Semi-analytical Method for Analyzing Models and Model Selection Measures based on Moment Analysis

Amit Dhurandhar asd@cise.ufl.edu
Computer and Information Science and Engineering
University of Florida
Gainesville, FL 32611, USA

Alin Dobra adobra@cise.ufl.edu
Computer and Information Science and Engineering
University of Florida
Gainesville, FL 32611, USA

Editor:

Abstract

In this paper we propose a moment based *semi-analytical* method for studying models and model selection measures. By focusing on the probabilistic space of classifiers induced by the classification algorithm rather than on of datasets, we obtain efficient characterizations for computing the moments which is followed by visualization of the resulting formulae that are too complicated for direct interpretation. By assuming the data to be drawn i.i.d. from the underlying probability distribution, and by going over the space of all possible datasets, we establish general relationships between the Generalization error, Hold-out-set error, Cross-validation error and Leave-one-out error. We later exemplify the method and the results by studying the behavior of the errors for the Naive Bayes Classifier.

Keywords: Hold out set, Cross validation, Generalization Error, Naive Bayes Classifier

1. Introduction

Most of the work in machine learning is dedicated to designing new learning methods or better understanding, at a macroscopic level (i.e. performance over various datasets), the known learning methods. The body of work that tries to understand microscopic behavior of either models or methods to evaluate models – which we think is crucial for deepening the understanding of machine learning techniques and results – and establish solid connections with Statistics is rather small. The two prevalent approaches to establish such results are based on either theory or empirical studies but usually not both, unless empirical studies are used to validate the theory. While both methods are powerful in themselves, each suffers from at least a major deficiency.

The theoretical method depends on *nice*, closed form formulae that usually restricts the types of results that can be obtained to asymptotic results or statistical learning theory(SLT) type of results(Vapnik, 1998). Should formulae become large and tedious to manipulate, the theoretical results are hard to obtain and use/interpret.

The empirical method is well suited for validating intuitions but is significantly less useful for finding novel, interesting things since large number of experiments have to be

conducted in order to reduce the error to a reasonable level. This is particularly difficult when small probabilities are involved, making the empirical evaluation impractical in such a case.

An ideal scenario, from the point of view of producing interesting results, would be to use theory to make as much progress as possible but potentially obtaining uninterpretable formulae, followed by visualization to understand and find consequences of such formulae. This would avoid the limitation of theory to use only *nice* formulae and the limitation of empirical studies to perform large experiments. The role of the theory could be to *significantly* reduce the amount of computation required and the role of visualization to understand the potentially complicated theoretical formulae. This is precisely what we propose in this paper, a new hybrid method to characterize and understand models and model selection measures. What makes such an endeavor possible is the fact that, mostly due to the linearity of expectation, moments of complicated random variables can be computed exactly with efficient formulae, even though deriving the exact distribution in the form of small close formulae is a daunting task.

The specific contributions we make in this paper are:

1. We propose a new methodology to analyze, at a microscopic level, statistical behavior of models and model evaluation measures. The methodology is based on defining random variables for quantities of interest, exactly computing their moments and then understanding their behavior by visualization.

2. For the concrete example of model selection measures – we focus on hold-out-set error, cross validation error and Leave-one-out error. We establish connections between the moments of these three error measures and the generalization error indicating efficient strategies for computing these moments. By introducing a probability space over the classifiers and computing the moments of the generalization error, we have the following two advantages over the theoretical results given by *Statistical Learning Theory*: (a) we obtain qualitative results about classifiers based on the actual classification algorithms, not the expressiveness of the class of functions to which the classifier belongs to, and (b) the results are not as pessimistic.

3. We exemplify the general theory by comparing the behavior of the three model selection measures for the Naive Bayes Classifier(NBC). The data generation model we consider for the discrete case is truly the most generic, in that it can be used to represent any discrete data distribution over the input output space. It is important to note that in the simulations we present there is no explicit data being generated but instead the formulations are used to obtain the plots.

4. The method proposed can be used as an exploratory tool to observe the *non-asymptotic* behavior of the errors for a particular model under desired conditions, staying away from generalized statements which are more prone to error. The method encapsulates the generalization of theory and specificness of experiments bearing in mind scalability.

The rest of the paper is organized as follows. Section 2 is a brief survey of the related work. In Section 3 we mention the neccessity to put a classifier into a class and characterize

this class. In Section 4 we derive relationships between the moments of the Generalization error, Hold out set error, Multifold cross validation error and Leave-one-out error which are independent of any classification algorithm. Section 5 discusses the aspects critical to the methodology we propose. Section 6 is an illustration of the actual methodology applied to a Naive Bayes Classifier. It discusses the application of Theorem 1 in the paper and reports simulation studies for the NBC with one input. Section 7 and Section 8 discuss the problems involved in directly extending the method to NBC with multiple inputs. We also discuss strategies to solve these problems and maintain scalability of the proposed method. In Section 9 we report simulation studies for the multi-input NBC and discuss their implications. We finally, discuss promising lines for future research and summarize the major developments in the paper in Section 10.

2. Related Work

There is a large body of both experimental and theoretical work that addresses the problem of understanding various model selection measures. Shao (Shao, 1993) showed that asymptotically Leave-one-out(LOO) chooses the best but not the simplest model. Devroye, Gyor, and Lugosi (L. Devroye and Lugosi, 1996) derived distribution free bounds for cross validation. The bounds they found were for the nearest neighbour model. Breiman (Breiman, 1996) showed that cross validation gives an unbiased estimate of the first moment of the Generalization error. Though cross validation has desired characteristics with estimating the first moment, Breiman stated that its variance can be significant. Theoritical bounds on LOO error under certain algorithmic stability assumptions were given by Kearns and Ron (Kearns and Ron, 1997). They showed that the worst case error of the LOO estimate is not much worse than the training error estimate. Elisseeff and Pontil (A and M, 2003) introduced the notion of training stability. They showed that even with this weaker notion of stability good bounds could be obtained on the generalization error. Blum, Kalai and Langford (Blum et al., 1999) showed that v-fold cross validation is at least as good as $\frac{N}{v}$ hold out set estimation on expectation. Kohavi (Kohavi, 1995) conducted experiments on Naive Bayes and C4.5 using cross-validation. Through his experiments he concluded that 10 fold stratified cross validation should be used for model selection. Moore and Lee proposed heuristics to speed up cross-validation (Moore and Lee, 1994). Plutowski's (Plutowski, 1996) survey included papers with theoritical results, heuristics and experiments on cross-validation. His survey was especially geared towards the behavior of cross-validation on neural networks. He inferred from the previously published results that cross-validation is robust. More recently, Bengio and Grandvalet (Bengio and Grandvalet, 2003) proved that there is no universally unbiased estimator of the variance of cross-validation. Zhu and Rohwer (Zhu and Rohwer, 1996) proposed a simple setting in which cross-validation performs poorly. Goutte (Goutte, 1997) refuted this proposed setting and claimed that a realistic scenario in which cross-validation fails is still an open question.

The work we present here covers the middle ground between these theoretical and empirical results by allowing classifier specific results based on moment analysis. Such an endeavor is important since the gap between theoretical and empirical results is significant(Langford, 2005).

3. Statistical Characterization of Classifiers

Classifiers produced by learning algorithms need to perform well with respect to the underlying probability distribution for the problem at hand rather than produce good results on datasets available during learning. This immediately leads to the notion of *generalization error*, and empirical errors such as *hold-out-set*, *cross validation*, *Leave-one-out*. When the amount of testing data is very large, the generalization error of any given classifier is well approximated by any empirical error. In such a case the performance of any classifier can be estimated accurately by these empirical errors. Unfortunately, the amount of testing data can be small for namely two reasons, the amount of available data is small and a large part of the data needs to be used for training. With these observations it is can be seen that a given classifier cannot be evaluated accurately enough with respect to the underlying distribution since large amount of testing data is not available and the distribution is unknown. This remains true even if the learning process is disregarded since the total amount of available data is usually not large enough to warrant use of asymptotic results.

The main difficulty with direct characterization of a given classifier without extra information is the fact that the behavior of the empirical errors cannot be determined independent of the underlying probability distribution, thus no statistical statements can be made about the characterization produced. To avoid this difficulty, an indirect characterization of the classifier could be produced by characterizing a *class of classifiers* to which this classifier belongs to. Identifying such a class of classifiers and characterizing them seems the only practical method to produce a statistical characterization for a given classifier.

Statistical Learning Theory(SLT) categorizes classification algorithms(actually the more general learning algorithms) into different classes called Concept Classes. The concept class of a classification algorithm is determined by its Vapnik-Chervonenkis(VC) dimension which related to the shattering capability of the algorithm. Distribution free bounds on the generalization error of a classifier built using a particular classification algorithm belonging to a concept class are derived in SLT. The bounds are functions of the VC dimension and the sample size. The strength of this technique is that by finding the VC dimension of an algorithm we can derive error bounds for the classifiers built using this algorithm without ever referring to the underlying distribution. A fallout of this very general characterization is that the bounds are usually loose(Olivier Bousquet and Lugosi, 2005; Williamson, 2001) which in turn result in making statements about any particular classifier weak.

The idea we pursue in this paper is to define the class of classifiers as the classifiers induced by a given learning algorithm and i.i.d. data of a given size. Any member of this class can be viewed as a *sample classifier*, and the characterization of the class is strongly connected to the behavior of the classifier. This class of classifiers are much smaller than the classes considered in SLT. With this in the next section we establish relationships between the statistical behavior of the generalization error and empirical errors for the members of this class of classifiers and, relationships that have the character of general results(they hold irrespective of the classification algorithm that induces the class). The downside of our method is the fact that we loose the strength to make generalized statements to the extent that SLT makes. In order to obtain an actual characterization, involved computations for the specific classifier are required. However, we exemplify our methodology in this paper

for the *Naive Bayesian Classifier*. While this process might be tedious, we believe that it leads to a better understanding of learning algorithms.

4. Moment Analysis

Probability distributions completely characterize the behavior of a random variable. Moments of a random variable give us information about its probability distribution. Thus, if we have knowledge of the moments of a random variable we can make statements about its behavior. In some cases characterizing a finite subset of moments may prove to be a more desired alternative than characterizing the entire distribution which can be wild and computationally expensive to compute. This is precisely what we do when we study the behavior of the generalization error of a classifier and the error estimation methods viz. Hold-out-error, Leave-one-out error and Cross-validation error. Characterizing the distribution though possible, can turn out to be a tedious task and studying the moments instead is a more viable option. As a result, we employ moment analysis and use linearity of expectation to explore the relationship between various estimates for the error of classifiers: generalization error(GE), hold-out-set error(HE), and cross validation error(CE) — leave-one-out error is just a particular case of CE and we do not analyze it independently. The relationships are drawn by going over the space of all possible datasets. The actual computation of moments though is conducted by going over the space of classifiers induced by a particular classification algorithm and i.i.d. data as can be seen in Section 6. This is done since it leads to computational efficiency. We interchangably go over the space of datasets and space of classifiers as deemed appropriate, since the classification algorithm is assumed to be deterministic. That is we have,

$$E_{D(N)}[\mathcal{F}(\zeta[D(N)])] = E_{\mathcal{Z}(N)}[\mathcal{F}(\zeta)] = E_{x_1 \times ... \times x_m}[\mathcal{F}(\zeta(x_1, x_2, ..., x_m))]$$

where $\mathcal{F}()$ is some function that operates on a classifier. We also consider the learning algorithms to be symmetric(the algorithm is oblivious to random permutations of the samples in the training dataset).

Throughout this section and in the rest of the paper we use the notation in Table 1 unless stated otherwise.

4.1 Generalization Error

The notion of generalization error is defined with respect to an underlying probability distribution defined over the input output space and a loss function (error metric). We model this probability space with the random vector X for input and random variable Y for output. When the input is fixed, $Y(x)$ is the random variable that models the output[1]. We assume in this paper that the domain $\mathcal{X}$ of X is discrete; all the theory can be extended to continuous essentially by replacing the counting measure with Lebesque measure and sums with integrals. Whenever the probability and expectation is with respect to this probabilistic space(i.e. (X, Y)) that models the problem we will not use any index. For

1. By modeling the output for a given input as a random variable, we allow the output to be randomized, as it might be in most real circumstances.

