictions either by examining attribute CI or the overall prediction CI. Attribute CI might be useful in selecting which attributes to use during prediction, and as such act as a type of dynamic (in the sense of context-aware) attribute selection algorithm; attributes with weak conviction should not be allowed to influence a prediction causing it to miss. Similarly, the overall prediction CI could be useful for discriminating between class predictions for multi-class datasets; for binary datasets the prediction CI varies only with instance, not with class.

2.4.1 Classifiers Utilizing Confidence Intervals

The `NBCi` classifier developed for Section 2.3.1 implements a naïve Bayesian classifier that calculates and stores attribute and prediction CIs in the model built during training. This class is thus easily extended to experiment with alternative prediction algorithms. This section describes attempts during the course of our research to incorporate CI into NB prediction. These attempts admittedly depict a random walk through ideas informed by other research in conjunction with our intuitive sense of how conviction might be used. They are based in part upon earlier experiments we conducted with LOR-based NB classifiers and CI which failed to yield a positive outcome. We do not include the results of those experiments in the interest of saliency and because they are documented in our poster submission to the FLAIRS conference [36].

We now describe the classifiers which we implemented in Weka and evaluated using the Experimenter module. CI values may range from very close to zero (very strong conviction) to 99.99 for attributes (defined by convention to be the weakest possible conviction) and are normalized for use within our classifiers. When a term x has been normalized, it appears in the equations below as $n(x)$.

- `NBCiAttr`—This classifier multiplies the predicted class probability by the normalized prediction CI. The intended effect is to diminish class predictions for which the prediction conviction is weak (prediction CI is high). The additional pci_X term has no influence on binary predictions, because $pci_X(c) = pci_X(\bar{c})$, and thus $n(pci_X(c)) = n(pci_X(\bar{c})) = 0.5$. This classifier is noteworthy because it the subject of the remainder of our thesis.

$$pred_X(c) = \underset{c}{argmax}\ P(c)\ n(\frac{1}{pci_X(c)})\ \prod_i P(a_i|c) \tag{2.4}$$

- `NBCiAugmentHigh`—This classifier augments the attribute term with the highest associated confidence by skipping over it, which is equivalent to changing the conditional attribute probability to 1. Therefore its prediction equation is the same as (1.3) with an added condition to eliminate the $P(a_i|c)$ term with the strongest conviction (smallest attribute CI).

- `NBCiDropLow`—This classifier eliminates the contribution of the weakest conviction (greatest attribute CI) by not including it in the product of attribute probabilities. It can be viewed as a complement of `NBCiAugmentHigh`.

- `NBCiExp`—Individual attribute confidence intervals are normalized and then used as exponents for their corresponding attribute probabilities. The intent is to adjust the influence of each term in the classification probability according to the confidence interval of the attribute. If the CI array cannot be normalized, all exponents are given equal weights. The idea for weighting attributes by exponentiation is taken from Zhang and Sheng's work on weighted naïve Bayes [25].

$$pred_X(c) = \underset{c}{argmax}\ P(c)\ \prod_i P(a_i|c)^{n(aci_x(c))} \tag{2.5}$$

- `NBCiExpInv`—Identical to `NBCiExp`, except that the normalized attribute CIs are inverted, this classifier better captures the intent of increasing the influence of high confidence attributes and diminishing the influence of lower confidence attributes. The reader will note that `NBCiExp` does the opposite, effectively increasing the impact of low confidence attributes.

$$pred_X(c) = \underset{c}{argmax}\ P(c)\ \prod_i P(a_i|c)^{n(aci_x(c))^{-1}} \tag{2.6}$$

2.4.2 Classifier Evaluation

We evaluate these classifiers against the datasets in the UCI corpus. Datasets marked with the β superscript are binary, all others are multi-class. Weka's `NaiveBayes` classifier serves as the baseline, and the full results using accuracy as the basis of comparison are given in Table 2.6. The results depict classifier accuracy after 10-times 10-fold cross validation. The (x/y/z) notation is one we will use henceforth to indicate the number of (wins/ties/losses) relative to the baseline; the $\circ$ denotes a win while the $\bullet$ a loss. In all results for which wins and losses are tallied, the victory must be statistically significant in terms of the paired Student's t-test with a twin-tailed confidence of 0.05 (95%).

Running the experiment all six classifiers across the 37 datasets takes about 6.5 minutes on reasonably current workstation-class PC. Weka's Experimenter interface provides visual feedback during the experiment, and the results include average classifier time for model generation and testing. Although we will take up the topic of run-time performance in some detail in Section 4.1, it is anecdotally mentioned here as we will not return to the exponentiation-based classifiers. Model generation is equivalent across all classifiers due to the shared model, but significant differences in prediction time were noted for classifiers employing exponentiation. `NBCiExp` and `NBCiExpInv` required from 3 to 6 times longer to perform testing than the other classifiers due to the exponentiation operations, raising floating point numbers to fractional powers. This is an area for possible performance improvement by moving the computation to occur via log-based arithmetic.

2.4.3 CI Classifier Evaluation Discussion

As is evident in Table 2.6, all of our experimental classifiers save but one are, generally-speaking, less accurate than stock NB. `NBCiDropLow` and `NBCiExpInv` are both appallingly poor performers, offering no advantages whatsoever while for many datasets decreasing accuracy (often significantly). We interpret this to mean that attributes with low sup-

Dataset	NB	NBCiAttr	NBCiAH	NBCiDL	NBCiExp	NBCiExpInv
anneal	96.13± 2.16	96.11± 2.15	96.05± 2.16	90.79± 3.02 •	87.59± 1.74 •	76.17± 0.55 •
audiology	72.64± 6.10	72.15± 6.49	72.73± 6.14	66.75± 6.83 •	61.14± 7.13 •	25.21± 1.85 •
autos	70.59±10.20	75.22± 9.81 ○	70.59±10.20	68.87±10.08	71.65± 8.96	34.58± 4.67 •
balance-scale	71.08± 4.29	71.10± 4.29	71.11± 4.40	72.29± 3.96	71.35± 4.21	71.52± 4.16
breast-cancer$^{\beta}$	72.70± 7.74	72.70± 7.74	73.08± 7.63	73.93± 7.09	69.91± 1.68	70.30± 1.37
breast-w$^{\beta}$	97.18± 1.79	97.18± 1.79	97.43± 1.64	97.27± 1.81	97.11± 1.82	92.16± 3.06 •
colic$^{\beta}$	79.68± 5.72	79.68± 5.72	79.73± 5.71	75.85± 6.79 •	75.82± 5.86	63.05± 1.13 •
contact-lenses	76.17±25.54	72.83±27.69	72.50±27.26	64.33±32.40	64.33±23.69	64.33±23.69
credit-a$^{\beta}$	86.39± 3.98	86.39± 3.98	86.23± 3.94	84.91± 3.70	83.07± 3.97 •	70.48± 3.75 •
credit-g$^{\beta}$	75.43± 3.84	75.43± 3.84	75.43± 3.84	74.23± 3.75	70.00± 0.00 •	70.00± 0.00 •
diabetes$^{\beta}$	77.85± 4.67	77.85± 4.67	77.85± 4.67	71.52± 4.15 •	73.39± 3.24 •	65.11± 0.34 •
glass	74.39± 7.95	74.15± 7.87	74.21± 8.04	66.71± 8.84 •	68.79± 7.53 •	46.47± 8.46 •
heart-c	83.97± 6.37	83.97± 6.37	83.97± 6.37	81.96± 8.31	81.81± 6.21	67.05± 5.38 •
heart-h	84.24± 6.31	84.24± 6.31	84.24± 6.31	83.51± 6.38	80.29± 5.70	63.95± 1.36 •
heart-statlog$^{\beta}$	83.74± 6.25	83.74± 6.25	83.74± 6.25	82.37± 7.05	82.00± 6.72	55.56± 0.00 •
hepatitis$^{\beta}$	85.12± 9.73	85.12± 9.73	85.12± 9.73	86.21± 9.54	83.89± 5.19	79.38± 2.26
hypothyroid	98.62± 0.56	99.20± 0.47 ○	98.62± 0.56	93.78± 1.26 •	93.21± 0.42 •	92.29± 0.09 •
ionosphere$^{\beta}$	90.77± 4.76	90.77± 4.76	90.77± 4.76	90.51± 4.55	94.30± 3.34 ○	64.50± 1.73 •
iris	94.47± 5.61	93.93± 5.61	94.67± 5.69	96.53± 4.07	93.67± 5.56	88.80± 9.13 •
kr-vs-kp$^{\beta}$	87.79± 1.91	87.79± 1.91	87.78± 1.91	77.62± 2.16 •	81.73± 2.13 •	52.22± 0.10 •
labor$^{\beta}$	92.53±11.68	92.53±11.68	92.53±11.68	93.30±10.27	90.53±12.50	64.67± 3.07 •
letter	74.00± 0.88	74.91± 0.83 ○	74.00± 0.88	67.84± 1.00 •	74.17± 0.93	2.00± 0.37 •
lymphography	84.97± 8.30	83.66± 7.84	84.97± 8.19	81.94± 9.29	75.78± 8.97 •	54.76± 2.32 •
mushroom$^{\beta}$	95.76± 0.73	95.76± 0.73	95.76± 0.73	90.28± 0.95 •	99.77± 0.18 ○	58.23± 0.77 •
primary-tumor	49.71± 6.46	49.12± 5.60	49.82± 6.41	43.81± 6.81 •	24.87± 1.56 •	24.78± 1.47 •
schizo$^{\beta}$	59.12± 6.89	59.12± 6.89	58.15± 6.93	54.79± 7.18 •	59.47± 7.45	57.97± 6.33
segment	91.71± 1.68	93.22± 1.50 ○	91.71± 1.68	88.81± 1.73 •	94.30± 1.42 ○	7.94± 1.47 •
sick$^{\beta}$	97.22± 0.80	97.22± 0.80	97.22± 0.80	93.63± 1.39 •	93.88± 0.08 •	93.88± 0.08 •
sonar$^{\beta}$	85.16± 7.52	85.16± 7.52	85.16± 7.52	82.66± 8.01	81.42± 8.78	53.38± 1.63 •
soybean	92.94± 2.92	94.41± 2.29 ○	92.97± 2.90	90.16± 3.24 •	89.87± 3.43 •	17.00± 2.72 •
splice	95.41± 1.18	95.40± 1.18	95.07± 1.27	95.42± 1.14	16.79± 4.59 •	51.88± 0.16 •
titanic$^{\beta}$	77.85± 2.40	77.85± 2.40	78.20± 2.52	77.42± 1.93	77.10± 2.76	68.33± 0.53 •
vehicle	62.52± 3.81	63.10± 3.82	62.29± 3.73	61.84± 4.06	64.49± 3.89	54.03± 4.51 •
vote$^{\beta}$	90.02± 3.91	90.02± 3.91	89.75± 4.34	88.02± 4.73 •	90.44± 4.50	88.21± 4.63
vowel	65.23± 4.53	66.23± 4.57	65.28± 4.42	60.61± 4.38 •	55.09± 4.63 •	11.99± 2.11 •
waveform	80.72± 1.50	80.89± 1.45	80.72± 1.50	78.45± 1.53 •	82.33± 1.40 ○	33.84± 0.08 •
zoo	93.21± 7.35	94.56± 6.81	93.21± 7.35	93.98± 7.14	60.43± 3.06 •	40.61± 2.92 •
	(0/37/0)	(5/32/0)	(0/37/0)	(0/20/17)	(6/15/16)	(0/8/29)

○, • statistically significant improvement or degradation

Table 2.6: Accuracy of CI-augmented Classifier Performance Relative to Naïve Bayes.

port are significant components of the naïve Bayes proposition—they very likely serve to diminish the probability calculation for the incorrect class sufficiently so as to exclude it from consideration, an important function. `NBCiAugmentHigh` neither suffers losses nor garners wins and in many cases has identical accuracy to NB. We explicate this result by observing that the classifier merely increases the probability for the attribute value with the greatest amount of support in the dataset, which in almost all cases is the attribute value with the highest conditional probability (by virtue of the fact that both are based

on counts). Therefore, the slight differences from unadulterated NB would be due to predictions where the class priors are similar and the augmented attribute probability is sufficient to modify the prediction outcome.

In general the mixed results are not particularly surprising given that their use of CI is based on intuition and not founded in statistical theory. Nonetheless, it is encouraging that one classifier is able to outperform NB on some datasets without sacrificing accuracy on others. We posit that one component of the success of `NBCiAttr` is that it does not modify individual terms in the Bayesian product. It thereby avoids disrupting predictions which hinge on specific attributes and instead serves to nudge the classifier towards the correct prediction, lending support support to the hypothesis that weak conviction (high prediction CI) is a contributor to missed predictions. The degree of success for the multi-class datasets, 25% of which showed improvement and 75% of which were statistically equivalent to NB, warrants additional investigation. It could be that classes for which there is very little support (*i.e.* low confidence) are effectively removed from consideration by argmax. Or the opposite could be the case, wherein several competing classes have similar probabilities until the confidence-based term is factored in, which effectively boosts the correct prediction to obtain a plurality. Although their performance is generally dismal, we believe the other classifiers need more research before their approaches can be entirely discounted. We maintain the notion that it could well be a single attribute that causes a missed prediction. However we do not dwell on them further in our research, focusing instead on the potentially promising results of `NBCiAttr` by vetting the algorithm more thoroughly.