Symbol	Meaning
X	Random vector modeling input
$\mathcal{X}$	Domain of random vector (input space) X
Y	Random variable modeling output
$Y(x)$	Random variable modeling output for input x
$\mathcal{Y}$	Set of class labels (output space)
D	Dataset
(x, y)	Data-point from dataset D
D_t	Training dataset
D_s	Testing dataset
D_i	The i'th part/fold of D (for cross validation)
N	Size of dataset
N_t	Size of training dataset
N_s	Size of testing dataset
v	Number of folds of cross validation
ζ	Classifier
$\zeta[D]$	Classifier build from dataset D
$GE(\zeta)$	Generalization error of classifier ζ
$HE(\zeta)$	Hold-out-set error of classifier ζ
$CE(\zeta)$	Cross validation error of classifier ζ
$\mathcal{Z}(S)$	The set of classifiers obtained by application of classification algorithm to an i.i.d. set of size S
$D(S)$	Dataset of size S
$E_{\mathcal{Z}(S)}[]$	Expectation w.r.t. the space of classifiers built on a sample of size S

Table 1: Notation used in the paper.

other probabilistic spaces, we will specify by an index what is the probability space we refer to. We denote the error metric by $\lambda(a, b)$; in this paper we will use only the 0-1 metric that takes value 1 if $a \neq b$ and 0 otherwise. With this, the generalization error of a classifier ζ is:

$$\begin{aligned} GE(\zeta) &= E\left[\lambda(\zeta(X), Y)\right] \\ &= P\left[\zeta(X) \neq Y\right] \\ &= \sum_{x \in \mathcal{X}} P\left[X = x\right] P\left[\zeta(x) \neq Y(x)\right] \end{aligned} \tag{1}$$

where we used the fact that, for 0-1 loss function the expectation is the probability that the prediction is erroneous. Notice that the notation using $Y(x)$ is really a conditional on $X = x$. We use this notation since it is intuitive and more compact. The last equation for the generalization error is the most useful in this paper since it decomposes a global measure, generalization error, defined over the entire space into micro measures, one for each input.

By carefully selecting the class of classifiers for which the moment analysis of the generalization error is performed, meaningful and relevant probabilistic statements can be made about the generalization error of a particular classifier from this class. The probability distribution over the classifiers will be based on the randomness of the data used to produce the classifier. To formalize this, let $\mathcal{Z}(N)$ be the class of classifiers built over a dataset of size N with a probability space defined over it. With this, the k-th moment around 0 of the generalization error is:

$$E_{\mathcal{Z}(N)}\left[GE(\zeta)^k\right] = \sum_{\zeta \in \mathcal{Z}(N)} P_{\mathcal{Z}}\left[\zeta\right] GE(\zeta)^k$$

The problem with this definition is that it talks about global characterization of classifiers which can be hard to capture. We rewrite the formulae for the first and second moment in terms of fine granularity structure of the classifiers.

While deriving these moments, we have to consider double expectations of the form: $E_{\mathcal{Z}(N)}\left[E\left[\mathcal{F}(x, \zeta)\right]\right]$ with $\mathcal{F}(x, \zeta)$ a function that depends both on the input x and the classifier. With this we arrive at the following result:

$$\begin{aligned} E_{\mathcal{Z}(N)}\left[E\left[\mathcal{F}(x, \zeta)\right]\right] &= \sum_{\zeta \in \mathcal{Z}(N)} P_{\mathcal{Z}(N)}\left[\zeta\right] \sum_{x \in \mathcal{X}} P\left[X = x\right] \mathcal{F}(x, \zeta) \\ &= \sum_{x \in \mathcal{X}} P\left[X = x\right] \sum_{\zeta \in \mathcal{Z}(N)} P_{\mathcal{Z}(N)}\left[\zeta\right] \mathcal{F}(x, \zeta) \\ &= E\left[E_{\mathcal{Z}(N)}\left[\mathcal{F}(x, \zeta)\right]\right] \end{aligned} \tag{2}$$

that uses the fact that $P\left[X = x\right]$ does not depend on a particular ζ and $P_{\mathcal{Z}(N)}\left[\zeta\right]$ does not depend on a particular x, even though both quantities depend on the underlying probability distribution.

Using the definition of the moments above, Equation 1 and Equation 2 we have the following theorem.

Theorem 1 *The first and second moment of GE are given by,*

$$E_{\mathcal{Z}(N)}\left[GE(\zeta)\right] = \sum_{x \in \mathcal{X}} P\left[X=x\right] \sum_{y \in \mathcal{Y}} P_{\mathcal{Z}(N)}\left[\zeta(x)=y\right] P\left[Y(x) \neq y\right]$$

and

$$\begin{aligned} E_{\mathcal{Z}(N)\times\mathcal{Z}(N)}\left[GE(\zeta)GE(\zeta')\right] = & \sum_{x \in \mathcal{X}} \sum_{x' \in \mathcal{X}} P\left[X=x\right] P\left[X=x'\right] \cdot \\ & \sum_{y \in \mathcal{Y}} \sum_{y' \in \mathcal{Y}} P_{\mathcal{Z}(N)\times\mathcal{Z}(N)}\left[\zeta(x)=y \wedge \zeta'(x')=y'\right] \cdot \\ & P\left[Y(x) \neq y\right] P\left[Y(x') \neq y'\right] \end{aligned}$$

Proof

$$\begin{aligned} E_{\mathcal{Z}(N)}\left[GE(\zeta)\right] &= E_{\mathcal{Z}(N)}\left[E\left[\lambda(\zeta(X), Y)\right]\right] \\ &= E\left[E_{\mathcal{Z}(N)}\left[\lambda(\zeta(X), Y)\right]\right] \\ &= \sum_{x \in \mathcal{X}} P\left[X=x\right] \sum_{\zeta \in \mathcal{Z}(N)} P_{\mathcal{Z}(N)}\left[\zeta\right] P\left[\zeta(x) \neq Y(x) | \zeta\right] \\ &= \sum_{x \in \mathcal{X}} P\left[X=x\right] \sum_{\zeta \in \mathcal{Z}(N)} P_{\mathcal{Z}(N)}\left[\zeta\right] P\left[\zeta(x) = y, Y(x) \neq y | \zeta\right] \\ &= \sum_{x \in \mathcal{X}} P\left[X=x\right] \sum_{y \in \mathcal{Y}} \sum_{\zeta \in \mathcal{Z}(N) | \zeta(x)=y} P_{\mathcal{Z}(N)}\left[\zeta\right] P\left[\zeta(x) = y, Y(x) \neq y | \zeta\right] \\ &= \sum_{x \in \mathcal{X}} P\left[X=x\right] \sum_{y \in \mathcal{Y}} P_{\mathcal{Z}(N)}\left[\zeta(x)=y\right] P\left[Y(x) \neq y\right] \end{aligned}$$

$$\begin{aligned} & E_{\mathcal{Z}(N)\times\mathcal{Z}(N)}\left[GE(\zeta)GE(\zeta')\right] \\ &= E_{\mathcal{Z}(N)\times\mathcal{Z}(N)}\left[E\left[\lambda(\zeta(X), Y)\right] E\left[\lambda(\zeta'(X), Y)\right]\right] \\ &= \sum_{(\zeta,\zeta') \in \mathcal{Z}(N)\times\mathcal{Z}(N)} P_{\mathcal{Z}(N)\times\mathcal{Z}(N)}\left[\zeta, \zeta'\right] \left(\sum_{x \in \mathcal{X}} P\left[X=x\right] P\left[\zeta(x) \neq Y(x)\right]\right) \\ & \qquad \left(\sum_{x \in \mathcal{X}} P\left[X=x\right] P\left[\zeta'(x) \neq Y(x)\right]\right) \\ &= \sum_{x \in \mathcal{X}} \sum_{x' \in \mathcal{X}} P\left[X=x\right] P\left[X=x'\right] \sum_{(\zeta,\zeta') \in \mathcal{Z}(N)\times\mathcal{Z}(N)} P_{\mathcal{Z}(N)\times\mathcal{Z}(N)}\left[\zeta, \zeta'\right] \\ & \qquad P\left[\zeta(x) \neq Y(x)\right] P\left[\zeta'(x') \neq Y(x')\right] \\ &= \sum_{x \in \mathcal{X}} \sum_{x' \in \mathcal{X}} P\left[X=x\right] P\left[X=x'\right] \\ & \qquad \sum_{y \in \mathcal{Y}} \sum_{y' \in \mathcal{Y}} P_{\mathcal{Z}(N)\times\mathcal{Z}(N)}\left[\zeta(x)=y \wedge \zeta'(x')=y'\right] \\ & \qquad\qquad P\left[Y(x) \neq y\right] P\left[Y(x') \neq y'\right] \end{aligned}$$

■

In both series of equations we made the transition from a summation over the class of classifiers to a summation over the possible outputs since the focus changed from the classifier to the prediction of the classifier for a specific input(x is fixed inside the first summation). What this effectively does is it allows the computation of moments using only local information(behavior on particular inputs) not global information(behavior on all inputs). This results in speeding the process of computing the moments as can be seen in Section 6.

4.2 Alternative Methods for computing the moments of GE

The method we introduced above for computing the moments of the generalization error are based on decomposing the moment into contributions of individual input-output pairs. With such a decomposition, not only the analysis becomes simpler, but the complexity of the algorithm required is reduced. In particular, the complexity of computing the first moment is proportional to the size of the input-output space and the complexity of estimating probabilities of the form $P_{\mathcal{Z}}[\zeta(x)=y]$. The complexity of the second moment is quadratic in the size of the input-output space and proportional to the complexity of estimating $P_{\mathcal{Z}}[\zeta(x)=y \wedge \zeta(x')=y']$. To see the advantage of this method, we compare it with the other two alternatives for computing the moments: definition based computation and Monte Carlo simulation.

Definition based computation uses the definition of expectation. It consists in summing over all possible datasets and multiplying the generalization error of the classifier built from the dataset with the probability to obtain the dataset as an i.i.d. sample from the underlying probability distribution. Formally,

$$E_{\mathcal{D}(N)}[GE(\zeta)] = \sum_{D \in \mathcal{D}(N)} P[D]\, GE(\zeta[D]) \tag{3}$$

where $\mathcal{D}(N)$ is the set of all possible datasets of size N. The number of possible datasets is exponential in N with the base of the exponent proportional to the size of the input-output space (the product of the sizes of the domains of inputs and outputs). Evaluating the moments in this manner is impractical for all but very small spaces and dataset sizes.

Monte Carlo simulation is a simple way to estimate moments that consists in performing experiments to produce *samples* that determine the value of the generalization error. In this case, to estimate $E_{\mathcal{D}(N)}[GE(\zeta)]$, datasets of size N have to be generated, one for each sample desired. For each of these datasets a classifier has to be constructed according to the classifier construction algorithm. For the classifier produced, samples from the underlying probability distribution have to be generated in order to estimate the generalization error of this classifier. Especially for second moments, the amount of samples required will be large in order to obtain reasonable accuracy for the moments. If a study has to be conducted in order to determine the influence of various parameters of the data generation model, the overall amount of experiments that have to performed becomes infeasible.

In summary, the advantages of the method we propose for estimating the moments are: (a) it is exact, (b) it needs only local behavior of the classifier, (c) so the time complexity

is reduced and (d) does not depend on the fact that some of the probabilities are small. We will use this method to compute moments of the generalization error for the NBC in Section 6.

4.3 Error Estimation Methods

The exact computation of the generalization error depends on the actual underlying probability distribution, which is unknown, and hence other estimates for the generalization error have been introduced: hold out set(HOS), leave-one-out(LOO), and v-fold cross validation(CV). In the this subsection we establish relationships between moments of these error metrics and the moments of the generalization error with respect to some distribution over the classifiers. The general setup for the analysis for all these metrics is the following. A dataset D of size N is provided, containing i.i.d. samples coming from the underlying distribution over the input and outputs. The set is further divided and used both to build a classifier and to estimate the generalization error; the particular way this is achieved is slightly different for each error metric. The important question we will ask is how the values of the error metrics relate to the generalization error. In all the developments that follow we will assume that $\zeta[D]$ is the classifier built deterministically from the dataset D.

4.3.1 Hold out Set(HOS) Error

The HOS error involves randomly partitioning the dataset D into two parts D_t, the training dataset of fixed size N_t, and D_s, the test dataset of fixed size N_s. A classifier is built over the training dataset and the generalization error is estimated as the average error over the test dataset. Formally, denoting the random variable that gives the HOS error by HE we have:

$$HE = \frac{1}{N_s} \sum_{(x,y) \in D_s} \lambda(\zeta[D_t](x), y) \tag{4}$$

where y is the actual label for the input x.

Proposition 1 *The expected value of HE is given by,*

$$E_{D_t(N_t) \times D_s(N_s)}[HE] = E_{D_t(N_t)}[GE(\zeta[D_t])]$$

Proof Using the notation in Table 1 and realizing that all the datapoints are i.i.d. we derive the above result.

$$\begin{aligned}
E_{D_t(N_t) \times D_s(N_s)}[HE] &= E_{D_t(N_t)}\left[E_{D_s(N_s)}\left[\frac{\sum_{(x,y) \in D_s} \lambda(\zeta[D_t](x), y)}{N_s}\right]\right] \\
&= E_{D_t(N_t)}\left[E_{D_s(N_s)}\left[P[\zeta[D_t](x) \neq y | D_s]\right]\right] \\
&= E_{D_t(N_t)}\left[\sum_{D_s} P[\zeta[D_t](x) \neq y | D_s] P[D_s]\right] \\
&= E_{D_t(N_t)}\left[\sum_{D_s} P[\zeta[D_t](x) \neq y, D_s]\right] \\
&= E_{D_t(N_t)}\left[P[\zeta[D_t](x) \neq y]\right] \\
&= E_{D_t(N_t)}\left[GE(\zeta[D_t])\right]
\end{aligned}$$

where we used the fact that by going over all values of one r.v. we get the probability fo the other. ■

We observe from the above result that the expected value of HE is dependent only on the size of the training set D_t. This result is intuitive since only N_t data-points are used for building the classifier.

Lemma 1 *The second moment of HE is given by,*
$E_{D_t(N_t)\times D_s(N_s)}\left[HE^2\right] = \frac{1}{N_s}E_{D_t(N_t)}\left[GE(\zeta[D_t])\right] + \frac{N_s-1}{N_s}E_{D_t(N_t)}\left[GE(\zeta[D_t])^2\right]$.

Proof To compute the second moment of HE, from the definition in Equation 4 from the paper we have:

$$E_{D_t(N_t)\times D_s(N_s)}\left[HE^2\right] =$$
$$\frac{1}{N_s^2}E_{D_t(N_t)\times D_s(N_s)}\left[\sum_{(x,y)\in D_s}\sum_{(x',y')\in D_s}\lambda(\zeta[D_t](x),y)\lambda(\zeta[D_t](x'),y')\right]$$

The expression under the double sum depends on whether (x,y) and (x',y') are the same or not. When they are the same, we are precisely in the case we derived for $E_{D_t(N_t)\times D_s(N_s)}\left[HE\right]$ above, except that we have N_s^2 in the denominator. This gives us the following term, $\frac{1}{N_s}E_{D_t(N_t)}\left[GE(\zeta[D_t])\right]$. When they are different, i.e. $(x,y)\neq(x',y')$ then we get,

$$\begin{aligned}
&\frac{1}{N_s^2}E_{D_t(N_t)\times D_s(N_s)}\left[\sum_{(x,y)\in D_s}\sum_{(x',y')\in D_s\setminus(x,y)}\lambda(\zeta[D_t](x),y)\lambda(\zeta[D_t](x'),y')\right]\\
&= \frac{N_s-1}{N_s}E_{D_t(N_t)\times D_s(N_s)}\left[\frac{\sum_{(x,y)\in D_s}\lambda(\zeta[D_t](x),y)}{N_s}\frac{\sum_{(x',y')\in D_s\setminus(x,y)}\lambda(\zeta[D_t](x'),y')}{N_s-1}\right]\\
&= \frac{N_s-1}{N_s}E_{D_t(N_t)\times D_s(N_s)}\left[P[\zeta[D_t](x)\neq y|(x,y)\in D_s]P[\zeta[D_t](x')\neq y'|(x',y')\in D_s\setminus(x,y)]\right]\\
&= \frac{N_s-1}{N_s}E_{D_t(N_t)}\left[E_{D_s}\left[P[\zeta[D_t](x)\neq y|(x,y)\in D_s]\right]E_{D_s}\left[P[\zeta[D_t](x')\neq y'|(x',y')\in D_s\setminus(x,y)]\right]\right]\\
&= \frac{N_s-1}{N_s}E_{D_t(N_t)}\left[GE(\zeta[D_t])^2\right]
\end{aligned}$$

where we used the primary fact that since the samples are i.i.d. any function applied on two distinct inputs is also independent. This the reason why the $E_{D_s}[]$ factorizes.

Putting everything together, and observing that terms inside summations are constants, we have:

$$E_{D_t(N_t)\times D_s(N_s)}\left[HE^2\right] = \frac{1}{N_s}E_{D_t(N_t)}\left[GE(\zeta[D_t])\right] + \frac{N_s-1}{N_s}E_{D_t(N_t)}\left[GE(\zeta[D_t])^2\right]$$

■

Theorem 2 *The variance of HE is given by,*
$Var_{D_t(N_t)\times D_s(N_s)}(HE) = \frac{1}{N_s}E_{D_t(N_t)}\left[GE(\zeta[D_t])\right] + \frac{N_s-1}{N_s}E_{D_t(N_t)}\left[GE(\zeta[D_t])^2\right] - E_{D(N_t)}\left[GE(\zeta[D_t])\right]^2$.

Proof The proof of the theorem immediately follows from the Proposition 1 and Lemma 1 and by using the formula for the variance of a random variable in this case HE, $Var(HE) = E[HE^2] - E[HE]^2$. ■

Unlike the first moment the variance depends on the sizes of both, the training set as well as the test set.

4.3.2 Multifold Cross Validation Error

v-fold cross validation consists in randomly partitioning the available data into v equi-size parts and then training v classifiers using all data but one chunk and then testing the performance of the classifier on this chunk. The estimate of the generalization error of the classifier build from the entire data is the average error over the chunks. Using notation in this paper and denoting by D_i the i-th chunk of the data set D, the Cross Validation Error (CE) is:

$$CE = \frac{1}{v}\sum_{i=1}^{v} HE_i$$

Notice that we expressed CE in terms of HE, the HOS Error. By substituting the formula for HE in Equation 4 into the above equation, a direct definition for CV is obtained, if desired.

In this case we have a classifier for each chunk not a single classifier for the entire data. We model the selection of N i.i.d. samples that constitute the dataset D and the partitioning into v chunks. With this we have:

Proposition 2 *The expected value of CE is given by,*

$$E_{D_t\left(\frac{v-1}{v}N\right)\times D_i\left(\frac{N}{v}\right)}[CE] = E_{D_t\left(\frac{v-1}{v}N\right)}[GE(\zeta[D_t])]$$

Proof Using the above equation, Proposition 1 we get the above result.

$$\begin{aligned} E_{D_t\left(\frac{v-1}{v}N\right)\times D_i\left(\frac{N}{v}\right)}[CE] &= \frac{1}{v}\sum_{i=1}^{v} E_{D_t\left(\frac{v-1}{v}N\right)\times D_i\left(\frac{N}{v}\right)}[HE_i] \\ &= \frac{1}{v}\sum_{i=1}^{v} E_{D_t\left(\frac{v-1}{v}N\right)}[GE(\zeta[D_t])] \\ &= E_{D_t\left(\frac{v-1}{v}N\right)}[GE(\zeta[D_t])] \end{aligned}$$

■

This results follows the intuition since is states that the expected error is the generalization error of a classifier trained on $\frac{v-1}{v}N$ data-points. Thus, at least on expectation, the cross validation behaves exactly like HOS estimate that is trained over $v-1$ chunks.

For CE we could compute the second moment around zero using the strategy previously shown and then compute the variance. Here, we compute the variance using the relationship between the variance of the sum and the variances and covariances of individual terms.

In this way we can decompose the overall variance of CE into the sum of variances of individual estimators and the sum of covariances of pairs of such estimators; this decomposition significantly enhances the understanding of the behavior of CE, as we will see in the example in Section 6.

$$Var(CE) = \frac{1}{v^2}\left(\sum_{i=1}^{v} Var(HE_i) + \sum_{i\neq j} Cov(HE_i, HE_j)\right) \tag{5}$$

The quantity $Var(HE_i)$, the variance of the HE on training data of size $\frac{v-1}{v}N$ and test data of size $\frac{1}{v}N$, is computed using the formulae in the previous section. The only things that remain to be computed are the covariances. Since we already computed the expectation of HE, to compute the covariance it is enough to compute the quantity $\mathcal{Q} = E[HE_iHE_j]$ (since for any two random variables X, Y we have $Cov(X,Y) = E[XY] - E[X]E[Y]$). Let D^i denote $D \setminus D_i$ and let $N_s = \frac{N}{v}$. With this we have the following lemma,

Lemma 2 *The* $E_{D_t^i(\frac{v-1}{v}N)\times D_i(\frac{N}{v})\times D_t^j(\frac{v-1}{v}N)\times D_j(\frac{N}{v})}[HE_iHE_j] =$
$E_{D_t^i(\frac{v-1}{v}N)\times D_t^j(\frac{v-1}{v}N)}[GE(\zeta[D\setminus D_i])GE(\zeta[D\setminus D_j])]$.

Proof

$$\begin{aligned}
&E_{D_t^i(\frac{v-1}{v}N)\times D_i(\frac{N}{v})\times D_t^j(\frac{v-1}{v}N)\times D_j(\frac{N}{v})}[HE_iHE_j] \\
&= E_{D_t^i(\frac{v-1}{v}N)\times D_t^j(\frac{v-1}{v}N)\times D_i(\frac{N}{v})}\left[HE_i E_{D_j(\frac{N}{v})}\left[\frac{\sum_{(x_j,y_j)\in D_j}\lambda(\zeta[D_t^j](x_j), y_j)}{N_s}\right]\right] \\
&= E_{D_t^i(\frac{v-1}{v}N)\times D_t^j(\frac{v-1}{v}N)}\left[GE(\zeta[D\setminus D_j])E_{D_i(\frac{N}{v})}[HE_i]\right] \\
&= E_{D_t^i(\frac{v-1}{v}N)\times D_t^j(\frac{v-1}{v}N)}[GE(\zeta[D\setminus D_j])GE(\zeta[D\setminus D_i])]
\end{aligned}$$

where used the fact that the datasets D_i and D_j are disjoint and drawn i.i.d. It is important to observe that due to the fact that $D \setminus D_i$ and $D \setminus D_j$ intersect (the intersection is $D \setminus (D_i \cup D_j)$), the two classifiers will neither be independent nor identical. As was the case for the first and second moment of GE, this moment will depend only on the size of the intersection and the sizes of the two sets since all points are i.i.d. This means that the expression has the same value for any pair $i, j, i \neq j$. ■

Theorem 3 *The variance of* CE *is given by,*

$$\begin{aligned}
Var_{D_t^i(\frac{v-1}{v}N)\times D_i(\frac{N}{v})\times D_t^j(\frac{v-1}{v}N)\times D_j(\frac{N}{v})}CE = &\tfrac{1}{N}E_{D_t(\frac{v-1}{v}N)}[GE(\zeta[D_t])] + \\
&\tfrac{N-v}{vN}E_{D_t(\frac{v-1}{v}N)}[GE(\zeta[D_t])^2] - \tfrac{v-2}{v}E_{D_t(\frac{v-1}{v}N)}[GE(\zeta[D_t])]^2 + \\
&\tfrac{v-1}{v}E_{D_t^i(\frac{v-1}{v}N)\times D_t^j(\frac{v-1}{v}N)}[GE(\zeta[D\setminus D_j])GE(\zeta[D\setminus D_i])]
\end{aligned}$$

Proof The expression for the covariance is immediate from the above result and so using Equation 5 we derive the variance of CE. ■

It is worth mentioning that leave-one-out(LOO) is just a special case of v-fold cross validation ($v = N$ for leave-one-out). The formulae above apply to LOO as well thus no separate analysis is necessary.

With this we have related the first two moments of HE and CE to that of GE. In the next section we shall leverage this fact and utilise it to study the model selection measures.

5. Methodology

In the previous section we derived relationships between the moments of hold-out-set error, cross validation error, leave-one-out error and the moments of the generalization error. Thus, if we know the moments of GE we can compute the moments of the other error measures and vice-versa. Notice from the formulations that to compute these moments we need to know the underlying distribution and the classification algorithm. We now discuss these two requirements in detail.