Before moving on we motivate the use of prediction CI with one additional basis of comparison. Figure 2.3 combines the results of the average versus missed CI investigation in Section 2.3.4 (Table 2.3) and the results of the `NBCiAttr` classifier for multi-class class datasets. Points to the right of 1.0 are desired, and points in the upper right

quadrant (noting that the y-axis is log-scale) best support the hypothesis that high ratios of missed instance CI to average instance CI are indicative of the usefulness of CI during classification.

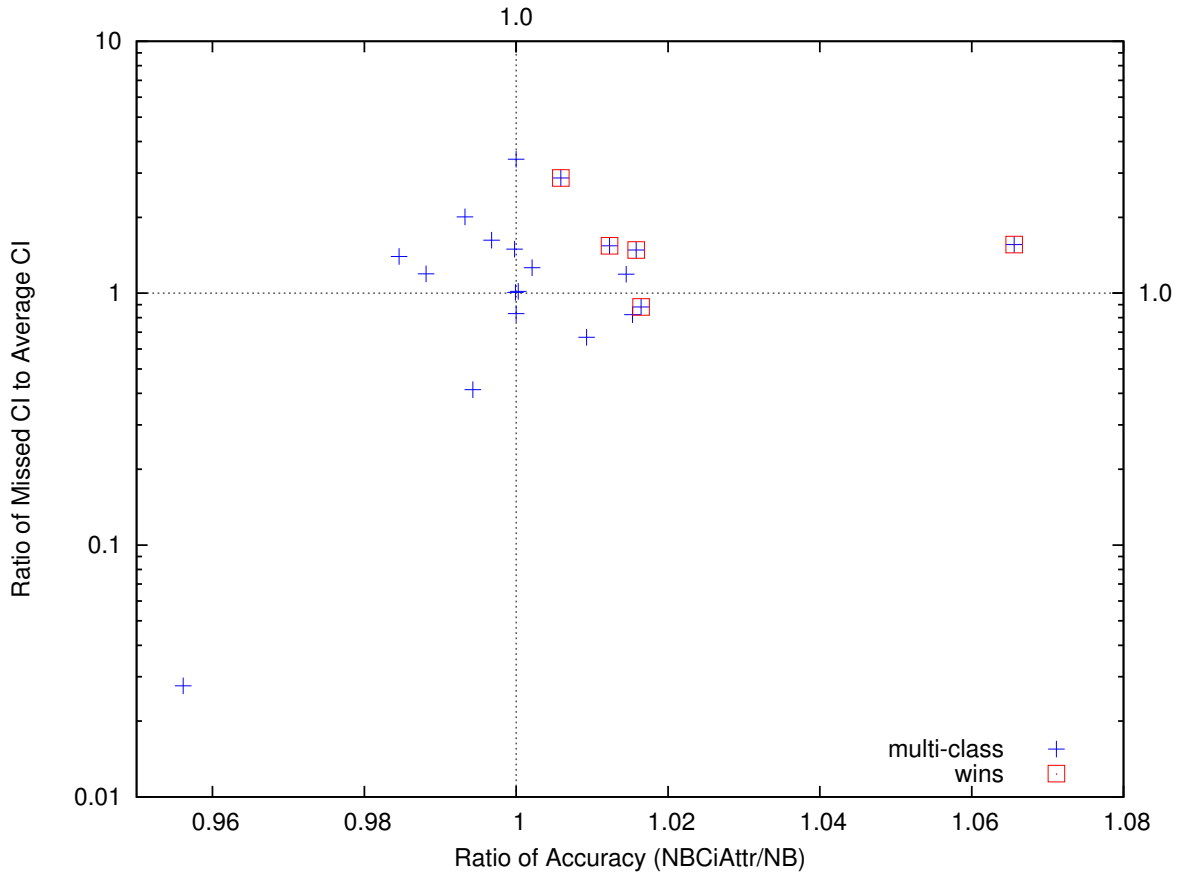


Figure 2.3: CI Miss to Average Prediction Ratio vs. Accuracy Ratio

We close this chapter with a demonstration of the `NBCiAttr` classifier in action. We take the example problem given in Section 1.2.4 and extend it such that NB fails to correctly classify an instance in the test set. We then show the use of prediction CI to "correct" the instance where naïve Bayes fails to predict the correct class.

2.5 An Example of CI-Augmented NB in Action

The intuitive conceptualization of why prediction CI (2.3) aids in NB prediction is that the value indicates how much support exists for each class hypothesis about an instance. The larger the CI for a given hypothesis, the weaker our conviction in its support, and the more cautious we should be of it. When we take the prediction confidence intervals for each class and invert them, we have a ranking of 1–k for which prediction we believe to be the strongest in terms of CI. The fact that the value is normalized is immaterial, as this multiplies all of the inverted prediction CIs by the same factor. It merely serves to scale our CI attribute into a familiar range (0–1).

To motivate this notion of support, we provide an example for which NB fails to correctly learn the example pattern given back in Section 1.2.4, in this case due to noise in the dataset. The training dataset is shown in Table 2.7; it is the same dataset as used in the original example replicated four times and with four additional training instances. The first three of these are valid instances of XOR, while the fourth is an example of class noise. Table 2.8 contains the results of the model after training, including the class and attribute confidence intervals. Note that we employ the same Laplace estimators as in the previous example, hence the denominator 23 in the class priors.

x	**y**	**Class**	**Comment**
0	1	xor	repeated 4x
1	0	xor	
1	1	and	
0	0	nor	
1	0	xor	
1	0	xor	
1	0	xor	
0	0	xor	(noise!)

Table 2.7: NB and NBCiAttr Example Training Dataset

			Conditional probabilities and *aci*							
	Prior	*cci*	x=0		x=1		y=0		y=1	
xor	13/23	0.4206	6/14	0.5226	8/14	0.4647	9/14	0.4451	5/14	0.5684
and	5/23	0.5055	1/6	99.9999	5/6	0.3705	1/6	99.9999	5/6	0.4944
nor	5/23	0.5055	5/6	0.4410	1/6	99.9999	5/6	0.3456	1/6	99.9999

Table 2.8: NB and NBCiAttr Example Model After Training

We calculate predictions for the test instance $x = 0, y = 0$ in Table 2.9 using both NB and our CI-augmented classifier that incorporates *pci* as a prediction term (`NBCiAttr` Equation (2.4)). In this example naïve Bayes fails to calculate the true class NOR, instead predicting XOR, and will unfortunately do so forevermore unless the model is updated with additional training instances to improve the likelihood of NOR for instances with these attribute values. `NBCiAttr` is able classify the test instance correctly by slightly deemphasizing class XOR in favor of class NOR.

Prediction	$p(c) \times p_{x=0}(c) \times p_{y=0}(c)$			$1/pci$	**Result**	**Normalized**
$p(xor\|x=0,y=0)$	13/23 ×	6/14 ×	9/14		0.15572	0.49795
$p(and\|x=0,y=0)$	5/23 ×	1/6 ×	1/6		0.06038	0.01931
$p(nor\|x=0,y=0)$	5/23 ×	5/6 ×	5/6		0.15097	0.48274
$p(xor\|x=0,y=0)$	13/23 ×	6/14 ×	9/14	1.24218	0.19343	0.49154
$p(and\|x=0,y=0)$	5/23 ×	1/6 ×	1/6	0.00707	0.00004	0.00010
$p(nor\|x=0,y=0)$	5/23 ×	5/6 ×	5/6	1.32517	0.20005	0.50836

Table 2.9: NB and NBCiAttr Example Prediction

Chapter 3

NBCiAttr Evaluation

3.1 Evaluation of NBCiAttr on UCI Datasets

Our emphasis from this point forward is on validating the performance of the `NBCiAttr` classifier. This means that we will no longer be concerned with binary datasets, and accuracy comparisons will be against naïve Bayes alone; the metrics considered now include root mean squared error (RMSE) and the area under the ROC curve (AUC). Next we investigate the performance of our classifier in the presence of noise, which in this context refers to random permutations of the class from its true value. From there we expand our domain beyond the UCI datasets and exercise `NBCiAttr` on real-world document classification datasets. This in turn leads to a discussion regarding the problem of underflow, which NB and derivatives are susceptible to when finding patterns in datasets with large numbers of attributes.

3.1.1 Accuracy

The results in Table 3.1 for `NBCiAttr` are the same as those found in Table 2.6; the binary datasets have been removed, the precision has been increased, and the standard deviation

of the accuracy across the 10 iterations of 10-fold cross validation is included. Of the 20 datasets total, there are 5 for which `NBCiAttr` improves accuracy and 15 for which the results are statistically equivalent. Importantly, there are no datasets for which there are statistically significant decreases in accuracy—for `NBCiAttr` to be a viable alternative to NB without predicating analysis as to its applicability beforehand (which raises the self-referential issue of not knowing when a tool is applicable until after it has been applied to the problem), it must above all do no harm.

We also add secondary indicators: σ to draw attention to datasets where the standard deviation is improved (lower) for `NBCiAttr` than it is for NB; and ϵ to indicate a higher standard deviation. Unlike for accuracy, the Student's t-test is not applied to test for statistical significance as standard deviation is already a statistical entity. The results for standard deviation are varied, with 13 wins, 2 ties, and 5 losses. The ties are for the *heart-c* and *heart-h* for which there is also no change whatsoever in accuracy, indicative of a case where the prediction CI attribute has no effect on the prediction. Four of the five standard deviation losses occur with datasets for which the missed prediction CI is less than the average prediction CI (see Section 2.3.4, Table 2.3), and so based on our supposition that CI would not play a factor in improving prediction (and in fact it did not improve accuracy for those datasets 5 datasets) we accept these results as expected. We also bring to the reader's attention the fact that there are no datasets for which the standard deviation is worse while accuracy is improved.

We have not encountered other research that compares standard deviation directly, but believe it is worthy of investigation because it signifies lower variance in classifier accuracy across the 100 runs. We claim that decreased standard deviation is a positive trait indicative of a dampening or tightening of the variability of accuracy, which we interpret as decreased sensitivity to fluctuations in the training dataset—recall that we are exercising different folds for each iteration—and potentially a sign of a classifier that is better

generalized. Or in Dietterich's parlance in [24], the classifier is narrowing the hypothesis space to find a better approximation of the true function f. Although the differences in accuracy and standard deviation are small, we assert that the novelty of employing CI is that the time complexity of NB is maintained and therefore the incremental improvement is "free." This is in distinct contrast to many other approaches to modifying naïve Bayes.

Dataset	**NB**	**NBCiAttr**	
anneal	96.1350 ± 2.1573	96.1126 ± 2.1463	σ
audiology	72.6383 ± 6.1010	72.1482 ± 6.4929	ϵ
autos	70.5881 ± 10.2016	75.2238 ± 9.8054	$\circ$ σ
balance-scale	71.0817 ± 4.2925	71.0978 ± 4.2891	σ
contact-lenses	76.1667 ± 25.5418	72.8333 ± 27.6923	ϵ
glass	74.3939 ± 7.9461	74.1515 ± 7.8746	σ
heart-c	83.9667 ± 6.3678	83.9667 ± 6.3678	
heart-h	84.2356 ± 6.3085	84.2356 ± 6.3085	
hypothyroid	98.6188 ± 0.5580	99.1995 ± 0.4734	$\circ$ σ
iris	94.4667 ± 5.6102	93.9333 ± 5.6134	ϵ
letter	73.9960 ± 0.8763	74.9080 ± 0.8305	$\circ$ σ
lymphography	84.9667 ± 8.2978	83.6571 ± 7.8369	σ
primary-tumor	49.7068 ± 6.4618	49.1159 ± 5.5955	σ
segment	91.7100 ± 1.6787	93.2208 ± 1.5049	$\circ$ σ
soybean	92.9422 ± 2.9175	94.4077 ± 2.2913	$\circ$ σ
splice	95.4075 ± 1.1842	95.4013 ± 1.1839	σ
vehicle	62.5169 ± 3.8093	63.0965 ± 3.8248	ϵ
vowel	65.2323 ± 4.5291	66.2323 ± 4.5746	ϵ
waveform	80.7180 ± 1.4980	80.8920 ± 1.4465	σ
zoo	93.2091 ± 7.3452	94.5636 ± 6.8079	σ

$\circ$, $\bullet$ statistically significant improvement or degradation
σ, ϵ decreased or increased standard deviation

Table 3.1: Accuracy - UCI Datasets

3.1.2 RMSE

We now turn to root mean squared error, shown in Table 3.2. Note that when evaluating RMSE, lower numbers are better as they indicate less error in terms of the how close the classifier's predicted probabilities were to the actual class value (arbitrarily assigned a value of 1) or the non-class values (arbitrarily assigned to be 0) for nominal datasets. In the multi-class context, RMSE cannot be meaningfully compared between datasets of different class cardinalities. For RMSE across the UCI datasets, our first look at the probabilistic understanding of error, `NBCiAttr` offers 9 improvements, 10 ties, and 1 loss compared to NB. The comparison of standard deviation of the RMSE values for both classifiers is depicted alongside RMSE wins. Although there are only 2 wins, with 9 ties and 9 losses, the absolute differences for the losses are small, and so do not indicate large variations in RMSE amongst different folds compared to NB.