5.1 Data Generation Model

The generalization error depends both on the classifier and the joint underlying probability distribution of the predictor attributes and the class label. The joint probability distribution of the inputs and output is completely specified by the parameters:

$$p_{i,k} = P\left[X = \bar{x_i} \wedge Y = y_k\right] \tag{6}$$

the probability to see each of the combinations of values for the inputs and outputs. Here, $p_{i,k}$ is the probability of being in class y_k for the input vector $\bar{x_i}$. This is indeed the most general data generation model for the discrete case when only the random vector X is considered as input since any extra information would be irrelevant from the point of view of classification process. Under this data generation model, the distribution of the random variables N_{ik} is multinomial with the total number of items N and the probabilities that determine the multinomial, $p_{i,k}$. In other words, we can model our generic data generation process by a multinomial distribution. The term data generation is somewhat of a misnomer here, since it is important to note that in all the simulations we report there is *no explicit data generation*. The probabilistic formulations that we soon propose are used for the simulations. Thus, the formulations act as parametric equations whose parameters we vary to produce the plots. This aspect is critical since it means that just by altering the values of the parameters(individual cell probabilities of the multinomial, dataset size, etc.) we can observe the behavior of the errors under various circumstances in a short amount of time. Moreover, it gives us control over what we wish to observe.

5.2 Classification Algorithm

The results(i.e. expressions and relationships) we derived in the previous section were generic applicable to any deterministic classification algorithm. We can thus use those results to study the behavior of the errors for a classification algorithm of our choice.

Table 2: Contingency table of input X

X	y_1	y_2	
x_1	N_{11}	N_{12}	
x_2	N_{21}	N_{22}	
$\vdots$			
x_n	N_{n1}	N_{n2}	
	N_1	N_2	N

The classification algorithm we consider in this paper is Naive Bayes. We first study the Naive bayes for a single input attribute(i.e. for one dimension) and later the generalized version maintaining scalability.

5.3 Roadmap

With this, we deploy the proposed methodology in the remaining part of the paper which involves;

1. Deriving expressions for the first two moments of GE for the NBC which in turn also provide expressions for the first two moments of HE and CE(LOO just a special case of cross-validation).

2. Using the expressions in conjunction with the data generation model as an exploratory tool to observe the behavior of the errors under different circumstances leading to interesting explanations for certain observed facts. Realize that the goal is not to make generalized statements but to use the method to observe specifics. However, we still discuss the qualitative extensibility of certain results.

6. Example: Naive Bayes Classifier

In this section we will compute explicitly these moments for the NBC with a single input. As we will see, these moments are too complicated, as mathematical formulae, to interpret. We will plot these moments to gain an understanding of the behavior of the errors under different conditions thus portraying the usefulness of the proposed method.

6.1 Naive Bayes Classifier Model

In order to compute the moments of the generalization error, moments that we linked in Section 4 with the moments of the hold-out set error and cross validation, we first have to select a classifier and specify the construction method. We selected the single input, naive Bayes classifier since the analysis is not too complicated but highlights both the method and the difficulties that have to be overcome. As we will see, even this simplified version exhibits interesting behavior. We fix the number of class labels to 2 as well. In the next section, we discuss how the analysis we present here extends to the general NBC.

Given values for any of the inputs, the NBC computes the probability to see any of the class labels as output under the assumption that the inputs influence the output independently. The prediction is the class label that has the largest such estimated probability. For the version of the naive Bayes classifier we consider here(i.e. a single input), the prediction given input x is:

$$\zeta(x) = \text{argmax}_{k\in\{1,2\}} P\left[Y = y_k\right] P\left[X = x | Y = y_k\right]$$

The probabilities that appear in the formula are estimated using the counts in the contingency table in Table 2. Using the fact that $P\left[Y = y_k\right] P\left[X = x | Y = y_k\right] = P\left[X = x \wedge Y = y_k\right]$ and the fact that $P\left[X = x_i \wedge Y = y_k\right]$ is $\frac{N_{ik}}{N}$, the prediction of the classifier is:

$$\zeta(x_i) = \begin{cases} y_1 & \text{if } N_{i1} \geq N_{i2}) \\ y_2 & \text{if } N_{i1} < N_{i2}) \end{cases}$$

6.2 Computation of the Moments of GE

Under the already stated data generation model, the moments of the generalization error for the NBC can be computed. We now present three approaches for computing the moments and show that the approach using theorem 1 is by far the most practical.

Going over datasets: If we calculate the moments by going over all possible datasets the number of terms in the formulation for the moments is exponential in the number of attribute values with the base of the exponential being the size of the dataset(i.e. $O(N^n)$ terms). This is because each of the cells in Table 2 can take $O(N)$ values. The formulation of the first moment would be as follows.

$$E_{D(N)}[GE(\zeta(D(N)))] = \sum_{N_{11}=0}^{N} \sum_{N_{12}=0}^{N-N_{11}} \ldots \sum_{N_{n1}=0}^{N-(N_{11}+\ldots+N_{(n-1)2})} eP[N_{11}, \ldots, N_{n2}] \qquad (7)$$

where e is the corresponding error of the classifier. We see that this formulation can be tedious to deal with. So can we do better? Yes we definitely can and this spurs from the following observation. For the NBC built on Table 2 all we care about in the classification process is the relative counts in each of the rows. Thus, if we had to classify a datapoint with attribute value x_i we would classify it into class y_1 if $N_{i1} > N_{i2}$ and vice-versa. What this means is that irrespective of the actual counts of N_{i1} and N_{i2} as long $N_{i1} > N_{i2}$ the classification algorithm would make the same prediction i.e. we would have the same classifier. We can hence switch from going over the space of all possible datasets to going over the space of all possible classifiers with the advantage of reducing the number of terms.

Going over classifiers: If we find the moments by going over the space of possible classifiers we reduce the number of terms from $O(N^n)$ to $O(2^n)$. This is because there are only two possible relations between the counts in any row($\geq$ or $<$). The formulation for the first moment would then be as follows.

$$\begin{aligned} E_{Z(N)}[GE(\zeta)] \quad = \quad & e_1 P[N_{11} \geq N_{12}, \ldots, N_{n1} \geq N_{n2}] + e_2 P[N_{11} < N_{12}, \ldots, N_{n1} \geq N_{n2}] + \ldots \\ & + e_{2^n} P[N_{11} < N_{12}, \ldots, N_{n1} < N_{n2}] \end{aligned}$$

where e_1, e_2 to e_{2^n} are the corresponding errors. Though this formulation reduces the complexity significantly since $N >> 2$ for any practical scenario, nonetheless the number of terms are still exponential in n. Can we still do better? The answer is yes again. Here is where theorem 1 gains prominence. To restate, theorem 1 says that while calculating the first moment we just need to look at particular rows of the table and to calculate the second moment just pairs of rows. This reduces the complexity significantly without compromising on the accuracy as we will see.

Going over classifiers using Theorem 1: If we use theorem 1 the number of terms reduces from an exponential in n to small polynomial in n. Thus the number of terms in finding the first moment is just $O(n)$ and that for the second moment is $O(n^2)$. The formulation for the first moment would then be as follows.

$$E_{Z(N)}[GE(\zeta)] = e_1 P[N_{11} \geq N_{12}] + e_2 P[N_{11} < N_{12}] + ... + e_{2n} P[N_{n1} < N_{n2}]$$

where e_1, e_2 to e_{2n} are the corresponding errors. They are basically the respective cell probabilities of the multinomial(i.e. e_1 is probability of a datapoint belonging to the cell x_1C_2, e_2 is probability to belong to x_1C_1 and so on). For the second moment we would have joint probabilities with the expression being the following,

$$\begin{aligned} E_{Z(N)}[GE(\zeta)^2] &= (e_1 + e_3) P[N_{11} \geq N_{12}, N_{21} \geq N_{22}] + (e_2 + e_3) P[N_{11} < N_{12}, N_{21} \geq N_{22}] + ... \\ &\quad + (e_{2n-2} + e_{2n}) P[N_{(n-1)1} < N_{(n-1)2}, N_{n1} < N_{n2}] \end{aligned}$$

We have thus reduced the number of terms from $O(N^n)$ to $O(n^k)$ where k is small and depends on the order of the moment we are interested in. This formulation has another advantage. The complexity of calculating the individual probabilities is also significantly reduced. The probabilities for the first moment can be computed in $O(N^2)$ time and that for the second in $O(N^4)$ time rather than $O(N^{n-1})$ and $O(N^{2n-2})$ time respectively.

Further optimizations can be done by identifying independence between random variables and expressing them as binomial cdf's and using the incomplete regularized beta function to calculate these cdf's in essentially constant time. Infact in future sections we discuss the general NBC model for which the cdf's(probabilities) cannot be computed directly as it turns out to be too expensive. Over there we propose strategies to efficiently compute these probabilities. The same strategies can be used here to make the computation more scalable.

The situation when ζ and ζ' are the classifiers constructed for two different folds in cross validation requires special treatment. Without loss of generality, assume that the classifiers are built for the folds 1 and 2. If we let $D_1, \ldots, D_v$ be the partitioning of the datasets into v parts, ζ is constructed using $D_2 \cup D_3 \cup \cdots \cup D_v$ and ζ' is constructed using $D_1 \cup D_3 \cup \cdots \cup D_v$, thus $D_3 \cup \cdots \cup D_v$ training data is common for both. If we denote by N_{jk} the number of data-points with $X = x_j$ and $Y = y_k$ in this common part and by $N_{jk}^{(1)}$ and $N_{jk}^{(2)}$ the number of such data-points in D_1 and D_2, respectively, then we have to compute probabilities of the form,

$$P\left[(N_{i1}^{(2)} + N_{i1} > N_{i2}^{(2)} + N_{i2}) \wedge (N_{j1}^{(1)} + N_{j1} > N_{j2}^{(1)} + N_{j2})\right]$$

The estimation of this probability using the above method requires fixing the values of 6 random variables, thus giving an $O(N^6)$ algorithm. Again, further optimizations can be carried out using the strategies in Sections 7 and 8.

Using the moments of GE, the moments of HE and CE are found using relationships already derived.

6.3 Moment Comparison of Test Metrics

In the previous subsections we pointed out how the moments of the generalization error can be computed. In Section 4 we established connections between the moments of the generalization error (GE) and the moments of Hold-out-set error (HE) and Cross validation error (CE). Neither of these relationships, i.e. between the moments of these errors and the moments of the generalization error give a direct indication of the behavior of the different types of errors in specific circumstances. In this section we provide a graphical, simple to interpret, representation of these moments for specific cases. While these visual representations do not replace general facts, they greatly improve our understanding, allowing rediscovery of empirically discovered properties of the error metrics and portraying the flexibility of the method to model different scenarios.

6.3.1 Generalization Error

In our first study we want to establish the prediction behavior of the naive Bayes classifier as a function of the probability to see the first class label, which we call the prior and denote by p, and the amount of correlation between the input and the output. To this end, we consider the situation when the domain of the input variable consists of x_1, x_2, the size of the dataset $N = 1000$, and the underlying probability distribution given by the following prior probabilities:

$$
\begin{aligned}
P\left[X = x_1 \wedge y = y_1\right] &= \frac{p}{2}(1-c) \\
P\left[X = x_2 \wedge y = y_1\right] &= \frac{p}{2}(1+c) \\
P\left[X = x_1 \wedge y = y_2\right] &= \frac{1-p}{2}(1+c) \\
P\left[X = x_2 \wedge y = y_2\right] &= \frac{1-p}{2}(1-c)
\end{aligned}
$$

where c is the correlation coefficient which we vary between 0 and 1 (the behavior is symmetric when c is varied between -1 and 0). The behavior of the expectation of the generalization error is depicted in Figure 1. On the most part, the behavior is expected and reaffirms the fact that the naive Bayes classifier is optimal (it has been shown to be optimal Domingos and Pazzani (1997) if the independence between inputs holds; this is the case here since there is only one input). The bumps in the graphs (they appear more clear at a higher resolution) are due to the way the ties are broken, i.e. when the counts are equal, the prediction of each of the class labels is with probability 1/2. They appear only in the region of the graph in which the two counts compete; should the ties have been broken with probabilities p and $1-p$, the graph would have been smooth. The ability to make such

predictions shows the power of the moment analysis method we introduced here; it was much easier to observe such behavior visually than from formulae.