3.1.3 AUC

For our last analysis using the stock UCI datasets, we compare the area under the receiver operating characteristic curve (AUC) in order to gain some insight into the ranking performance of our classifier compared to naïve Bayes; these results are found in Table 3.3. Recall from Section 1.3.4 that AUC is essentially a race to collect true positives more quickly than false positives. Of the 20 datasets total, there are 5 for which NB exhibits an AUC of 1.0 which `NBCiAttr` retains, and so there are 15 datasets for which AUC can be improved. `NBCiAttr` accomplishes this for 4 of those 15 with no statistically significant decreases. Again we compare standard deviation in AUC and observe mixed results of 8 wins, 8 ties, and 4 losses. We note that averaging of multiple AUC is a complex topic with several possible interpretations, even in the case of binary datasets, and is beyond the scope of our evaluation. The interested reader can find more information on this topic in [9].

Dataset	NB	NBCiAttr		
anneal	0.095 ± 0.025	0.090 ± 0.025	∘	
audiology	0.135 ± 0.014	0.138 ± 0.016		ϵ
autos	0.255 ± 0.049	0.241 ± 0.053	∘	ϵ
balance-scale	0.326 ± 0.013	0.324 ± 0.014	∘	ϵ
contact-lenses	0.296 ± 0.107	0.353 ± 0.167		ϵ
glass	0.230 ± 0.034	0.237 ± 0.037		ϵ
heart-c	0.221 ± 0.047	0.221 ± 0.047		
heart-h	0.216 ± 0.045	0.216 ± 0.045		
hypothyroid	0.074 ± 0.011	0.062 ± 0.011	∘	
iris	0.127 ± 0.098	0.136 ± 0.095		σ
letter	0.119 ± 0.002	0.118 ± 0.002	∘	
lymphography	0.232 ± 0.068	0.247 ± 0.057		σ
primary-tumor	0.175 ± 0.008	0.181 ± 0.009	•	ϵ
segment	0.141 ± 0.014	0.129 ± 0.014	∘	
soybean	0.080 ± 0.016	0.071 ± 0.016	∘	
splice	0.151 ± 0.017	0.151 ± 0.017		
vehicle	0.380 ± 0.018	0.378 ± 0.019		ϵ
vowel	0.205 ± 0.009	0.202 ± 0.011	∘	ϵ
waveform	0.327 ± 0.011	0.326 ± 0.012	∘	ϵ
zoo	0.089 ± 0.052	0.084 ± 0.052		

∘, • statistically significant improvement or degradation
σ, ϵ decreased or increased standard deviation

Table 3.2: RMSE - UCI Datasets

3.2 Resistance to Class Noise

Li *et al.* [37] assert that class noise is indeed a problem worthy of scientific inquiry and develop an approach to detect and compensate for noise using a probabilistic Kernel Fisher method. Zhu and Wu [38] investigate both class noise and attribute noise and conclude that attribute noise is "usually less harmful" (than class noise) and can be addressed with noise correction, whereas class noise is best addressed by filtering noisy instances from the dataset. Both refer to earlier work in this area by Brodley and Friedl [39] that asserts that mislabeled classes occur due to subjectivity, data-entry error and incomplete data

Dataset	NB	NBCiAttr		
anneal	0.9879 ± 0.0377	0.9760 ± 0.0737		ϵ
audiology	0.9703 ± 0.0907	0.9606 ± 0.1239		ϵ
autos	1.0000 ± 0.0000	1.0000 ± 0.0000		
balance-scale	0.8924 ± 0.0347	0.8917 ± 0.0345		σ
contact-lenses	0.9750 ± 0.1486	0.9400 ± 0.2165		ϵ
glass	0.8973 ± 0.0626	0.8986 ± 0.0600		σ
heart-c	0.9124 ± 0.0529	0.9135 ± 0.0523		σ
heart-h	0.9210 ± 0.0481	0.9224 ± 0.0474		σ
hypothyroid	0.9966 ± 0.0022	0.9979 ± 0.0022	∘	
iris	1.0000 ± 0.0000	1.0000 ± 0.0000		
letter	0.9858 ± 0.0049	0.9896 ± 0.0035	∘	σ
lymphography	1.0000 ± 0.0000	1.0000 ± 0.0000		
primary-tumor	0.8970 ± 0.0600	0.8964 ± 0.0559		σ
segment	0.9990 ± 0.0012	0.9994 ± 0.0008	∘	σ
soybean	1.0000 ± 0.0000	1.0000 ± 0.0000		
splice	0.9949 ± 0.0029	0.9949 ± 0.0029		
vehicle	0.7744 ± 0.0449	0.7839 ± 0.0449	∘	
vowel	0.9919 ± 0.0081	0.9926 ± 0.0096		ϵ
waveform	0.9443 ± 0.0093	0.9443 ± 0.0092		σ
zoo	1.0000 ± 0.0000	1.0000 ± 0.0000		

∘, • statistically significant improvement or degradation
σ, ϵ decreased or increased standard deviation

Table 3.3: AUC - UCI Datasets

during creation of the training set, and go on to discuss algorithms to filter class noise. Their approach is to develop consensus filters (ensembles) based on outlier detection in regression analysis.

We recognize the validity and utility of these efforts, but would argue that such approaches are only reasonable if one can expect to know *a priori* that instances can contain noise and has a sufficient time and processing budget to employ noise/outlier detection algorithms. Given our focus on maintaining the time complexity of naïve Bayes, we evaluate the performance of `NBCiAttr` relative to NB in the presence of class noise. Because prediction CI, Equation (2.3), includes attribute confidence terms *relative to class*, we

submit that there is a basis upon which to believe that our CI-augmented algorithm can accommodate class outliers without a severe degradation in performance—*i.e.* that we may be able to improve classification performance amidst ambient noise.

3.2.1 Evaluation at Fixed Percentages

We begin by preparing 4 variants of the 20 multi-class UCI input datasets using Weka's `AddNoise` filter[1] to introduce 5, 10, 15, and 20% noise to each. We then perform 10 times 10-fold CV across datasets and compare the results, found in Tables 3.4a–3.4d, to plain NB. Recall from Table 3.1 that in the absence of noise, `NBCiAttr` delivered 5 wins across the 20 datasets. There continue to be some wins as noise increases. However, the data do not bear our hypothesis regarding improved performance in detecting outliers with respect to class. At 5% noise, the result is 6 wins without any losses, with the addition of two datasets, *anneal* and *waveform* to the win column, moving *hypothyroid* to the tie column. At 10% noise, 2 of those wins (*autos* and *waveform*) dissipate. For 15% noise we observe our first statistical loss with `NBCiAttr`, for the *audiology* dataset, and record only 3 statistical wins (*soybean* has dropped out of the wins column). And finally at 20% noise, the result is 4 wins and 16 ties, again back to *anneal*, *letter*, *segment*, and *soybean*, the same winners at the 10% noise level.

Tables 3.5a and 3.5b compare the RMSE and AUC metrics at the 5% noise level. In the interest of space we summarize the RMSE and AUC results at the various noise levels in Table 3.6. From this table we note that while classifier performance in terms of accuracy is maintained in the presence of increasing noise, RMSE and AUC degrade, and between those two, RMSE more quickly.

[1] `weka.filters.unsupervised.attribute.AddNoise`

Dataset	NB	NBCiAttr	
anneal-n05	90.19 ± 2.54	91.58 ± 2.33	○
audiology-n05	69.77 ± 6.80	68.88 ± 7.33	
autos-n05	68.04 ± 9.23	72.22 ± 8.93	○
balance-scale-n05	69.36 ± 3.98	69.41 ± 4.07	
contact-lenses-n05	74.17 ± 28.27	73.50 ± 27.43	
glass-n05	69.84 ± 7.69	69.18 ± 7.56	
heart-c-n05	79.48 ± 6.37	79.55 ± 6.39	
heart-h-n05	79.83 ± 5.56	80.21 ± 5.48	
hypothyroid-n05	93.15 ± 0.95	93.36 ± 0.96	
iris-n05	89.00 ± 7.95	88.40 ± 8.19	
letter-n05	69.63 ± 1.06	70.43 ± 1.06	○
lymphography-n05	79.44 ± 10.51	78.90 ± 11.00	
primary-tumor-n05	46.58 ± 5.72	45.13 ± 5.99	
segment-n05	85.76 ± 2.10	86.80 ± 2.09	○
soybean-n05	87.85 ± 3.47	89.13 ± 3.48	○
splice-n05	89.79 ± 1.54	89.79 ± 1.54	
vehicle-n05	60.37 ± 4.45	60.52 ± 4.51	
vowel-n05	60.05 ± 4.49	60.93 ± 4.25	
waveform-n05	76.90 ± 1.51	77.06 ± 1.53	○
zoo-n05	88.78 ± 9.35	89.49 ± 9.05	

○, • statistically significant improvement or degradation

(a) 5% Noise

Dataset	NB	NBCiAttr	
anneal-n10	82.63 ± 3.50	84.14 ± 3.41	○
audiology-n10	63.54 ± 7.62	63.72 ± 7.46	
autos-n10	67.10 ± 10.51	70.13 ± 9.88	
balance-scale-n10	67.81 ± 4.53	67.92 ± 4.59	
contact-lenses-n10	54.00 ± 30.90	55.33 ± 29.48	
glass-n10	67.75 ± 8.75	66.91 ± 8.95	
heart-c-n10	75.30 ± 6.72	75.07 ± 6.75	
heart-h-n10	75.84 ± 5.65	76.38 ± 5.09	
hypothyroid-n10	87.82 ± 1.20	88.03 ± 1.03	
iris-n10	87.00 ± 8.96	87.60 ± 8.53	
letter-n10	65.75 ± 1.04	66.34 ± 1.05	○
lymphography-n10	74.37 ± 9.03	73.84 ± 9.65	
primary-tumor-n10	42.12 ± 7.68	41.76 ± 6.65	
segment-n10	80.72 ± 2.52	81.29 ± 2.55	○
soybean-n10	82.66 ± 4.31	84.38 ± 4.10	○
splice-n10	84.15 ± 1.87	84.15 ± 1.87	
vehicle-n10	56.86 ± 4.94	56.72 ± 4.85	
vowel-n10	56.11 ± 4.94	56.47 ± 4.86	
waveform-n10	73.15 ± 1.46	73.25 ± 1.45	
zoo-n10	83.16 ± 9.66	85.15 ± 10.01	

○, • statistically significant improvement or degradation

(b) 10% Noise

Dataset	NB	NBCiAttr	
anneal-n15	78.56 ± 3.89	80.08 ± 3.60	○
audiology-n15	61.45 ± 7.70	58.54 ± 7.59	•
autos-n15	62.39 ± 9.14	64.91 ± 9.11	
balance-scale-n15	66.29 ± 4.14	66.48 ± 4.22	
contact-lenses-n15	55.67 ± 28.84	56.00 ± 27.88	
glass-n15	63.32 ± 10.30	64.30 ± 9.47	
heart-c-n15	70.80 ± 6.93	70.63 ± 7.07	
heart-h-n15	71.89 ± 6.13	72.16 ± 6.05	
hypothyroid-n15	83.15 ± 1.21	83.33 ± 1.17	
iris-n15	82.60 ± 10.33	82.60 ± 10.33	
letter-n15	61.79 ± 1.00	62.25 ± 1.00	○
lymphography-n15	71.16 ± 9.94	70.62 ± 9.62	
primary-tumor-n15	38.17 ± 6.71	37.02 ± 6.93	
segment-n15	75.80 ± 3.19	76.20 ± 3.13	○
soybean-n15	77.78 ± 4.40	78.53 ± 4.52	
splice-n15	78.77 ± 2.10	78.77 ± 2.10	
vehicle-n15	54.74 ± 4.14	54.63 ± 4.14	
vowel-n15	52.91 ± 4.68	53.29 ± 4.72	
waveform-n15	69.27 ± 2.06	69.33 ± 2.04	
zoo-n15	76.65 ± 10.41	77.84 ± 10.16	

○, • statistically significant improvement or degradation

(c) 15% Noise

Dataset	NB	NBCiAttr	
anneal-n20	73.31 ± 3.79	74.83 ± 3.57	○
audiology-n20	56.38 ± 6.60	54.43 ± 6.22	
autos-n20	57.66 ± 9.68	59.08 ± 9.68	
balance-scale-n20	63.27 ± 4.99	63.27 ± 4.98	
contact-lenses-n20	55.17 ± 28.69	55.17 ± 28.30	
glass-n20	56.30 ± 8.21	56.68 ± 8.49	
heart-c-n20	66.95 ± 6.31	66.71 ± 6.50	
heart-h-n20	67.57 ± 7.02	67.50 ± 6.89	
hypothyroid-n20	77.90 ± 1.31	78.23 ± 1.17	
iris-n20	78.07 ± 11.50	77.87 ± 11.37	
letter-n20	57.90 ± 0.93	58.37 ± 0.95	○
lymphography-n20	66.95 ± 12.37	67.42 ± 12.14	
primary-tumor-n20	34.84 ± 5.99	35.25 ± 6.01	
segment-n20	70.77 ± 2.87	71.26 ± 2.85	○
soybean-n20	72.39 ± 3.91	73.52 ± 3.79	○
splice-n20	73.55 ± 2.49	73.55 ± 2.49	
vehicle-n20	52.22 ± 4.71	52.27 ± 4.66	
vowel-n20	50.96 ± 4.53	50.95 ± 4.42	
waveform-n20	65.92 ± 1.87	65.99 ± 1.87	
zoo-n20	72.96 ± 12.44	72.76 ± 12.49	