6.3.2 Hold-out-set Error

Our second study involves the dependency of the hold-out-set error on the splitting of the data into testing (the hold-out-set) and training. We consider datasets of size $N = 100$ and we set the underlying probability distribution such that the output is independent of the input and the prior probability of the first class label is $p = 0.4$. To get insight into the behavior of HE, we plotted the expectation in Figure 2, the variance in Figure 3 and the sum of the expectation and one standard deviation in Figure 4. As expected, the expectation of HE grows as the size of the training dataset reduces. On the other hand, the variance is reduced until the size of test data is 50% than it increases slightly. The general downwards trend is predictable using intuitive understanding of the naive Bayes classifier but the fact that the variance has an upwards trend is not. We believe that the behavior on the second part of the graph is due to the fact that the behavior of the classifier becomes unstable as the size of the training dataset is reduced and this competes with the reduction due the increase in the size of testing data. Our methodology established this fact exactly without the doubts associated with intuitively determining which of the two factors is dominant.

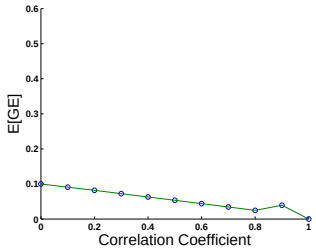
From the plot for the sum of the expectation and the standard deviation of HE in Figure 4, which indicates the pessimistic expected behavior, a good choice for the size of the test set is 40-50% for this particular instance. This best split depends on the size of the dataset and it is hard to select only based on intuition.

6.3.3 Cross Validation Error

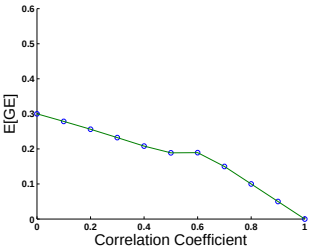
In our third study we used the same setup described above, for cross validation and depicted the results in Figures 5-9. As the number of folds increases, the following trends are observed: (a) the expectation of CE reduces (Figure 5) since the size of training data increases, (b) the variance of the classifier for each of the folds increases (Figure 6) since the size of the test data decreases, (c) the covariance between the estimates of different folds decreases first then increases again (Figure 7) – we explain this behavior below – and the same trend is observed for the total variance of CE (Figure 8) and the sum of the expectation and the standard deviation of CE (Figure 9). Observe that the minimum of the sum of the expectation and the standard deviation (which indicates the pessimistic expected behavior) is around 10-20 folds, which coincides with the number of folds usually recommended.

A possible explanation for the behavior of the covariance between the estimates of different folds is based on the following two observations. First, when the number of folds is small, the errors of the estimates have large correlations despite the fact that the classifiers are negatively correlated – this happens because almost the entire training dataset of one classifier is the test set for the other, 2-fold cross validation in the extreme. Due to this, though the classifiers built may be similar or different their errors are strongly positively correlated. Second, for large number of folds (leave-one-out situation in the extreme), there is a huge overlap between the training sets, thus the classifiers built are almost the same and so the corresponding errors they make are highly correlated again. These two opposing

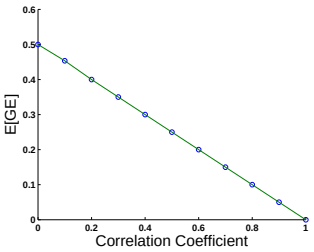
2. Plateau at 0.4 is $E_{\mathcal{Z}}[GE(\zeta)]$.



(a) Prior=0.1



(b) Prior=0.3



(c) Prior=0.5

Figure 1: Dependency of expectation of generalization error on correlation factor for various priors.

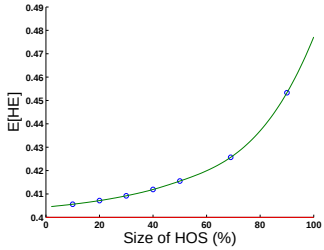


Figure 2: Hold-out-set expectation

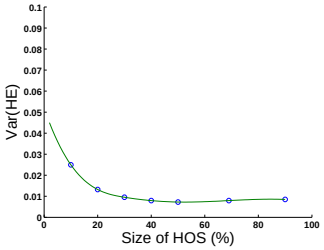


Figure 3: Hold-out-set variance

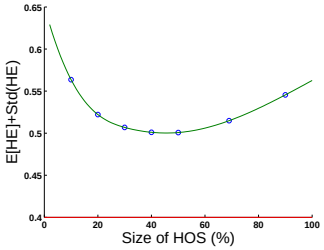


Figure 4: $E[] + \sqrt{\mathrm{Var}()}$ for hold-out-set

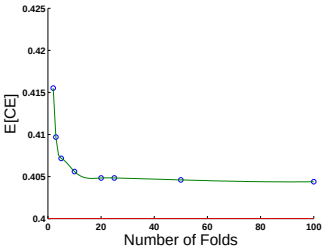


Figure 5: Expectation of CE.

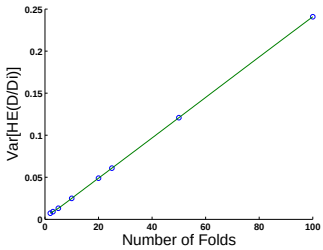


Figure 6: Individual run variance of CE.

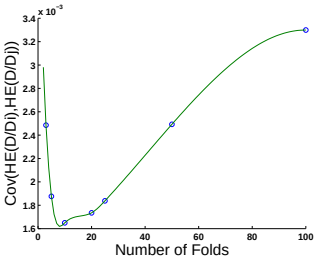


Figure 7: Pairwise covariances of CV.

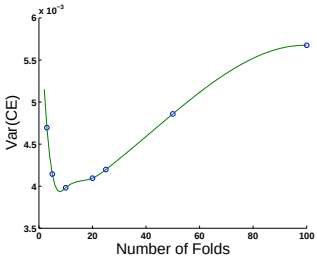


Figure 8: Total variance of cross validation.

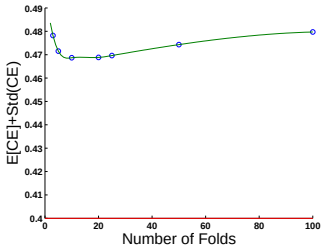


Figure 9: $E[] + \sqrt{\mathrm{Var}(())}$ of CV

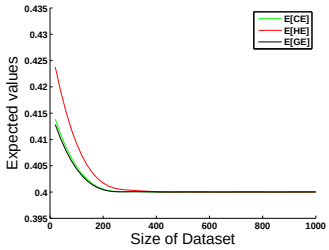


Figure 10: Convergence behavior.

Symbol	Symantics
p_{c1}	prior of class C_1
p_{c2}	prior of class C_2
p^x_{11}	joint probability of being in $x_1,\ C_1$
p^x_{12}	joint probability of being in $x_1,\ C_2$
p^y_{11}	joint probability of being in $y_1,\ C_1$
p^y_{12}	joint probability of being in $y_1,\ C_2$
N_1	r.v. denoting number of datapoints in class C_1
N_2	r.v. denoting number of datapoints in class C_2
N^x_{11}	r.v. denoting number of datapoints in $x_1,\ C_1$
N^x_{12}	r.v. denoting number of datapoints in $x_1,\ C_2$
N^y_{11}	r.v. denoting number of datapoints in $y_1,\ C_1$
N^y_{12}	r.v. denoting number of datapoints in $y_1,\ C_2$
N_{ijk}	r.v. denoting number of datapoints in $x_i,\ y_j,\ C_k$
N	Size of dataset

Table 3: Notation

trends produce the U-shaped curve of the covariance. This has a significant effect on the overall variance and so the variance also has a similar form with the minimum around 10 folds. Predicting this behavior using only intuition or reasonable number of experiments or just theory is unlikely since it is not clear what the interaction between the two trends is.

In the studies for both hold-out-set error and cross validation error, we observed the similar qualitative results for different priors and for various degrees of correlation between the inputs and outputs.

6.3.4 Comparison of GE, HE and CE

The purpose of our last study we report was to determine the dependency of the three errors on the size of the dataset which indicates the convergence behavior and relative merits of hold-out-set and cross validation. In Figure 10 we plotted the expected value of GE, HE and CE for the same underlying distribution as in the study CE, the size of the hold-out-set for HE was set to 40%(which we found to be the best in experiments not reported for lack of space) and 20 folds for CE. As it can be observed from the figure, the error of hold-out-set is significantly larger for small datasets. The expectation of cross validation is almost on par with the expectation of the generalization error. This property of cross validation to reliably estimate the generalization error is known from empirical studies. But the method can be used to estimate how quickly (at what dataset size) HE and CE converge to GE.

6.4 Extension to Higher Dimensions

All of that we have seen so far was for the 1 dimensional case. What happens when the dimensionality increases? Can we still compute the moments efficiently? The answers to these questions are addressed in the next couple of sections.

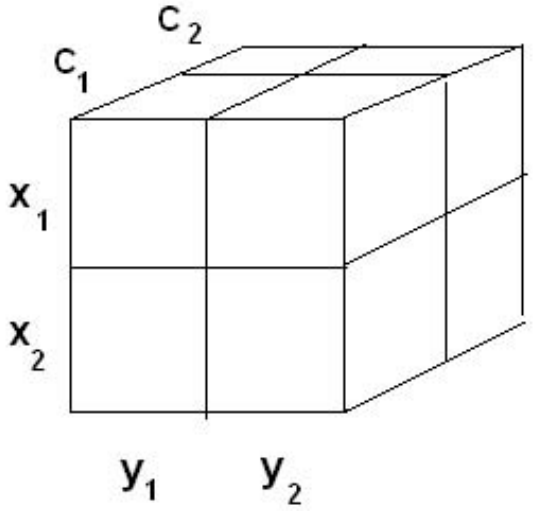


Figure 11: The figure depicts the scenario when we have two attributes each having two values with 2 class lables.

7. Calculation of basic probabilities

Having come up with the probabilistic formulation for discerning the moments of the generalization error, we are now faced with the daunting task of efficiently calculating the probabilities involved for the NBC when the number of dimensions is more than 1. In this section we will mainly discuss single probabilities and the extension to joint probabilities is in the next section. Let us now briefly preview the kind of probabilities we need to decipher.

With reference to Figure 11, considering the cell x_1y_1(w.l.o.g.) and by the Naive Bayes classifier independence assumption, we need to find the probability of the following condition being true for the 2-dimensional case,

$$\begin{aligned} p_{c1}\frac{p^x_{11}}{p_{c1}}\frac{p^y_{11}}{p_{c1}} &> p_{c2}\frac{p^x_{12}}{p_{c2}}\frac{p^y_{12}}{p_{c2}} \\ i.e.\ \ p_{c2}p^x_{11}p^y_{11} &> p_{c1}p^x_{12}p^y_{12} \\ i.e.\ \ N_2N^x_{11}N^y_{11} &> N_1N^x_{12}N^y_{12} \end{aligned}$$

In general for the d-dimensional($d \geq 2$) case we have to find the following probability,

$$P[N_2^{(d-1)}N^{x1}_{11}N^{x2}_{11}...N^{xd}_{11} > N_1^{(d-1)}N^{x1}_{12}N^{x2}_{12}...N^{xd}_{12}] \tag{8}$$

7.1 Direct Calculation

We can find the probability $P[N_2^{(d-1)}N^{x1}_{11}N^{x2}_{11}...N^{xd}_{11} > N_1^{(d-1)}N^{x1}_{12}N^{x2}_{12}...N^{xd}_{12}]$ by summing over all possible assignments of the multinomial random variables involved. For the 2 di-

mensional case shown in Figure 11 we have,

$P[N_2 N_{11}^x N_{11}^y > N_1 N_{12}^x N_{12}^y] =$

$\sum_{N_{111}} \sum_{N_{121}} \sum_{N_{211}} \sum_{N_{112}} \sum_{N_{122}} \sum_{N_{212}} \sum_{N_{222}} P[N_{111},\ N_{121},\ N_{211},\ N_{112},\ N_{122},\ N_{212},\ N_{221},\ N_{222}] \cdot I[N_2 N_{11}^x N_{11}^y > N_1 N_{12}^x N_{12}^y]$

where $N_2 = N_{112} + N_{122} + N_{212} + N_{222}$, $N_{11}^x = N_{111} + N_{121}$, $N_{11}^y = N_{111} + N_{211}$, $N_1 = N - N_2$, $N_{12}^x = N_{112} + N_{122}$, and $I[condition] = 1$ if *condition* is true else $I[condition] = 0$. Each of the summations takes $O(N)$ values and so the worst case time complexity is $O(N^7)$. We thus observe that for the simple scenario depicted, the time to compute the probabilities is unreasonable even for small size datasets($N = 100$ say). The number of summations increases linearly with the dimensionality of the space. Hence, the time complexity is exponential in the dimensionality. We thus need to resort to approximations to speed up the process.

7.2 Approximation Techniques

If all the moments of a random variable are known then we know the moment generating function(MGF) of the random variable and as a consequence the probability generating function and hence the precise cdf for any value in the domain of the random variable. If only a subset of the moments are known then we can at best approximate the MGF and so the cdf.

We need to compute probabilities(cdf's) of the following form $P[X > 0]$ where $X = N_2^{(d-1)} N_{11}^{x1} N_{11}^{x2} ... N_{11}^{xd} - N_1^{(d-1)} N_{12}^{x1} N_{12}^{x2} ... N_{12}^{xd}$. Most of the alternative approximation techniques we propose in the subsections that follow, to efficiently compute the above probabilities(cdf's), are based on the fact that we have knowledge of some finite subset of the moments of the random variable X. We now ellucidate a method to obtain these moments.

Derivation of Moments: As previously mentioned the data generative model we use is a multinomial distribution. We know the moment generating function for it. A moment generating function generates all the moments of a random variable, uniquely defining its distribution. The MGF of a multivariate distribution is defined as follows,

$$M_R(t) = E(e^{R't}) \tag{9}$$

where R is a q dimensional random vector, R' is transpose of R and $t \in R^q$. In our case q is the number of cells in the multinomial.

Taking different order partial derivatives of the moment generating function w.r.t. the elements of t and setting those elements to zero, gives us moments of the product of the random variables in the multinomial raised to those orders. Formally,

$$\frac{\partial^{v1+v2+...+vq} M_R(t)}{\partial t_1^{v1} \partial t_2^{v2} ... \partial t_q^{vq}} \mid_{(t_1=t_2=...=t_q=0)} = E(R_1^{v1} R_2^{v2} ... R_q^{vq}) \tag{10}$$

where $R' = (R_1, R_2, ..., R_q)$, $t = (t_1, t_2, ..., t_q)$ and $v_1, v_2, ..., v_q$ is the order of the partial derivatives w.r.t. $t_1, t_2, ..., t_q$ respectively.

The expressions for these derivatives can be precomputed or computed at run time using tools such as mathematica(Wolfram-Research). But how does all of what we have just discussed relate to our problem? Consider the 2 dimensional case given in Figure 11. We need to find the probability $P[Z > 0]$ where $Z = N_2 N_{11}^x N_{11}^y - N_1 N_{12}^x N_{12}^y$. The individual terms in the product can be expressed as a sum of certain random variables in the multinomial. Thus Z can be written as the sum of the product of some of the multinomial random variables. Consider the first term in Z,

$$\begin{aligned} N_2 N_{11}^x N_{11}^y &= (N_{112} + N_{122} + N_{212} + N_{222})(N_{111} + N_{121})(N_{111} + N_{211}) \\ &= N_{112} N_{111}^2 + ... + N_{222} N_{121} N_{211} \end{aligned}$$

the second term also can be expressed in this form. Thus Z can be written as the sum of the products of the multinomial random variables.

$$\begin{aligned} E(Z) &= E(N_2 N_{11}^x N_{11}^y - N_1 N_{12}^x N_{12}^y) \\ &= E(N_2 N_{11}^x N_{11}^y) - E(N_1 N_{12}^x N_{12}^y) \\ &= E(N_{112} N_{111}^2 + ... + N_{222} N_{121} N_{211}) - \\ &\quad E(N_{111} N_{112}^2 + ... + N_{221} N_{122} N_{212}) \\ &= E(N_{112} N_{111}^2) + ... + E(N_{222} N_{121} N_{211}) - \\ &\quad E(N_{111} N_{112}^2) - ... - E(N_{221} N_{122} N_{212}) \end{aligned}$$

These expectations can be computed using the technique in the discussion before. Higher moments can also be found in the same vein since we would only need to find expectations of higher degree polynomials in the random variables of the multinomial. Similarly, the expressions for the moments in higher dimensions will also include higher degree polynomials.

7.2.1 Series Approximations(SA)

The Edgeworth or the Gram-Charlier A series(Hall, 1992) are used to approximate distributions of random variables whose moments or more specifically cumulants are known. These expansions consist in writing the characteristic function of the unknown distribution whose probability density is to be approximated in terms of the characteristic function of another known distribution(usually normal). The density to be found is then recovered by taking the inverse Fourier transform.
Let $p_{uc}(t)$, $p_{ud}(x)$ and κ_i be the characteristic function, probability density function and the i^{th} cumulant of the unknown distribution respectively. And let $p_{kc}(t)$, $p_{kd}(x)$ and γ_i be the characteristic function, probability density function and the i^{th} cumulant of the known distribution respectively. Hence,

$$\begin{aligned} p_{uc}(t) &= e^{[\sum_{a=1}^{\infty}(\kappa_a - \gamma_a)\frac{(it)^a}{r!}]} p_{kc}(t) \\ p_{ud}(x) &= e^{[\sum_{a=1}^{\infty}(\kappa_a - \gamma_a)\frac{(-D)^a}{r!}]} p_{kd}(x) \end{aligned}$$

where D is the differential operator. If $p_{kd}(x)$ is a normal density then we arrive at the following expansion,

$$p_{ud}(x) = \frac{1}{\sqrt{2\pi\kappa_2}} e^{[\frac{-(x-\kappa_1)^2}{2\kappa_2}]} [1 + \frac{\kappa_3}{\kappa_2^{\frac{3}{2}}} H_3(\frac{x-\kappa_1}{\sqrt{\kappa_2}}) + ...] \tag{11}$$

where $H_3 = (x^3 - 3x)/3!$ is the 3^{rd} Hermite polynomial. This method works reasonably well in practice as can be seen in (Levin, 1981) and (Butler and Sutton, 1998). The major challenge though, lies in choosing a distribution that will approximate the unknown distribution "well", as the accuracy of the cdf estimate depends on this. The performance of the method may vary significantly on the choice of this distribution, since choosing the normal distribution may not always give satisfactory results. This task of choosing an appropriate distribution is non-trivial.

7.2.2 Optimization

We have just seen a method of approximating the cdf using series expansions. Interestingly, this problem can also be framed as an optimization problem, wherein we find upper and lower bounds on the possible values of the cdf by optimizing over the set of all possible distributions having these moments. Since our unknown distribution is an element of this set, its cdf will lie within the bounds computed. This problem is called the classical moment problem and has been studied in literature(Isii, 1960, 1963; Karlin and Shapely, 1953). Infact upto 3 moments known, there are closed form solutions for the bounds(Prekopa, 1989). In the material that follows, we present the optimization problem in its primal and dual form. We then explore strategies for solving it, given the fact that the most obvious ones can prove to be computationally expensive.

Assume that we know m moments of the discrete random variable X denoted by $\mu_1, ..., \mu_m$ where μ_j is the j^{th} moment. The domain of X is given by $U = x_0, x_1, ..., x_n$. $P[X = x_r] = p_r$ where $r \in 0, 1, ..., n$ and $\sum_r p_r = 1$. We only discuss the maximization version of the problem (i.e. finding the upper bound) since the minimization version (i.e. finding the lower bound) has an analogous description. Thus, in the primal space we have the following formulation,

$$max \quad P[X <= x_r] = \sum_{i=0}^{r} p_i,\ r \leq n$$

$$\begin{aligned} subject\ to: & \sum_{i=0}^{n} p_i = 1 \\ & \sum_{i=0}^{n} x_i p_i = \mu_1 \\ & \quad . \\ & \quad . \\ & \quad . \\ & \sum_{i=0}^{n} x_i^m p_i = \mu_m \\ & p_i \geq 0, \qquad \forall\ i \leq n \end{aligned}$$

Solving the above optimization problem gives us an upper bound on $P[X <= x_r]$. On inspecting the formulation of the objective and the constraints we notice that it is a Linear Programming(LP) problem with $m+1$ equality constraints and 1 inequality constraint. The number of optimization variables(i.e. the p_r's) is equal to the size of the domain of X which can be large. For example, in the 2 dimensional case when $X = N_2 N_{11}^x N_{11}^y - N_1 N_{12}^x N_{12}^y$, X

takes $O(N^2)$ values(as N_1 constraints N_2 given N and N_1 also constraints the maximum value of N_{11}^x, N_{11}^y, N_{12}^x and N_{12}^y). Thus with N as small as 100 we already have around 10000(ten thousand) variables. In an attempt to monitor the explosion in computation, we derive the dual formulation of our problem. A dual is a complimentary problem and solving the dual is usually easier than solving the primal since the dual is always convex for primal maximization problems(concave for minimization) irrespective of the form of the primal. For most convex optimization problems(which includes LP) the optimal solution of the dual is the optimal solution of the primal, technically speaking only if the Slaters conditions are satisfied(i.e. duality gap is zero). Maximization(minimization) problems in the primal space map to minimization(maximization) problems in the dual space. For a maximization Standardized LP problem the primal and dual have the following form,

Primal:
$max \;\; c^T x$
$subject\ to: \;\; Ax = b, \;\; x \geq 0$

Dual:
$min \;\; b^T y$
$subject\ to: \;\; A^T y \geq c$
where y is the dual variable.

For our LP problem the dual is the following,

$min \;\; \sum_{k=0}^{m} y_k \mu_k$

$$subject\ to: \;\; \sum_{k=0}^{m} y_k x^k - 1 \geq 0; \qquad \forall\ x \in W$$
$$\sum_{k=0}^{m} y_k x^k \geq 0; \qquad \forall\ x \in U$$

where y_k represent the dual variables and W represents a subset of U over which the cdf is computed. We observe that the number of variables is reduced to just $m+1$ in the dual formulation but the number of constraints has increased to the size of the domain of X. We now propose some strategies to solve this optimization problem; discuss their shortcomings and eventually suggest the strategy preffered by us.

Using Standard Linear Programming Solvers: We have a linear programming problem whose domain is discrete and finite. On careful inspection for our problem we observe that the number of variables in the primal formulation and the number of constraints in the dual increases exponentially in the dimensionality of the space (i.e. the domain of the r.v. X). Though the current state-of-the-art LP solvers(using interior point methods) can solve linear optimization problems of the order of thousands of variables and constraints rapidly, our problem can exceed these counts by a significant margin even for moderate dataset sizes and reasonable dimension thus becoming computationally intractable. Since standard methods for solving this LP can prove to be inefficient we investigate other possibilities.

In the next three approaches we extend the domain of the random variable X to include all the integers between the extremeties of the original domain. The current domain is thus

a superset of the original domain and so are the possible distributions. Another way of looking at it is, in the space of $\bar{y}$'s we have a superset of the original set of constraints. Thus the upper bound calculated in this scenario will be greater than or equal to the upper bound of the original problem. This extension is done to enhance the performance of the next two approaches since it treadjumps the problem of explicitly enumerating the domain of X and is a requirement for the third, as we will soon see.

Gradient descent with binary search(GD): We use gradient descent on the dual to find new values of the vector $\bar{y} = [y_0, ..., y_m]$ and a reasonably large step length since the objective to be minimized is an affine function and it tread jumps the problem of having a large step size in optimizing convex functions. Fixing $\bar{y}$ and assuming x to be continuous the two equations representing the inequalities above can be viewed as polynomials in x. The basic intuition in the method we propose is that if the polynomials are always non-negative within the domain of X then all the constraints are satisfied else some of the constraints are violated. To check if the polynomials are always non-negative we find their roots and perform the following checks. The polynomials will change sign only at their roots and hence we need to carefully examine their behavior at these points. Here are the details of the algorithm.

We check if the roots of the polynomial lie within the extremeties of the domain of X.

1. If they don't then we check to see if the value of polynomial at any point within this range satisfies the inequalities. On the inequalities being satisfied we jump to the next $\bar{y}$ using gradient descent storing the current value of $\bar{y}$ inplace of the previously stored(if it exists) one. If the inequalities are dis-satisfied we reject the value of $\bar{y}$ and perform a binary search between this value and the previous legal value of $\bar{y}$ along the gradient until we reach the value that minimizes the objective satisfying the constraints.