○, • statistically significant improvement or degradation

(d) 20% Noise

Table 3.4: Accuracy - UCI Datasets with Variable Noise

3.2.2 Visualizing Accuracy as a Function of Class Noise

In an attempt to gain a better understanding of how classifier accuracy degrades as a function of noise, we plot the accuracy of NB and `NBCiAttr` as noise increases from 0–20%

Dataset	NB	NBCiAttr
anneal-n05	0.165 ± 0.021	0.156 ± 0.021 ○
audiology-n05	0.142 ± 0.015	0.147 ± 0.017 ●
autos-n05	0.266 ± 0.042	0.256 ± 0.042
balance-scale-n05	0.353 ± 0.016	0.351 ± 0.016 ○
contact-lenses-n05	0.341 ± 0.102	0.367 ± 0.164
glass-n05	0.251 ± 0.029	0.259 ± 0.033 ●
heart-c-n05	0.257 ± 0.037	0.259 ± 0.038
heart-h-n05	0.252 ± 0.034	0.251 ± 0.035
hypothyroid-n05	0.179 ± 0.011	0.177 ± 0.011 ○
iris-n05	0.225 ± 0.110	0.230 ± 0.112
letter-n05	0.130 ± 0.002	0.129 ± 0.002 ○
lymphography-n05	0.281 ± 0.074	0.290 ± 0.076
primary-tumor-n05	0.179 ± 0.007	0.185 ± 0.009 ●
segment-n05	0.189 ± 0.014	0.183 ± 0.014 ○
soybean-n05	0.107 ± 0.016	0.102 ± 0.017 ○
splice-n05	0.239 ± 0.019	0.239 ± 0.019 ○
vehicle-n05	0.394 ± 0.023	0.391 ± 0.023 ○
vowel-n05	0.221 ± 0.009	0.219 ± 0.009
waveform-n05	0.361 ± 0.012	0.360 ± 0.012 ○
zoo-n05	0.141 ± 0.070	0.139 ± 0.072

●, ○ statistically significant improvement or degradation

(a) RMSE

Dataset	NB	NBCiAttr
anneal-n05	0.711 ± 0.228	0.676 ± 0.241
audiology-n05	0.545 ± 0.108	0.582 ± 0.130
autos-n05	0.110 ± 0.061	0.130 ± 0.059
balance-scale-n05	0.854 ± 0.041	0.852 ± 0.041
contact-lenses-n05	0.930 ± 0.248	0.930 ± 0.248
glass-n05	0.884 ± 0.068	0.882 ± 0.067
heart-c-n05	0.870 ± 0.061	0.871 ± 0.060
heart-h-n05	0.885 ± 0.062	0.890 ± 0.063
hypothyroid-n05	0.801 ± 0.037	0.804 ± 0.037
iris-n05	0.940 ± 0.078	0.938 ± 0.081
letter-n05	0.957 ± 0.012	0.958 ± 0.013
lymphography-n05	0.663 ± 0.345	0.661 ± 0.350
primary-tumor-n05	0.882 ± 0.069	0.879 ± 0.072
segment-n05	0.971 ± 0.021	0.970 ± 0.022
soybean-n05	0.939 ± 0.112	0.953 ± 0.101
splice-n05	0.944 ± 0.016	0.944 ± 0.016
vehicle-n05	0.755 ± 0.059	0.763 ± 0.059 ○
vowel-n05	0.984 ± 0.014	0.987 ± 0.014
waveform-n05	0.908 ± 0.014	0.908 ± 0.014
zoo-n05	0.965 ± 0.078	0.966 ± 0.078

○, ● statistically significant improvement or degradation

(b) AUC

Table 3.5: RMSE and AUC - UCI Datasets 5% Noise

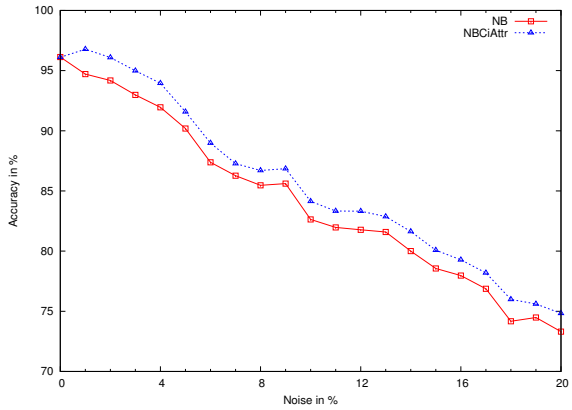
Noise %	Accuracy	RMSE	AUC
0	(5/15/0)	(9/10/1)	(4/16/0)
5	(6/14/0)	(9/8/3)	(1/19/0)
10	(4/16/0)	(7/11/2)	(1/19/0)
15	(3/16/1)	(4/11/5)	(1/19/0)
20	(4/16/0)	(2/10/8)	(0/20/0)

Table 3.6: Summary of NBCiAttr vs. NB with Class Noise (wins/ties/losses)

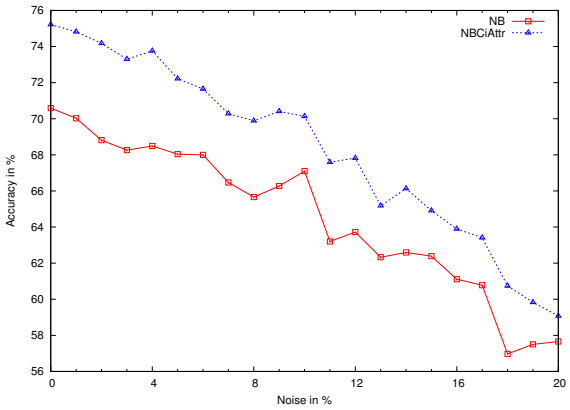
in 1 percentage point increments. This is done for several of the UCI datasets, 3 for which `NBCiAttr` outperforms NB (without noise) and 3 for which there is no statistical difference. The accuracy is found in Figures 3.1a–3.1f, and the associated standard deviations of accuracy in Figures 3.2a–3.2f.

Aside from *contact-lenses*, which is a very small dataset (24 instances), the remaining graphs indicate a fairly straight-forward correlation between the performance of NB and `NBCiAttr` as noise increases, both in terms of accuracy and the standard deviation of

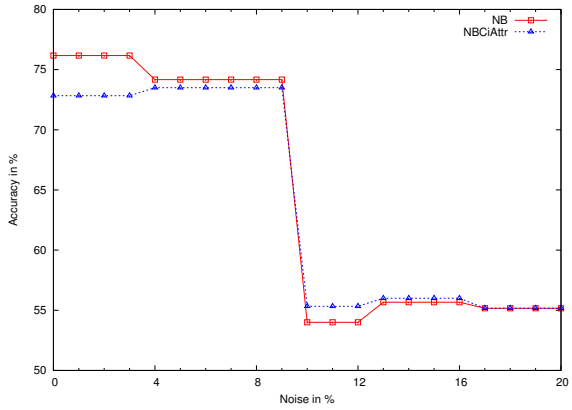
accuracy. We therefore conclude that `NBCiAttr` is not contributing to improved classifier performance in the presence of class noise. This could be due to the decrease in attribute conviction (increase in attribute confidence intervals) established during training in Equation (2.1), causing the overall effect of confidence to be muted.



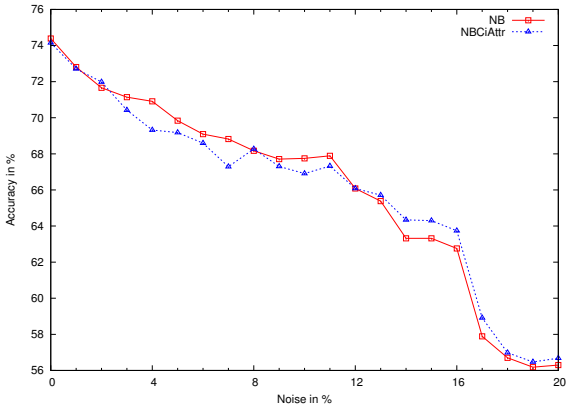
(a) Anneal



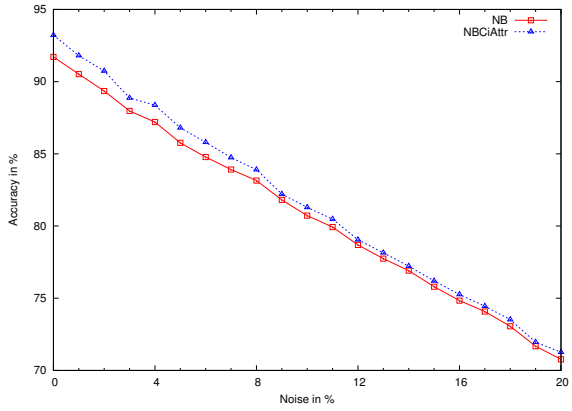
(b) Autos



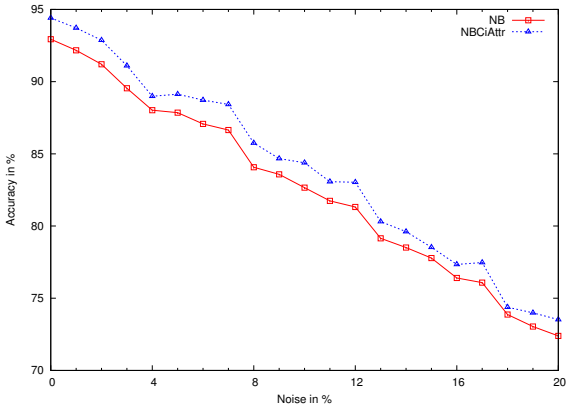
(c) Contact-lenses



(d) Glass

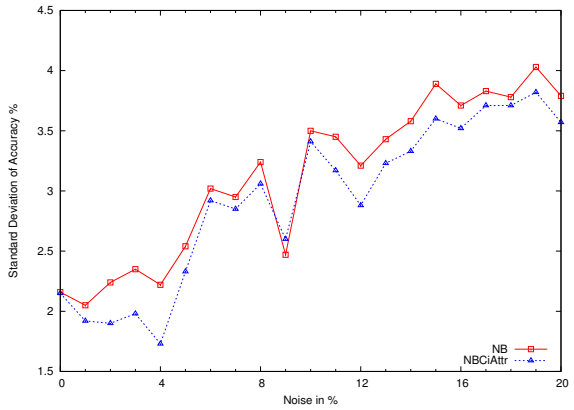


(e) Segment

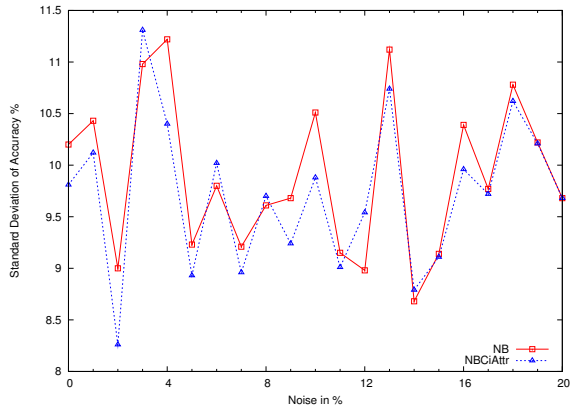


(f) Soybean

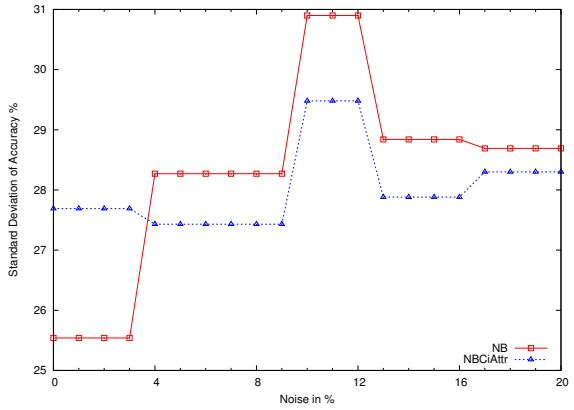
Figure 3.1: Accuracy with respect to Class Noise



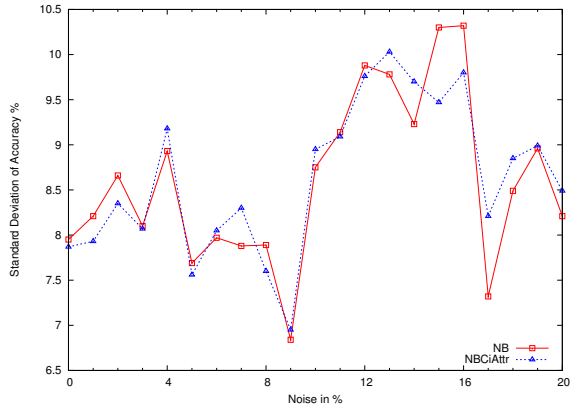
(a) Anneal



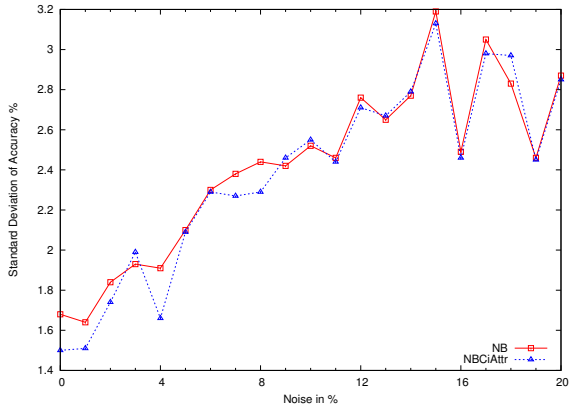
(b) Autos



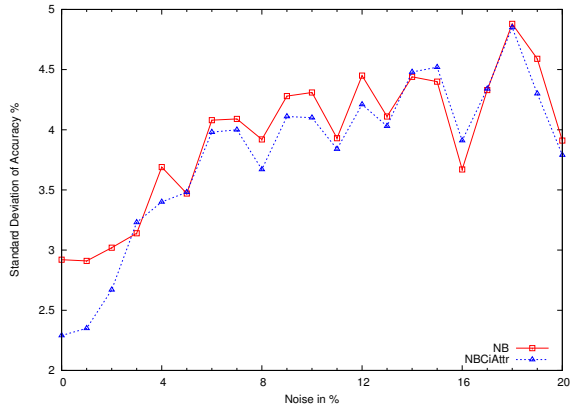
(c) Contact-lenses



(d) Glass



(e) Segment



(f) Soybean

Figure 3.2: Standard Deviation of Accuracy with respect to Class Noise

3.3 The Document Pipeline

We now describe the application of our CI-augmented NBCiAttr classifier to datasets other than the stock machine-learning UCI datasets. We construct classification problems using a real-world dataset in the field of document classification to lend support to the generality of our algorithm.

The source documents are legal documents from various county recording offices in Oregon and Washington that have been digitized using commodity scanners and then converted to text using the Tesseract OCR Engine [40]. The business problem is to automate the classification of these documents as belonging to a given class, a job currently performed manually by information workers. The problem space is rich due to the number of individual document types, 113 in all, and the multi-level taxonomy that defines the relationships between document types. At the top level of the taxonomy there are 5 logical groupings of documents, which we take as a 5-class classification problem. For this dataset, *TL5C*, the prior class distribution ranges greatly, as seen in Table 3.7.

Class	**Count**	**Pct**
Acknowledgement	88	1.99%
Legal	182	4.11%
Property	3548	80.07%
Claim	429	9.68%
Notice	184	4.15%

Table 3.7: Class Distribution for TL5C Dataset

At the bottom level of the taxonomy are the 113 different document types. We take the 6 most prevalent document types regardless of their taxonomic affiliation to create a 6-class dataset, *MT6C*, in which no single class dominates the prior class distribution (Table 3.8).

These datasets were prepared via the following steps. The documents are placed into

Class	Count	Pct
Deed of Trust	636	29.61%
Statutory Warranty Deed	290	13.50%
Appointment of Successor Trustee	237	11.03%
Full Reconveyance	229	10.66%
Substitution of Trustee and Deed of Reconveyance	515	23.98%
Deed of Reconveyance	241	11.22%

Table 3.8: Class Distribution for MT6C Dataset

a directory structure suitable for Weka's `TextDirectoryLoader` utility, which creates a "raw" document data ARFF file with 2 attributes, the class of the document and a string containing the document. Weka's `StringToWordVector` filter is then applied to the raw class file, converting it to a set of attributes, each of which represents a word (or *term*) found in the document. Only the n most prevalent terms across the corpus are included, configurable either on a per class basis or across all terms in the corpus. The value n and hence number of term attributes can be varied along with other parameters, such as the minimum number of occurrences of a term to be considered for inclusion as an attribute, any stemming to be applied to words in the raw dataset, and how the document string in the raw dataset is tokenized into words. The output of StringToWordVector is a dataset with numeric attributes for term counts and a nominal class attribute. The StringToWordVector filter supports a number of different counting methods for the term count attributes, including metrics typically used in document classification such as term frequency and inverse document term frequency. Because our goal is to create nominal datasets, the domain in which our CI-based classifiers are applicable, we eschew real-valued attributes in favor of these 2 representations: an integer count (0–k instances of a given term found in the document), and a binary term presence (0 or 1, indicating whether the term is found at least *minimum term count* times in the document). The settings used for our Document Pipeline experiment are summarized in Table 3.9. The

final step in dataset preparation is to convert the numeric attributes into nominal values, accomplished with the `NumericToNominal` filter.

Setting	Value
stemmer	weka.core.stemmers.LovinsStemmer
tokenizer	weka.core.tokenizers.AlphabeticTokenizer
minimum term count	1
minimum term count scope	per class
term attribute	integer count or binary presence
term count	10, 100 or 1000

Table 3.9: Document Pipeline StringToWordVector Settings

Once the datasets are prepared, we compare the performance of `NaiveBayes` to that of our CI-augmented classifiers, the results of which appear in Section 3.4. We also note that it is during the course of these experiments that we encountered issues with underflow. The impact of underflow and its effect on our classifier implementation is covered in Section 3.5

3.4 Document Pipeline Experiment Results

The experiments compare classifier accuracy, RMSE, and AUC for the two Document Pipeline datasets, each of which is present with (3) different terms counts and in both binary presence (*nom* suffix) and nominal term count variants (*nomtc*). Statistical relevance of the results are determined by Weka's Experimenter module using the paired corrected t-test. The range of term counts seeks to demonstrate that our classifier is not biased towards datasets with either few or many attributes; likewise the nominalized term count datasets dispel a preference for binary attribute values. The TL5C nominal term count datasets all include 9 attributes with 129 or more distinct values per attribute. In this evaluation we use a variant of `NBCiAttr` dubbed `NBCiAttrCiUF`; this classifier makes

accommodations for underflow and is introduced fully in Section 3.5.

Dataset	NB	NBCiAttrCiUF	
mt6c-10-nom	84.87 ± 2.29	84.79 ± 2.30	
mt6c-10-nomtc	94.16 ± 1.57	94.76 ± 1.53	∘
mt6c-100-nom	95.34 ± 1.41	96.71 ± 1.23	∘
mt6c-100-nomtc	96.39 ± 1.36	96.64 ± 1.34	∘
mt6c-1000-nom	95.42 ± 1.59	96.72 ± 1.28	∘
mt6c-1000-nomtc	96.71 ± 1.28	98.01 ± 0.97	∘
tl5c-10-nom	84.96 ± 1.27	86.74 ± 1.46	∘
tl5c-10-nomtc	81.02 ± 1.69	81.78 ± 1.56	∘
tl5c-100-nom	76.16 ± 2.44	78.25 ± 2.40	∘
tl5c-100-nomtc	82.93 ± 1.74	87.03 ± 1.39	∘
tl5c-1000-nom	44.38 ± 2.84	45.38 ± 2.86	∘
tl5c-1000-nomtc	61.24 ± 2.89	61.93 ± 2.75	
∘, • statistically significant improvement or degradation			

Table 3.10: Accuracy - Document Pipeline Datasets

The accuracy results are shown in Table 3.10. `NBCiAttrCiUF` outperforms `NaiveBayes` in accuracy for 10 of the 12 test datasets, and the standard deviations are comparable. As with the UCI datasets (see Section 2.4.2), the standard deviation is lower for a majority of the datasets (9 of 12). Table 3.11 documents the evaluation of RMSE, where the results are mixed. For the MT6C datasets, 2 wins are balanced by 2 ties and 2 losses (2/2/2), while the 10 and 100-term variants of TL5C datasets fare better with 4 wins, offset by losses for both of the 1000-term versions. The area under the ROC curve results in Table 3.12 are also mixed, with one statistical improvement for the MT6C dataset (1/5/0), and one improvement, three ties, and two degradations (1/3/2) for the TL5C dataset.

We do not find this result entirely surprising given the general difficultly that NB classifiers have with this dataset. Simply guessing the class based on the prior probability of the Property class for the 1000 term datasets well outperforms NB, yielding over 80%

Dataset	NB	NBCiAttrCiUF	
mt6c-10-nom	0.189 ± 0.012	0.188 ± 0.012	
mt6c-10-nomtc	0.126 ± 0.017	0.120 ± 0.018	∘
mt6c-100-nom	0.115 ± 0.018	0.100 ± 0.019	∘
mt6c-100-nomtc	0.105 ± 0.021	0.106 ± 0.020	
mt6c-1000-nom	0.120 ± 0.021	0.130 ± 0.017	•
mt6c-1000-nomtc	0.101 ± 0.021	0.214 ± 0.007	•
tl5c-10-nom	0.219 ± 0.009	0.212 ± 0.009	∘
tl5c-10-nomtc	0.261 ± 0.011	0.257 ± 0.011	∘
tl5c-100-nom	0.285 ± 0.014	0.273 ± 0.014	∘
tl5c-100-nomtc	0.256 ± 0.013	0.239 ± 0.012	∘
tl5c-1000-nom	0.465 ± 0.012	0.478 ± 0.010	•
tl5c-1000-nomtc	0.390 ± 0.014	0.416 ± 0.010	•
∘, • statistically significant improvement or degradation			

Table 3.11: RMSE - Document Pipeline Datasets

Dataset	NB	NBCiAttrCiUF	
mt6c-10-nom	1.000000 ± 0.000000	1.000000 ± 0.000000	
mt6c-10-nomtc	0.999993 ± 0.000027	0.999994 ± 0.000025	
mt6c-100-nom	0.999995 ± 0.000027	0.999995 ± 0.000027	
mt6c-100-nomtc	0.999992 ± 0.000035	0.997967 ± 0.004645	
mt6c-1000-nom	0.995261 ± 0.006133	0.999796 ± 0.000525	∘
mt6c-1000-nomtc	0.983265 ± 0.012908	0.989018 ± 0.008099	
tl5c-10-nom	0.932383 ± 0.038136	0.934524 ± 0.037563	∘
tl5c-10-nomtc	0.850876 ± 0.037811	0.846137 ± 0.040103	
tl5c-100-nom	0.920551 ± 0.052094	0.921045 ± 0.052690	
tl5c-100-nomtc	0.869363 ± 0.039091	0.872045 ± 0.038419	
tl5c-1000-nom	0.930677 ± 0.037959	0.847430 ± 0.064392	•
tl5c-1000-nomtc	0.897950 ± 0.041813	0.710352 ± 0.112131	•
∘, • statistically significant improvement or degradation			

Table 3.12: AUC - Document Pipeline Datasets

accuracy. A check of the confusion matrices for the TL5C dataset (Tables 3.13 and 3.14) shows that the classifiers have difficulty differentiating between the classes, particularly when binary presence is used. (Although it is remarkable to note that the accuracy of the minority classes—all save for Property—is increased when only the binary presence is used.) We acknowledge that better ML algorithms for text classification are available, but assert that our CI-augmented NB algorithm can be substituted for NB across a variety of nominal datasets without an impact to performance with respect to accuracy. We turn next topic of underflow and the rationale behind the `NBCiAttrCiUF` variant of `NBCiAttr`.

a	**b**	**c**	**d**	**e**	← **classified as**
0	0	57	31	0	**a** = Acknowledgement
0	38	74	70	0	**b** = Legal
0	0	2253	1295	0	**c** = Property
0	0	39	390	0	**d** = Claim
0	0	34	92	58	**e** = Notice

Table 3.13: Confusion Matrix for NBCiAttrCiUF on TL5C-1000-NOMTC

a	**b**	**c**	**d**	**e**	← **classified as**
46	10	15	13	0	**a** = Acknowledgement
3	150	10	19	0	**b** = Legal
175	285	1326	1309	453	**c** = Property
10	7	10	397	5	**d** = Claim
4	3	6	72	99	**e** = Notice

Table 3.14: Confusion Matrix for NBCiAttrCiUF on TL5C-1000-NOM

3.5 Underflow

One obstacle encountered during the evaluation our CI-augmented classifiers on the Document Pipeline datasets is that our initial implementation, based on the `NaiveBayesSimple`

found in Weka, failed during prediction of all 1000-term and the 100-term nominative term count datasets due to underflow.[2] We validate that the behavior is not due to our CI-related extensions as it is also exhibited by the `NaiveBayesSimple` classifier alone. In this context, this error occurs when all class predictions are zero and Weka's Utils.normalize() method is unable to assign relative percentages to the class distribution array because the sum of the raw prediction scores is 0. In the `NaiveBayesSimple` implementation no attempt is made to detect underflow; the product of the prior and attribute probabilities terms is returned as the result, causing a null prediction whenever all classes predictions underflow. This difficulty encouraged us to explore potential heuristics to cope with underflow, beginning with the approach taken in the `NaiveBayes` implementation.

3.5.1 Underflow Detection in Weka's `NaiveBayes`

The algorithm used within `NaiveBayes` is depicted in Listing 3.1; note that this code has been simplified to call out the salient aspects—namely that the outer loop is over attributes while the inner loop is over the class predictions. After each attribute is examined, the value of *max* is examined to see whether it is less than 1×10^{-75}. If so, then all of the probability predictions are scaled up by 1×10^{75} before the next attribute is added during the next loop iteration. The calculations use the Java primitive double type, represented internally as a 64-bit IEEE 754 number.[3]

[2]The specific exception observed in Weka is `java.lang.IllegalArgumentException: Can't normalize array. Sum is zero`.