2. If we do, then we check the value of the constraints at the two extremeties. If satisfied and if there exists only one root in the range we store this value of $\bar{y}$ and go on to the next. If there are multiple roots then we check to see if consecutive roots have any integral values between them. If not we again store this value of $\bar{y}$ and move to the next. Else we verify for any point between the roots that the constraints are satisfied based on which we either store or reject the value. On rejecting we perform the same binary search type procedure mentioned above.

Checking if consecutive roots of the polynomial have values in the domain of X, is where the extension in domain to include all integers between the extremeties helps in enhancing perfomance. In the absence of this extension we would need to find if a particular set of integers lie in the domain of X. This operation is expensive for large domains. But with the extension all the above operations can be performed efficiently. Finding roots of polynomials can be done extremely efficiently even for high degree polynomials by various methods such as computing eigenvalues of the companion matrix(Edelman and Murakami, 1995) as is implemented in Matlab. Since the number of roots is just the degree of the polynomial which is the number of moments, the above mentioned checks can be done quickly. The binary search takes $log(t)$ steps where t is the step length. Thus the entire optimization

can be done efficiently. Nonetheless, the method suffers from the following pitfall. The final bound is sensitive to the initial value of $\bar{y}$. Depending on the initial $\bar{y}$ we might stop at different values of the objective on hitting some constraint. We could thus have a suboptimal value as our solution as we only descend on the negative of the gradient. We can somewhat overcome this drawback by making multiple random restarts.

Gradient descent with local topology search(GDTS): Perform gradient descent as mentioned before. Choose a random set of points around the current best solution. Again perform gradient descent on the feasible subset of the choosen points. Choose the best solution and repeat until some reasonable stopping criteria. This works well sometimes in practice but not always.

Prekopa's Algorithm(PA): Prekopa(Prekopa, 1989) gave an algorithm for the discrete moment problem. In his algorithm we maintain an $m+1 \times m+1$ matrix called the basis matrix B which needs to have a particular structure to be dual feasible. We iteratively update the columns of this matrix until it becomes primal feasible, resulting in the optimal solution to the optimization problem[2]. The issue with this algorithm is that there is no guarantee w.r.t. the time required for the algorithm to find this primal feasible basis structure.

In the remaining approaches we further extend the domain of the random variable X to be continuous within the given range. Again for the same reason described before the bound is unintruisive. It is also worthwhile noting that the feasibility region of the optimization problem is convex since the objective and the constraints are convex(actually affine). Standard convex optimization strategies can't be used since the equation of the boundary is unknown and the length of the description of the problem is large.

Sequential Quadratic Programming(SQP): Sequential Quadratic Programming is a method for non-linear optimization. It is known to have local convergence for non-linear non-convex problems and will thus globally converge in the case of convex optimization. The idea behind SQP is the following. We start with an initial feasible point say, $\bar{y}_{init}$. The original objective function is then approximated by a quadratic function around $\bar{y}_{init}$, which then is the objective for that particular iteration. The constraints are approximated by linear constraints around the same point. The solution of the quadratic program is a direction vector along which the next feasible point should be chosen. The step length can be found using standard line search procedures or more sophisticated merit functions. On deriving the new feasible point the procedure is repeated until a suitable stopping criteria. Thus at every iteration a quadratic programming problem is solved.

Let $f(y)$, $ceq_j(y)$ and $cieq_j(y)$ be the objective function, the j^{th} equality constraint and the j^{th} inequality constraint respectively. For the current iterate $\bar{y}_k$ we have following quadratic optimization problem,

$$min \quad O_k(d_k) = f(\bar{y}_k) + \nabla f(\bar{y}_k)^T d_k + \tfrac{1}{2} d_k^T \nabla^2 \mathcal{L}(\bar{y}_k, \lambda_k) d_k$$

2. for explanation of the algorithm read (Prekopa, 1989)

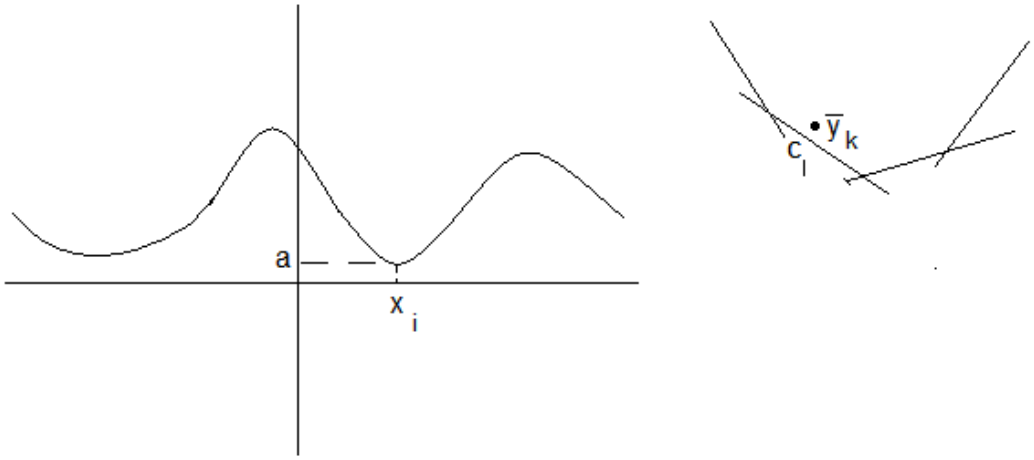


Figure 12: The current iterate $\bar{y}_k$ just satisfies the constraint c_l and easily satisfies the other constraints. Suppose, c_l is $\sum_{j=0}^{m} y_j x_i^j$ where x_i is a value of X, then in the diagram on the left we observe that for the k^{th} iteration $\bar{y} = \bar{y}_k$ the polynomial $\sum_{j=0}^{m} y_j x^j = 0$ has a minimum at $X = x_i$ with the value of the polynomial being a. This is also the value of c_l evaluated at $\bar{y} = \bar{y}_k$.

$$\begin{aligned} subject\ to: ceq_i(\bar{y}_k) + \nabla ceq_i(\bar{y}_k)^T d_k = 0; \quad i \in \mathcal{E} \\ cieq_i(\bar{y}_k) + \nabla cieq_i(\bar{y}_k)^T d_k \geq 0; \quad i \in \mathcal{I} \end{aligned}$$

where $O_k(d_k)$ is the quadratic approximation of the objective function around $\bar{y}_k$. The term $f(\bar{y}_k)$ is generally dropped from the above objective since it is a constant at any particular iteration and has no bearing on the solution. $\nabla^2\mathcal{L}()$ is the Hessian of the Lagrangian w.r.t. $\bar{y}$, $\mathcal{E}$ and $\mathcal{I}$ are the set of indices for the equality and inequality constraints respectively and d_k is the direction vector which is the solution of the above optimization problem. The next iterate $\bar{y}_{k+1}$ is given by $\bar{y}_{k+1} = \bar{y}_k + \alpha_k d_k$ where α_k is the step length.

For our specific problem the objective function is affine, thus a quadratic approximation of it yields the original objective function. We have no equality constraints. For the inequality constraints, we use the following idea. The two equations representing the infinite number of linear constraints given in the dual formulation can be perceived as being polynomials in x with coefficients $\bar{y}$. For a particular iteration with iterate($\bar{y}$) known, we find

the lowest value that the polynomials take. This value is the value of the most violated(if some constraints are violated)/just satisfied(if no constraint is violated) linear constraint. This is shown in Figure 12. The constraint $c_l = \sum_{j=0}^{m} y_j x_i^j$ is just satisfied. With this in view we arrive at the following formulation of our optimization problem at the k^{th} iteration,

$$min \quad \mu^T d_k$$

$$subject\ to: \ \sum_{j=0}^{m} y_j^{(k)} x_i^j + \sum_{j=0}^{m} x_i^j d_k \geq 0;\ \bar{y}_k = [y_0^{(k)}, ..., y_m^{(k)}]^3$$

This technique gives a sense of the non-linear boundary traced out by the constraints. The above mentioned values can be deduced by finding roots of the derivative of the 2 polynomials w.r.t. x and then finding the minimum of these values evaluated at the real roots of its derivative. The number of roots is bounded by the number of moments, infact it is equal to $m-1$. Since this approach does not require the enumeration of each of the linear constraints and operations described are fast with results being accurate, this turns out to be a good option for solving this optimization problem. We carried out the optimization using the Matlab function *fmincon* and the procedure just illustrated.

Semi-definite Programming(SDP): A semi-definite programming problem has a linear objective, linear equality constraints and linear matrix inequality(LMI) constraints. Here is an example formulation,

$$min \ \ c^T q$$

$$\begin{aligned} subject\ to: \ & q_1 F_1 + ... + q_n F_n + H \preceq 0 \\ & Aq = b \end{aligned}$$

where $H,\ F_1, ...,\ F_n$ are positive semidefinite matrices, $q \in R^n$, $b \in R^p$ and $A \in R^{p \times n}$. SDP's can be efficiently solved by interior point methods. As it turns out, we can express our semi-infinite LP as a SDP.

Consider the constraint $c_1(x) = \sum_{i=0}^{m} y_i x^i$. The constraint $c_1(x)$ satisfies $c_1(x) \geq 0$ $\forall x \in [a, b]$ iff $\exists$ a $m+1 \times m+1$ positive semidefinite matrix S such that,

$$\sum_{i+j=2l-1} S(i,j) = 0; \qquad l = 1, ..., m$$

$$\sum_{k=0}^{l} \sum_{r=k}^{k+m-l} y_r r C_k (m-r) C_{l-k} a^{r-k} b^k = \sum_{i+j=2l} S(i,j); \qquad l = 0, ..., m$$

$S \succeq 0$ means S is positive semidefinite.

The proof of this result is given in (Bertsimas and Popescu, 1998).

We derive the equivalent semidefinite formulation for the second constraint $c_2(x) = \sum_{i=0}^{m} y_i x^i - 1$ to be greater than or equal to zero. To accomplish this, we replace y_0 by

3. $y_i^{(k)}$ is the value of y_i at the k^{th} iteration

$y_0 - 1$ in the above set of equalities since $c_2(x) = c_1(x) - 1$. Thus $\forall x \in [a, b]$ we have the following semidefinite formulation for the second constraint,

$\sum_{i+j=2l-1} S(i,j) = 0; \qquad l = 1, ..., m$

$\sum_{k=1}^{l} \sum_{r=k}^{k+m-l} y_r r C_k (m-r) C_{l-k} a^{r-k} b^k +$
$\sum_{r=1}^{m-l} y_r (m-r) C_l a^r + y_0 - 1 = \sum_{i+j=2l} S(i,j); \qquad l = 1, ..., m$

$\sum_{r=1}^{m} y_r a^r + y_0 - 1 = S(0,0)$

$S \succeq 0$

Combining the above 2 results we have the following semidefinite program with $O(m^2)$ constraints,

$min \quad \sum_{k=0}^{m} y_k \mu_k$

$subject\ to: \quad \sum_{i+j=2l-1} G(i,j) = 0; \qquad l = 1, ..., m$

$\sum_{k=1}^{l} \sum_{r=k}^{k+m-l} y_r r C_k (m-r) C_{l-k} a^{r-k} b^k +$
$\sum_{r=1}^{m-l} y_r (m-r) C_l a^r + y_0 - 1 = \sum_{i+j=2l} G(i,j); \qquad l = 1, ..., m$

$\sum_{r=1}^{m} y_r a^r + y_0 - 1 = G(0,0)$

$\sum_{i+j=2l-1} Z(i,j) = 0; \qquad l = 1, ..., m$

$\sum_{k=0}^{l} \sum_{r=k}^{k+m-l} y_r r C_k (m-r) C_{l-k} b^{r-k} c^k = \sum_{i+j=2l} Z(i,j); \qquad l = 0, ..., m$

$G \succeq 0, \qquad Z \succeq 0$

G and Z are $m+1 \times m+1$ positive semidefinite matrices. The domain of the random variable is $[a, c]$. Solving this semidefinite program yields an upper bound on the cdf $P[X <= b]$, where $a \leq b \leq c$. We used a free online SDP solver(Wu and Boyd, 1996) to solve the above semidefinite program. Through empirical studies that follow we found this approach to be the best in solving the optimization problem in terms of a balance between speed, reliability and accuracy.

7.2.3 Random Sampling(RS)

In sampling we select a subset of observations from the universe consisting of all possible observations. Using this subset we calculate a function whose value we consider to be equal(or at least close enough) to the value of the same function applied to the entire observation set. Sampling is an important process, since in many cases we do not have access to this entire observation set(many times it is infinite). Numerous studies(Hall, 1992; Bartlett J. E. and C., 2001; L and J, 1977) have been conducted to analyze different

kinds of sampling procedures. The sampling procedure that is relevant to our problem is Random Sampling and hence we restrict our discussion only to it.