[3]http://java.sun.com/docs/books/jls/third_edition/html/typesValues.html#4.2.3

```
for (int j = 0; j < m_NumClasses; j++) {
  probs[j] = m_ClassDistribution.getProbability(j);
}
for (int i = 0; i < m_NumAttributes; i++) {
  double temp, max = 0;
  for (int j = 0; j < m_NumClasses; j++) {
    temp = Math.max(1e-75, getAttributeProbability(i));
    probs[j] *= temp;
    if (probs[j] > max) {
      max = probs[j];
    }
    if (Double.isNaN(probs[j])) { // throw Exception }
  }
  if ((max > 0) && (max < 1e-75)) {
    // Danger of underflow; scale up all probabilities.
    for (int j = 0; j < m_NumClasses; j++) {
      probs[j] *= 1e75;
    }
  }
}
```

Listing 3.1: Underflow Protection in `NaiveBayes`

3.5.2 Underflow Heuristics for `NBCiAttr`

Our first step was to reimplement our CI-augmented classifier based on Weka's `NaiveBayes` source code using its scaling approach to underflow. This allowed evaluation to proceed, and the resulting classifier was equivalent in the absence of underflow to the `NBCiAttr` classifier used throughout our work. However, we were disappointed to note that this classifier did not perform as well across all of the Document Pipeline datasets as one of our interim attempts to solve the underflow problem in our original implementation. In that interim incarnation, when underflow is detected, the prediction results based on Bayes' rule are ignored, and the classifier returns the prior class probabilities. That is, the classifier merely guesses the most likely class. We call this classifier `NBCiAttrPUF` (PUF indicates Priors UnderFlow). We investigate another approach when Bayes' rule does not produce a prediction due to underflow, which is to instead base the prediction on the inverse of the prediction CIs calculated, as depicted in Equation (3.1). Recall that smaller prediction CIs indicate stronger conviction, and so choosing the largest inverse of

the prediction CI selects the class prediction with the strongest conviction.

$$pred_X(c) = \underset{c}{argmax}\ \frac{1}{pci_X(c)} \tag{3.1}$$

We refer to this classifier as `NBCiAttrCiUF`. It is compared to plain `NaiveBayes` (or simply NB), CI-augmented NaiveBayes, and our PUF heuristic alternative in Tables 3.15–3.17.[4] We observe that `NBCiAttrPUF` is able to provide a statistical improvement in accuracy over NB in 4 of 6 cases, and `NBCiAttrCiUF` in 5 of 6 cases, but that both approaches fail to deliver consistent improvements when RMSE or AUC is used as the metric. However by employing the underflow implementation (in Listing 3.1) in conjunction with CI (*i.e.* `NBCiAttr`), RMSE is improved across the board, and AUC results are comparable to or improved over those for NB. Focusing in on only the `NBCiAttr` and `NBCiAttrCiUF`, there are 3 out of 6 cases where the CiUF heuristic increases the overall classification accuracy by a full percentage point or greater. We conclude this line of inquiry with the observation that prediction CI can be a useful indicator of class when other mechanisms to predict class have been exhausted due to underflow. This information is immediately available at prediction time and can be employed by the classifier when underflow is detected. The heuristic incurs only trivial additional computational overhead, as merely the index of the smallest prediction CI (strongest conviction) class need be determined, requiring $O(c)$ time. In the next chapter we turn to the time complexity and run-time characteristics of naïve Bayesian classifiers in more detail.

[4]Dataset names abbreviated in the interest of space; the datasets are those used in other experiments.

Dataset	NB	NBCiAttr	NBCiAttrPUF	NBCiAttrCiUF
m100tc	96.392 ± 1.36	96.643 ± 1.34 ○	96.643 ± 1.34 ○	96.643 ± 1.34 ○
m1000	95.423 ± 1.59	95.493 ± 1.60	96.629 ± 1.34 ○	96.718 ± 1.28 ○
m1000tc	96.708 ± 1.28	96.741 ± 1.29	97.243 ± 1.11	98.007 ± 0.97 ○
t100tc	82.934 ± 1.74	83.579 ± 1.69 ○	86.985 ± 1.38 ○	87.028 ± 1.39 ○
t1000	44.381 ± 2.84	45.225 ± 2.86 ○	45.202 ± 2.88 ○	45.380 ± 2.86 ○
t1000tc	61.244 ± 2.89	61.950 ± 2.82 ○	61.763 ± 2.73	61.927 ± 2.75
○, ● statistically significant improvement or degradation				

Table 3.15: Accuracy Employing Various Approaches to Underflow

Dataset	NB	NBCiAttr	NBCiAttrPUF	NBCiAttrCiUF
m100tc	0.105 ± 0.02	0.101 ± 0.02 ○	0.105 ± 0.02	0.106 ± 0.02
m1000	0.120 ± 0.02	0.118 ± 0.02 ○	0.119 ± 0.02	0.130 ± 0.02 ●
m1000tc	0.101 ± 0.02	0.100 ± 0.02 ○	0.185 ± 0.01 ●	0.214 ± 0.01 ●
t100tc	0.256 ± 0.01	0.252 ± 0.01 ○	0.224 ± 0.01 ○	0.239 ± 0.01 ○
t1000	0.465 ± 0.01	0.462 ± 0.01 ○	0.462 ± 0.01 ○	0.478 ± 0.01 ●
t1000tc	0.390 ± 0.01	0.386 ± 0.01 ○	0.386 ± 0.01	0.416 ± 0.01 ●
○, ● statistically significant improvement or degradation				

Table 3.16: RMSE Employing Various Approaches to Underflow

Dataset	NB	NBCiAttr	NBCiAttrPUF	NBCiAttrCiUF
m100tc	1.0000 ± 0.0000	1.0000 ± 0.0000	0.9980 ± 0.0046	0.9980 ± 0.0046
m1000	0.9953 ± 0.0061	0.9953 ± 0.0061	0.99987 ± 0.0005 ○	0.9998 ± 0.0005 ○
m1000tc	0.9833 ± 0.0129	0.9833 ± 0.0129	0.9892 ± 0.0080	0.9890 ± 0.0081
t100tc	0.8694 ± 0.0391	0.8714 ± 0.0375	0.8733 ± 0.0382	0.8720 ± 0.0384
t1000	0.9307 ± 0.0380	0.9321 ± 0.0375 ○	0.8482 ± 0.0601 ●	0.8474 ± 0.0644 ●
t1000tc	0.8980 ± 0.0418	0.8986 ± 0.0416 ○	0.6571 ± 0.1125 ●	0.7106 ± 0.1121 ●
○, ● statistically significant improvement or degradation				

Table 3.17: AUC Employing Various Approaches to Underflow

Chapter 4

Run-time Performance

4.1 Run-time Analysis

As intimated in Section 1.4, modifications to NB with the aim of improving the classification accuracy of NB may trade accuracy for asymptotic run-time complexity. It is reasonable to expect that all conceivable variants execute in polynomial time—to exceed polynomial time an algorithm would have to perform super-polynomial or exponential work either with respect to the number of classes, instances or attributes. As we will see, not all implementations exhibit polynomial time complexity, and not all of those that do share the equivalent polynomial time complexity.

4.1.1 Time Complexity of Naïve Bayes

For a classification problem with n instances, k attributes, and c classes, the algorithm requires only a single pass over the training data in order to construct a model, which completes in $O(nk)$ time. Since k is frequently small, Elkan refers to this as linear in [4], going on to state that "no learning algorithm that examines all of its training data can be faster." Strictly speaking, the time complexity of many implementations is slightly higher.

Given v possible nominal values the attribute can assume, the model must contain kcv cells to store the counts for each attribute value, and ergo requires $O(kcv)$ time (and space) to allocate and initialize those counts. Then to subsequently compute the conditional probabilities for all attribute values requires again O(kcv) calculations. Implementations can be clever regarding efficient data structures to accumulate counts; for example Weka uses a data format that converts all nominal classes and attribute values to non-negative integers, allowing the class and attribute values to be used as indices directly into the model during training.

For implementations that support indexing by attribute value, each classification occurs in the $O(k)$ time needed to compute the class prediction, repeated c times (once per class), and then c comparisons to select the maximum, yielding $O(kc+c)$ or simply $O(kc)$. The normalization step required in order to be able to calculate RMSE and AUC then requires c more steps. This can be thought of as approaching k for many datasets, where $c \ll k$.

4.1.2 Time Complexity of CI-Augmented Naïve Bayes

Our CI-based classifiers maintain both the original attributes and naïve Bayes structure. During model creation, the attribute CI, $aci_{a_i}(c)$ (Equation (2.1)), and class confidence, $ci_c(c)$ (Equation (2.2)), terms are calculated and stored in the model. Equation (2.1) contains a term N_{a_i} that refers to the number of times an attribute is observed irrespective of class. These counts can be accumulated during training without requiring any additional time and allocate only k elements of memory alongside the c elements for the class CIs (Equation (2.2)) and the kcv elements to store the attribute CIs. Therefore, the memory requirements are slightly greater than double the memory required for NB.

When prediction is performed, the prediction confidence for the test instance $pci_X(c)$ (Equation (2.3)) is computed in $O(kc)$ time, just as with the NB, and then applied to the

prediction. To be sure, we are not claiming that the run-time performance of `NBCiAttr` is equivalent to NB, only the asymptotic time complexity. The are several constant factors difference between the two, not the least of which being the operations to compute $pci_X(c)$. This term entails $k+1$ multiplications (the squaring of the attribute and class CI terms) and then the calculation of a square root, which is far more CPU intensive than any of the operations in basic NB. We address some potential performance improvements in Section 5.2.

4.1.3 Time Complexity of NB Variants

We now take a cursory glance at the time complexity of other approaches to improving NB classification introduced in Section 1.4 to motivate why we believe that `NBCiAttr` is distinct in its ability to improve classification accuracy without requiring additional computational complexity. Much of the literature on hybrid variants of NB we encountered neglects to discuss time complexity directly, and we found scant treatment of the topic in the form of surveys. Therefore, we introduce the theoretical reasons for time complexity related to these approaches and then attempt to address the question empirically.

When performing feature selection via either filters or wrappers, an exhaustive search of the attribute space—*i.e.* trying all possible combinations of attributes—requires evaluation of 2^k attribute sets, and so is only feasible for datasets with small numbers of attributes. In practice, feature selection often employs heuristic or greedy local search methods to traverse the search space typically in $O(k^2)$ time [41, 23]. Both features and wrappers must then perform an evaluation of each proposed feature set in order to evaluate its goodness relative to competitors, so the overall training time is on the order of $O(k^3cv)$. Obviously ensemble learning methods pay the penalty of a constant factor equal to the number of classifiers deployed, and the asymptotic run-time is that of the most expensive classifier in the ensemble.

Hall's attribute-weighted naïve Bayes (AWNB) [20] learns attribute weights by constructing decision trees of attributes, and thus incurs the cost of tree construction. In the case of this algorithm the cost is log-linear in the number of instances, adding a factor of $O(\log n)$ to the training time. (Unfortunately we are not able to locate an implementation of this algorithm to investigate its implementation or evaluate its runtime performance. From the description Hall's paper, it seems that the weighting scheme incurs another factor of $O(\log k)$ when constructing the attribute tree.) It should be noted that one of the advantages of decision trees is that their traversal only requires log-based time in the number of attributes, a potential advantage over NB during prediction, which requires linear time to examine every attribute. However, in terms of training time the theoretical lower bound for tree construction for tree-based hybrids remains $O(k \log_2 k)$ for attributes or $O(n \log_2 n)$ for instances.

`HNB` (Hidden Naïve Bayes) [42] also seeks to learn weights and must compare all attributes pair-wise during training to do so. Jiang *et al.* document its training complexity as $O(nk^2 + ck^2v^2)$ and the classification time as $O(ck^2)$, and claim their algorithm exhibits better runtime performance than TAN (tree-augmented NB) [43], for which the training time is $O(nk^2 + ck^2v^2 + k^2 \log k)$. (This last term would be significant for datasets like those in the Document Pipeline, where $c \ll \log k$.) Next, we note the time complexity of `AODE`, which trains in $O(nk^2)$ time and classifies in $O(ck^2)$ time, but requires more space—$O(ck^2v^2)$. Finally, we estimate the training complexity of `DMNBtext` based on the description in [28].[1] In order to generate the discriminative frequency estimate (DFE) for each of k attributes, the algorithm performs an arbitrary (parameterized) number of iterations across the attributes to improve the frequency estimate. Each iteration and update of the classifier (*i.e.* for all n instances in the training set) requires c steps to calculate the current prediction distribution. Therefore, `DMNBtext` varies from NB only

[1] The authors compare training time in their evaluation, but do not posit the algorithm's computational complexity.

in the factor for c and the number of iterations, and the learning complexity is $O(cnkM)$, where M is the (constant) number of iterations desired to improve the DFE. There is also an additional factor of c required for prediction of multi-class datasets, as the core DMNB algorithm only supports binary prediction.

4.2 Run-time Experiment

In order to validate our assertions about the computational complexity of `NBCiAttr`, we devise an experiment to record the training and testing time of NB, `NBCiAttr`, and several of the hybrid NB algorithms introduced in Sections 1.4 and 4.1.3. For the basis of comparison, we include `J48`, a pure decision tree ML algorithm that is Weka's implementation of Quinlan's seminal C4.5 [44]. Our list of ML algorithms is: `NaiveBayes`, `NBCiAttr`, `HNB`, `BayesNet`, `AODE`, `DTNB`, `DMNBtext`, and `J48`. As related work we cite an experiment by Williams *et al.* [45] wherein the model creation and classification runtime of five ML algorithms applied to the topic of IP traffic flow classification are compared using Weka's Experimenter.

For input datasets we initially selected the 1000-term variants of the Document Pipeline datasets from Section 3.3 but discovered that it was not only `NaiveBayesSimple` that was incapable of coping with large numbers of attributes without generating exceptions due to underflow. Hidden Naïve Bayes (`HNB`) also fails during prediction of these high-attribute datasets. Therefore, we took the 10 and 100-term versions of the *MT6C* datasets and expanded the number of instances by a factor of 20. The intention here being to increase the amount of time spent training and testing so as not to encounter timer resolution fidelity issues in either the JVM or operating system. The statistics for resulting datasets are shown in Table 4.1.[2] Note that the average number of attribute values decreases as the

[2] `DMNBtext` only allows numeric attributes, and we use separate but equivalent datasets for its evaluation.

number of terms considered increases because the dataset incorporates more infrequently encountered terms.

Dataset	Attributes	Values (avg)	Classes	Instances
mt6c-10x20-nom	27	2.0	6	42960
mt6c-100x20-nom	262	2.0	6	42960
mt6c-1000x20-nom	2763	2.0	6	42960
mt6c-10x20-nomtc	27	65.8	6	42960
mt6c-100x20-nomtc	262	24.0	6	42960
mt6c-1000x20-nomtc	2763	7.1	6	42960

Table 4.1: Datasets Considered for Empirical Run-time Experiment

4.2.1 Pragmatic Issues

Experimentation using these datasets led to several other issues. The first was that after running for over 19 hours, the *first* of 10 runs of Hall and Frank's DTNB classifier against the first 100x20 dataset had not completed. Although not explicitly stated in the DTNB paper [30], the computational complexity is obviously high relative to other classifiers. We examine the classifier source code to ascertain that the algorithm performs a search the authors describe as "forward selection [for attributes to use for NB] and backward elimination [for attributes to use for the decision table]" as well as considering which attributes to drop entirely. This search appears to require up to $O(2^k)$ time, refuting the notion that all NB variants should run in polynomial time.

The next issue we encountered was related to space complexity. The authors of `HNB` do not make a claim as to its memory requirements, and we know that they are bounded above by the time complexity because the algorithm cannot cover more space than the allocated time. However the space complexity could be as great as the time complexity, which includes the potentially large terms for k^2v^2 ($2^{30.73}$ for mt6c-100x20-nomtc, for which v averages 24, but ranges to a maximum of 161) and nk^2 (over $2^{31.46}$ for either

of the 100x20 datasets). Even assuming only the storage of an array of 64-bit doubles of either of those values would far exceed the addressable memory of a 32-bit operating system. We obtained access to a 64-bit JVM, which supports the creation of larger heaps, but the physical memory of the system itself was a limiting factor—it is not valid to compare the time required by an algorithm that is running in core to one for which virtual memory is being swapped to disk. Empirically we can assert that the nk^2 term does not appear in the space complexity, as we are able to process the mt6c-100x20-nom version of the dataset on our system without swapping.

We finally settled on a few compromises to our envisioned experiment. We drop the *nomtc* variants of the datasets entirely, exclude the `DTNB` algorithm, and exclude the 1000x20 datasets for the `AODE` and `HNB` algorithms. Also, instead of conducting 10 runs of 10-fold cross-validation, we use 10 runs with a 66% training, 34% test split of the datasets. This reduces the overall number of runs by a factor of 10, and slightly decreases the memory requirements during training as only 66% of the dataset is analyzed instead of 90%. We assert that these concessions benefit the algorithms with higher computational complexity than NB and `NBCiAttr`, but still allow us to formulate a relative picture of their runtime performance.

Evaluation was conducted on a headless (i.e. no graphics adapter or input devices) 2.2GHz dual-core system running 64-bit Linux. No other tasks or users were allowed on the system during the experiment to minimize interference from other jobs. By default Weka executes experiments sequentially within a single thread, a useful property for our test because we could depend on sufficient head-room on the second core to handle kernel tasks.

4.2.2 Results and Discussion

Results are those reported by Weka's *UserCPU_Time* fields and so represent the operating system's view of training and testing time. We represent training and testing in Tables 4.2 and 4.3. Empty cells indicate that the algorithm was not evaluated for that dataset.

	Datasets		
Algorithm	mt6c-10x20-nom	mt6c-100x20-nom	mt6c-1000x20-nom
NaiveBayes	0.17	1.73	35.05
NBCiAttr	0.16	1.63	33.34
HNB	0.28	27.19	-
BayesNet	0.58	7.31	104.94
AODE	0.25	36.88	-
J48	3.36	26.1	297.24
DMNBtext	4.61	23.76	65.04

Table 4.2: Training Time for Various ML Algorithms

	Datasets		
Algorithm	mt6c-10x20-nom	mt6c-100x20-nom	mt6c-1000x20-nom
NaiveBayes	0.29	2.93	49.73
NBCiAttr	0.58	6.18	85.41
HNB	1.24	235.81	-
BayesNet	0.39	4.01	96.91
AODE	0.91	83.2	-
J48	0.08	0.10	0.16
DMNBtext	0.04	0.14	0.32

Table 4.3: Testing Time for Various ML Algorithms

We note that training time for NB and `NBCiAttr` are consistent with each other across all datasets, and that testing time is greater for the CI-augmented classifier by approximately a factor of 2. This corresponds well to our expectations in Section 4.1.2; we anticipate a higher constant factor in order to calculate and apply the prediction CI. One puzzling aspect of the results is that both `NaiveBayes` and `NBCiAttr` lose ground when

we make the jump from the 100-term to the 1000-term dataset. We do not have a ready explanation for this discrepancy, as the computational complexity of both algorithms scales linearly in the number of attributes, and the number of instances is constant across all three datasets. Our conjecture is that there is an implementation-related factor at play here. Another subtle oddity is that `NBCiAttr` trains slightly faster than `NaiveBayes`. There is no reason to anticipate this, as `NBCiAttr` actually calculates a few extra model statistics during training (the attribute CI and class CI terms in Equations (2.1) and (2.2)). To verify, we executed the benchmark multiple times, including reversing the order of algorithm evaluation (in an attempt to expose any bias due to the first algorithm having to load the dataset prior it it being cached by the operating system), and the results do not change.

As expected we observe higher training times for hybrid variants, and a hodgepodge of testing times. The algorithms that employ tree structures take longer to train, but demonstrate the beauty of this data structure when prediction is performed. By comparison, `HNB` and `AODE` both exhibit longer testing times than training times (and this for testing sets half as large as the training set), a property they share with NB and `NBCiAttr`. Distinct from NB, both `HNB` and `AODE` appear to defer work at training time in favor of the creation of data structures and computation during testing.

`DMNBtext` stands out, both in terms of its classification accuracy[3] and runtime performance. For the experiment we set the iterations parameter for this classifier to 10, which should result in a linear increase in training time over NB, but the empirical results contradict this. We note that the algorithm could employ a stopping criterion to detect when subsequent iterations do not improve the DFE, but analysis of the implementation does not indicate the presence of such. The data structures used `DMNBtext` are lighter-weight than those used in Weka's `NaiveBayes` (simply arrays instead of Weka's Estimator

[3] `DMNBtext` is a very accurate classifier for text classification problems like those discussed in Section 3.3.

objects). At classification time, the dramatic difference is almost certainly due to the use of log-based calculations in the implementation—*i.e.* addition instead of multiplication.

We make one final comparison between the run-time of the algorithms, depicted in Figures 4.1 and 4.2. To generate this graphs, we normalize the times reported in Tables 4.2 and 4.3 twice. First we adjust for the number of attributes in our benchmark datasets (see Table 4.1); *i.e.* we scale the values for the 10x20 dataset by roughly a factor of 1000 (or exactly, 2763/27) and the values for the 100x20 dataset by a factor of about 10 (2763/262). We then normalize those results such that the time required for NB to train and test on the 42960 instances for the 10x20 dataset sums to 1. In essence, we are assuming that NB learning the 10-term dataset is the benchmark problem and want to observe how other algorithms compare as we vary the number of attributes. Therefore, the other values on the bar graph are relative factors of that the NB "unit" time. Figure 4.2 omits the long-running `AODE` and `HNB` algorithms for the sake of fidelity.

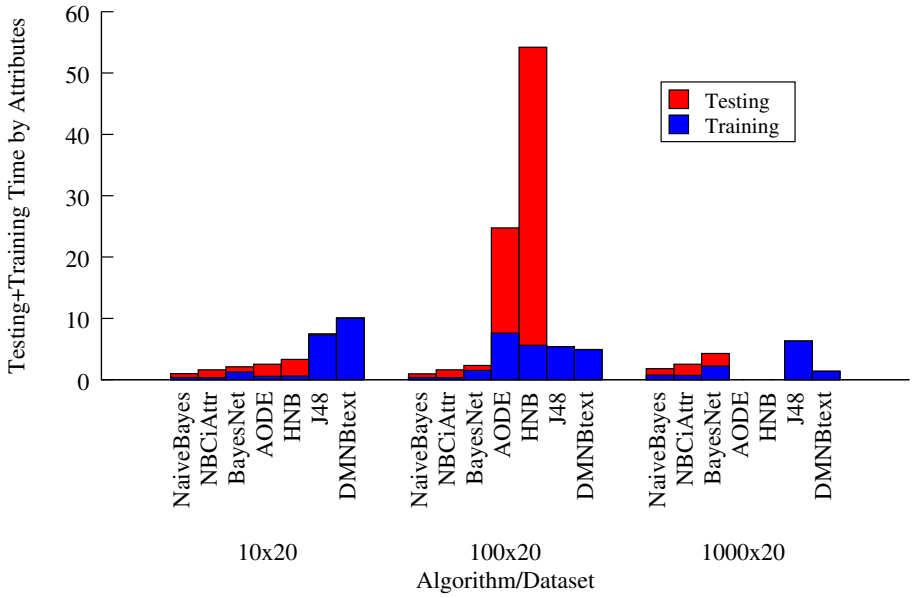


Figure 4.1: Training+Testing Time Normalized by Attribute Count

We note that the combined time for NB is the best; this is true regardless of the normalization. However `DMNBtext` gains relative speed as the number of attributes increase,

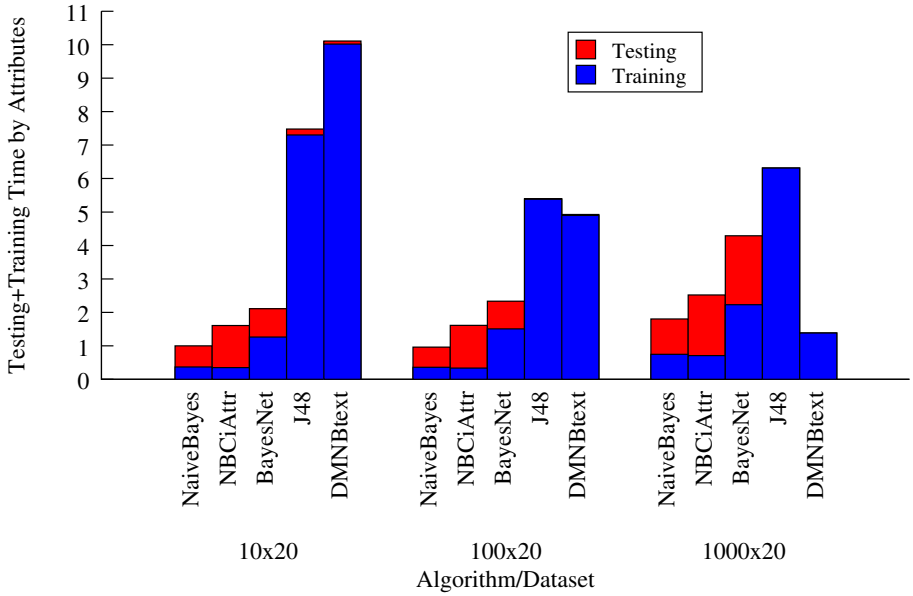


Figure 4.2: Training+Testing Time Normalized by Attribute Count (Detail)

and both it and `J48` spend so little time performing predictions so as not even to be visible on the graph.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

Our initial concept, to investigate the interaction between the confidence intervals (conviction) associated with conditional probabilities and class priors, led us through multiple ancillary topics: different metrics for the evaluation of classifier performance, analysis of performance in the presence of noise, and implementation issues such as the Laplace estimator and underflow. The goal of creating a classifier that makes use of confidence intervals has been realized. After several false starts documented in Section 2.4, we identified an approach applicable to multi-class datasets that warrants additional exploration and go on to evaluate `NBCiAttr` against laboratory datasets and real-life datasets using three diverse metrics of machine learning classifier performance.

This evaluation demonstrated a moderate improvement in classifier accuracy for some datasets without any statistically significant loss of accuracy. When considering RMSE and AUC, the results were similar to those for accuracy for the laboratory datasets, but not consistently statistically equivalent when working with real-world datasets containing large numbers of features. We hypothesize that this decrease in probability estimation and ranking ability is due to having numerous low conviction (infinite CI) attributes that are

in fact irrelevant to classification, but disrupt the prediction probabilities. Our conjecture that the use of CI may help classifier performance when class noise is present in the dataset is not supported by our experiments. Classifier accuracy for our CI-augmented classifier essentially tracked that of NB, although we acknowledge that our experiments only consider class noise and not attribute noise. We also experimented with two heuristics that can be utilized in absence of a prediction from the naïve Bayesian calculation due to underflow and observed an improvement in classifier accuracy in the majority of cases. Because the heuristics employ information directly available from the model, they can be used without any increase time or space requirements.

Given that `NBCiAttr` exhibited improved prediction performance in only a subset of our test cases, we sought to motivate its use by referring to its computational time and space complexity, which is asymptotically equivalent to naïve Bayes. We demonstrated this empirically, comparing the training and testing time of NB, `NBCiAttr` and several variants we deem related to our work. We conclude that `NBCiAttr` lives up to its promise, but is, along with all others, outclassed by Su *et al.* 's `DMNBtext` classifier in text classification situations where numeric instead of nominal attribute values are applicable. We thus close our exploration of confidence intervals and naïve Bayesian classification by documenting areas left unaddressed by this thesis and ideas for future work.

5.2 Future Work

As noted in Section 1.4, there is a great deal of prior work on improving naïve Bayes; much of that prior work focuses on either attribute selection (to address the naïve assumption of attribute independence) or the development of hybrid classifiers that combine NB with other ML algorithms. The drawback to these approaches is additional, in some cases significant, computational overhead for the improvement in accuracy realized. In applications where the model must be constructed quickly and classification should be

linear in the number of instances, we believe our CI-augmented classifiers to be unique and hold promise. We conclude with some open questions and ideas for future work in this area.

Despite the moderate successes of `NBCiAttr`, it is somewhat unsatisfying that the calculated prediction CI values cannot be applied directly to the conditional probabilities in Bayes' rule, as they are when the calculations are done in log-space as they are in Orange. Intuitively, it would be reassuring to look at an attribute CI and state that it affects the conditional probability by $\pm 3\%$, for example. However, we note that such an interpretation does not affect classifier accuracy and would potentially necessitate additional research into the alternatives for calculating probabilistic error and ranking-based metrics such as RMSE and AUC, as now predictions overlap by some margin of uncertainty.

We are aware that the class CI calculation in Equation (2.2) differs from the standard error calculation found in standard texts on statistics, which is $\pm\, z_{\alpha/2}\sqrt{p(1-p)/n}$ (referred to as *confidence intervals for proportions* in [35, p. 366]). Class CI uses the same terms, but all three of them appear in the denominator of the fraction. We would find this more troubling if there were a theoretical basis for applying the margin of error as a term when calculating $P(c|X)$. In the absence of this type of interpretation, we hypothesize that `NBCiAttr` functions by decreasing the highest conviction prediction by the least amount, while lower conviction predictions are more likely to be rejected by the classifier. Nonetheless, we would like to develop a mathematical model that explains the success of the `NBCiAttr` classifier. This would allow for a more comprehensive heuristic regarding when to use this classifier than "only useful for multi-class nominal datasets." Because `NBCiAttr` does not incorporate any truly new information into the model creation or prediction (the counts were already known), there could be an interpretation of it as shallow Bayes network a fixed structure that incorporates the prediction CI calculation.

We find the evidence that the use of prediction CI decreases the standard deviation in classifier accuracy (see Table 3.1) in a majority of cases interesting. The fact that the accuracy across 100 runs of the CI-augmented classifier remains more tightly clustered around the average accuracy implies that this classifier is more consistent and stable as the instances in the evaluated folds vary. We speculate as to whether this indicates a NB model less susceptible to over-fitting. Therefore we propose an experiment to evaluate resistance to overfit, such as that conducted by Dietterich in [24]. More specifically, future research includes conducting an evaluation of `NBCiAttr` in the presence of attribute noise instead of or in addition to class noise. Because the class CI and attribute CI will be using the proper (*i.e.* noise-free) class priors, `NBCiAttr` may function to filter attribute-value outliers by assigning them a weak conviction (high attribute CI).

Another avenue of potential inquiry is to explore the relationship of confidence intervals and the m-estimate (Section 1.4.3) and the DFE discriminative frequency estimate used in DMNBtext. In that work, the NB conditional probability terms are supplanted by their respective DFE counterparts. We ponder whether the prediction CI could be used in conjunction with the m-estimate or the DMNBtext prediction, or the attribute CIs as a component a stopping criterion during the DFE calculation. In the course of research on variants of NB, the m-estimate and DFE ideas (along with confidence intervals) strike us as the most promising approaches to improving NB while maintaining its desirable asymptotic complexity.

From our early work with log odds ratio (LOR) based NB classifiers, we believe that LOR-based classifiers are a potential area for research into a quicker NB classifier. The LOR-based NB classifier has the same asymptotic complexity for model construction, but with a slightly higher constant due to the more computationally expensive calculations. However, it enjoys a much smaller constant during prediction tasks, which can be done with addition, and any exponentiation for attribute weighting becomes multiplication.

The is evidenced by the superlative run-time of the DMNBtext classifier for large numbers of attributes seen in Section 4.2.2. Although the training task dominates in the laboratory where 90/10% folds are the norm, the prohibitive cost of manually classifying training data in non-synthetic environments suggests that trading training performance for run-time performance is warranted. Furthermore, the use of logarithms may avoid issues with underflow when large numbers of attributes are present as discussed in Section 3.5, or on architectures where high precision calculation come at a premium, *e.g.* in small embedded systems or wireless sensor networks. This effort is simply one of implementation. Finally, we conclude with a list of implementation-specific proposals to increase the usefulness of our CI-augmented NB classifier:

- Modify the CI-based classifiers to allow numeric attributes by ignoring these when CI is calculated (behavior would fall back to normal NB for those attributes). This extends the usefulness of this classifier to support datasets containing a mixture of nominal and numeric attributes.

- Modify the CI-based classifiers to support updates to the model after initial training. This merely entails a modification of the implementation to calculate the class and attribute confidence intervals whenever the model is updated. Prediction confidence intervals are calculated as they are currently.

- Optimize the run-time performance of `NBCiAttr` (at the expense of memory required) by caching the prediction confidence intervals $pci_X(c)$ as they are calculated. Similarly, once an instance prediction has taken place, subsequent instances with the same attribute values need not be calculated at all. This would serve to reduce the constant factor `NBCiAttr` pays over NB.

Bibliography

[1] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, December 2002.

[2] M. Možina, J. Demšar, M. Kattan, and B. Zupan, "Nomograms for visualization of naive bayesian classifier," in *PKDD '04: Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, (New York, NY, USA), pp. 337–348, Springer-Verlag New York, Inc., 2004.

[3] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, June 2005.

[4] C. Elkan, "Naive bayesian learning," September 1997.

[5] C. Ferri, J. Hernández-Orallo, and R. Modroiu, "An experimental comparison of performance measures for classification," *Pattern Recognition Letters*, vol. 30, no. 1, pp. 27–38, 2009.

[6] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *IJCAI*, pp. 1137–1145, 1995.

[7] A. Asuncion and D. Newman, "UCI machine learning repository." http://archive.ics.uci.edu/ml/, 2007.

[8] F. Provost and P. Domingos, "Tree induction for probability-based ranking," *Machine Learning*, vol. 52, pp. 199–215, September 2003.

[9] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, pp. 861–874, June 2006.

[10] D. J. Hand and R. J. Till, "A simple generalisation of the area under the roc curve for multiple class classification problems," *Machine Learning*, vol. 45, no. 2, pp. 171–186, 2001.

[11] P. Paclik, C. Lai, J. Novovicova, and R. Duin, "Variance estimation for two-class and multi-class roc analysis using operating point averaging," in *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pp. 1–4, Dec. 2008.

[12] B. Van Calster, V. Van Belle, G. Condous, T. Bourne, D. Timmerman, and S. Van Huffel, "Multi-class auc metrics and weighted alternatives," in *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pp. 1390–1396, June 2008.

[13] S. Vanderlooy and E. Hüllermeier, "A critical analysis of variants of the auc," *Machine Learning*, vol. 72, pp. 247–262, Sep 2008.

[14] J. Huang, J. Lu, and C. X. Ling, "Comparing naive bayes, decision trees, and svm with auc and accuracy," pp. 553–556, 2003.

[15] C. X. Ling, J. Huang, and H. Zhang, "Auc: a statistically consistent and more discriminating measure than accuracy," in *Proceedings of IJCAI 2003*, pp. 329–341, Morgan Kaufmann, 2003.

[16] P. Domingos and M. J. Pazzani, "Beyond independence: Conditions for the optimality of the simple bayesian classifier," in *International Conference on Machine Learning*, pp. 105–112, 1996.

[17] H. Zhang, "The optimality of naive bayes," in *FLAIRS 17: Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference* (V. Barr, Z. Markov, V. Barr, and Z. Markov, eds.), AAAI Press, 2004.

[18] M. Martinez-Arroyo and L. E. Sucar, "Learning an optimal naive bayes classifier," in *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, vol. 3, pp. 1236–1239, 2006.

[19] L. Jiang, D. Wang, and Z. Chi, "Scaling up the accuracy of bayesian network classifiers by m-estimate," in *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence* (D.-S. Huang, L. Huette, and M. Loog, eds.), vol. 4682 of *Lecture Notes in Computer Science*, pp. 475–484, Springer Berlin / Heidelberg, 2007.

[20] M. Hall, "A decision tree-based attribute weighting filter for naive bayes," *Knowledge-Based Systems*, vol. 20, pp. 120–126, March 2007.

[21] L. Jiang, D. Wang, Z. Cai, and X. Yan, *Survey of Improving Naive Bayes for Classification*, vol. 4632 of *Lecture Notes in Computer Science*, ch. Survey of Improving Naive Bayes for Classification, pp. 134–145. Springer Verlag, Heidelberg, DE, 2007.

[22] R. Kohavi, "Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid," in *Second International Conference on Knoledge Discovery and Data Mining*, pp. 202–207, AAAI, 1996.

[23] M. Hall, *Correlation-based Feature Selection for Machine Learning*. PhD thesis, University of Waikato, 1999.

[24] T. G. Dietterich, "Ensemble methods in machine learning," *Lecture Notes in Computer Science*, vol. 1857, pp. 1–15, 2000.

[25] H. Zhang and S. Sheng, "Learning weighted naive bayes with accurate ranking," *Data Mining, IEEE International Conference on*, vol. 0, pp. 567–570, 2004.

[26] D. M. Chickering, D. Geiger, and D. Heckerman, "Learning bayesian networks is NP-Hard," Tech. Rep. MSR-TR-94-17, Microsoft research, November 1994.

[27] L. Jiang, H. Zhang, and Z. Cai, "A novel bayes model: Hidden naive bayes," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, pp. 1361–1371, December 2008.

[28] J. Su, H. Zhang, C. X. Ling, and S. Matwin, "Discriminative parameter learning for bayesian networks," in *ICML '08: Proceedings of the 25th international conference on Machine Learning*, (New York, NY, USA), pp. 1016–1023, ACM, 2008.

[29] G. I. Webb, J. R. Boughton, and Z. Wang, "Not so naive bayes: Aggregating one-dependence estimators," *Machine Learning*, vol. 58, pp. 5–24, Jan 2005.

[30] M. Hall and E. Frank, "Combining naive bayes and decision tables," in *FLAIRS 21: Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference*, pp. 318–319, AAAI Press, 2008.

[31] S. Džeroski, B. Cestnik, and I. Petrovski, "Using the m-estimate in rule induction," *Journal of Computing and Information Technology*, vol. 1, pp. 37–46, 1993.

[32] T. L. Hankins, "Blood, dirt, and nomograms: A particular history of graphs," *Isis*, vol. 90, no. 1, pp. 50–80, 1999.

[33] B. Zupan and J. Demšar, "From experimental machine learning to interactive data mining," white paper, Faculty of Computer and Information Science, University of Ljubljana, 2004.

[34] A. Jakulin, M. Možina, J. Demšar, I. Bratko, and B. Zupan, "Nomograms for visualizing support vector machines," in *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, (New York, NY, USA), pp. 108–117, ACM, 2005.

[35] R. V. Hogg and E. A. Tanis, *Probability and Statistical Inference*. Macmillian Publishing Company, 3 ed., 1988.

[36] T. Mancill and S. A. Wallace, "Confidence-based tuning of nomogram predictions," in *FLAIRS 22: Proceedings of the Twenty-Second International Florida Artificial Intelligence Research Society Conference*, AAAI Press, 2009.

[37] Y. Li, L. F. A. Wessels, D. de Ridder, and M. J. T. Reinders, "Classification in the presence of class noise using a probabilistic kernel fisher method," *Pattern Recognition*, vol. 40, no. 12, pp. 3349–3357, 2007.

[38] X. Zhu and X. Wu, "Class noise vs. attribute noise: A quantitative study," *Artificial Intelligence Review*, vol. 22, no. 3, pp. 177–210, 2004.

[39] C. E. Brodley and M. A. Friedl, "Identifying mislabeled training data," *Journal of Artificial Intelligence Research*, vol. 11, pp. 131–167, 1999.

[40] R. Smith, "An overview of the tesseract ocr engine," in *ICDAR '07: Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2*, (Washington, DC, USA), pp. 629–633, IEEE Computer Society, 2007.

[41] P. Langley and S. Sage, "Induction of selective bayesian classifiers," in *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pp. 399–406, 1994.

[42] H. Zhang, L. Jiang, and J. Su, "Hidden naive bayes," in *Proceedings of the 20th National Conference on Artificial Intelligence*, (Stanford), pp. 919–924, AAAI, AAAI Press, 2005.

[43] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine Learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[44] J. R. Quinlan, *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

[45] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification," *SIGCOMM Computer Communication Review*, vol. 36, no. 5, pp. 5–16, 2006.