Random sampling is a sampling technique in which we select a sample from a larger population wherein each individual is chosen entirely by chance and each member of the population has possibly an unequal chance of being included in the sample. Random sampling reduces the likelihood of bias. It is known that asymptotically the estimates found using random sampling converge to their true values.

For our problem the cdf's can be computed using this sampling procedure. We sample data from the multinomial distribution(our data generative model) and add the number of times the condition whose cdf is to be computed is true. This number when divided by the total number of samples gives an estimate of the cdf. By finding the mean and standard deviation of these estimates we can derive confidence bounds on the cdf's using Chebyshev's inequality. The width of these confidence bounds depends on the standard deviation of the estimates, which in turn depends on the number samples used to compute the estimates. As the number of samples increases the bounds become tighter. We will observe this in the experiments that follow. Infact, all the estimates for the cdf's necessary for computing the moments of the generalization error using the mathematical formulations given by us, can be computed parrallely. The moments are thus functions of the samples for which confidence bounds can be derived. Realize that, only sampling in conjunction with our formulations for the moments makes for an efficient method. If we directly use sampling without using the formulations, we would first need to sample for building a set of classifiers, and then for each classifier built we would need to sample test sets from the multinomial. Reason being that the expectation in the moments is w.r.t. the $D \times X$ space. This process can prove to be computationally intensive for acquiring accurate estimates of the moments, as the required sample sizes can potentially be large. Thus we are still consistent with our methodology of going as far as possible in theory to reduce the computation for conducting experiments.

7.3 Empirical Comparison of the cdf computing methods

Consider the 2 dimensional case in Figure 11. We instantiated all the cell probabilities to be equal. We found the probability $P[N_2 N_{11}^x N_{11}^y > N_1 N_{12}^x N_{12}^y]$ by the methods suggested, varying the dataset size from 10 to 1000 in multiples of 10 and having knowledge of the first six moments of the random variable $X = N_2 N_{11}^x N_{11}^y - N_1 N_{12}^x N_{12}^y$. The actual probability in all the three cases is around 0.5(actually just lesser than 0.5). The execution speeds for the various methods are given in Table 4[4]. From the table we see that the SDP and gradient descent methods are lightining fast. The SQP and gradient descent with topology search methods take a couple of seconds to execute. The thing to notice here is that SDP, SQP, the two gradient descent methods and the series approximation method are oblivious to the size of the dataset with regards to execution time. In terms of accuracy the gradient descent method is sensitive to initialization and the series approximation method to the choice of distribution, as previously stated. A normal distribution gives an estimate of 0.5 which is good in this case since the original distribution is symmetric about the origin. But for finding cdf's near the extremeties of the domain of X the error can be considerable. Since

4. arnd means around

Method	Dataset Size 10	Dataset Size 100	Dataset Size 1000
Direct	25 hrs	arnd 200 centuries	arnd 200 billion yrs
SA	Instantaneous	Instantaneous	Instantaneous
LP	arnd 3.5 sec	arnd 2 min	arnd 2:30 hrs
GD	arnd 0.13s	arnd 0.13 sec	arnd 0.13 sec
PA	arnd 1s	arnd 25 sec	arnd 5 min
GDTS	arnd 3.5 sec	arnd 3.5 sec	arnd 3.5 sec
SQP	arnd 3.5 sec	arnd 3.5 sec	arnd 3.5 sec
SDP	arnd 0.1 sec	arnd 0.1 sec	arnd 0.1 sec
RS_{100}	arnd 0.08 sec	arnd 0.08 sec	arnd 0.1 sec
RS_{1000}	arnd 0.65 sec	arnd 0.66 sec	arnd 0.98 sec
RS_{10000}	arnd 6.3 sec	arnd 6.5 sec	arnd 9.6 sec

Table 4: Empirical Comparison of the cdf computing methods in terms of execution time. RS_n denotes the Random Sampling procedure using n samples to estimate the probabilities.

Samples	Dataset Size 10	Dataset Size 100	Dataset Size 1000
100	0.7-0.23	0.72-0.26	0.69-0.31
1000	0.54-0.4	0.56-0.42	0.57-0.42
10000	0.5-0.44	0.51-0.47	0.52-0.48

Table 5: 95% confidence bounds for Random Sampling.

Method	Accuracy	Speed
Direct	Exact solution	Low
Series Approximation	Variable	High
Standard LP solvers	High	Low
Gradient descent	Low	High
Prekopa's Algorithm	High	Moderate
Gradient descent (topology search)	Moderate	Moderate
Sequential Quadratic Programming	High	Moderate
Semi-definite Programming	High	High
Random Sampling	Very High	High - Moderate

Table 6: Comparison of methods for computing the cdf

the domain of X is finite, variants of the beta distribution with a change of variable(i.e. shifting and scaling the distribution) can provide better approximation capabilities. The SQP and SDP methods are robust and insensitive to initialization(as long as the initial point is feasible). The bound found by SQP is 0.64 to 0.34 and that found by SDP is 0.62 to 0.33. The LP solver also finds a similar bound of 0.62 to 0.34 but the execution time scales quadratically with the size of the input. For RS we observe from Table 4 and Table 5 that the method does not scale much in time with the size of the dataset but produces extremely good confidence bounds as the number of samples increases. With 1000 samples we already have pretty tight bounds with time required being just over half a second. Also as previously stated the cdf's can be calculated together rather than independently and hence using 10000 samples for estimation can prove to be more than acceptable.

8. Calculation of Cumulative Joint probabilities

Cumulative Joint probabilities need to be calculated for computation of higher moments. In the Random Sampling method, these probabilities can be computed in similar fashion as the single probabilities shown above. But for the other methods knowledge of the moments is required. Cumulative Joint probabilities are defined over multiple random variables wherein each random variable satisfies some inequality or equality. In our case, for the second moment we need to find the following kind of cumulative joint probabilities $P[X > 0, Y > 0]$, where X and Y are random variables(overiding their definition in Table 1). Since the probability is of an event over two distinct random variables the previous method of computing moments cannot be directly applied. An important question is can we somehow through certain transformations reuse the previous method? Fortunately the answer is affirmative. The intuition behind the technique we propose is as follows. We find another random variable $Z = f(X, Y)$(polynomial in X and Y) such that $Z > 0$ iff $X > 0$ and $Y > 0$. Since the two events are equivalent their probabilities are also equal. By taking derivatives of the MGF of the multinomial we get expressions for the moments of polynomials of the multinomial random variables. Thus, $f(X, Y)$ is required to be a polynomial in X and Y. We now discuss the challenges in finding such a function and eventually suggest a solution.

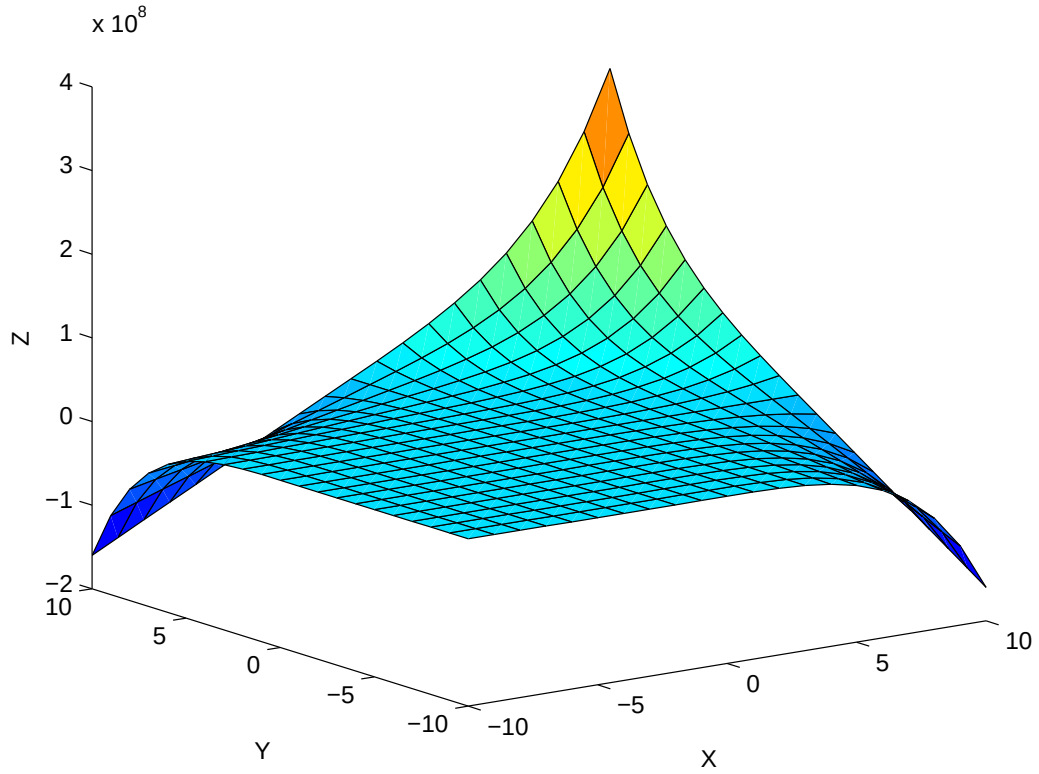


Figure 13: The plot is of the polynomial $(x+10)^4x^2y + (y+10)^4y^2x - z = 0$. We see that it is positive in the first quadrant and non-positive in the remaining three.

Geometrically, we can consider the random variables X, Y and Z to denote the three co-ordinate axes. Then the function $f(X, Y)$ should have a positive value in the first quadrant and negative in the remaining three. If the domains of X and Y were infinite and continuous then this problem is potentially intractable since the polynomial needs to have a discrete jump along the X and Y axis. Such behavior can be emulated at best approximately by polynomials. In our case though, the domains of the random variables are finite, discrete and symmetric about the origin. Therefore, what we care about is that the function behaves as desired only at these finite number of discrete points. One simple solution is to have a circle covering the relevant points in the first quadrant and with appropriate sign the function would be positive for all the points encompassed by it. This works for small domains of X and Y. As the domain size increases the circle intrudes into the other quadrants and no longer satisfying the conditions. Other simple functions such as XY or $X+Y$ or a product of the two also do not work. We now give a function that does work and discuss the basic intuition in constructing it. Consider the domain of X and Y to be integers in the interval [-a,a].[5] Then the polynomial is given by,

$$Z = (X + a)^r X^2 Y + (Y + a)^r Y^2 X \tag{12}$$

where $r = max_b \lfloor \frac{ln[b]}{ln[\frac{a+1}{a-b}]} \rfloor + 1$, $1 < b < a$ and $b \in \mathbb{N}$.[6] The value of r can be found numerically by finding the corresponding value of b which maximizes that function. For $5 \leq a \leq 10$ the value of b which does this is 5. For larger values $10 < a \leq 10^6$ the value of b is 4. Figure 13 depicts the polynomial for $a = 10$ where $r = 4$. The polynomial resembles a bird with its neck in the first quadrant, wings in the 2^{nd} and 4^{th} quadrants and its posterior in the third. The general shape remains same for higher values of a.

The first requirement for the polynomial was that it must be symmetric. Secondly, we wanted to penalise negative terms and so we have $X + a$(and $Y + a$) raised to some power which will always be positive but will have lower values for lesser X(and Y). The X^2Y(and Y^2X) makes the first(second) term zero if any of X and Y are zero. Moreover, it imparts sign to the corresponding term. If absolute value function(||) could be used we would replace the $X^2(Y^2)$ by $|X|(|Y|)$ and set $r = 1$. But since we cannot; in the resultant function r is a reciprocal of a logarithmic function of a. For a fixed r with increasing value of a the polynomial starts violating the biconditional by becoming positive in the 2^{nd} and 4^{th} quadrants(i.e. the wings rise). The polynomial is always valid in the 1^{st} and 3^{rd} quadrants. With increase in degree(r) of the polynomial its wings begin flattening out, thus satisfying the biconditional for a certain a.

With this we have all the ingredients necessary to perform experiments reasonably quickly. This is exactly what we report in the next section.

9. Experiments

Having come up with strategies to compute the probabilities and hence the moments of the errors efficiently, we now report some of the empirical studies we conducted. These

5. in our problem X and Y have the same domain.
6. proof in appendix

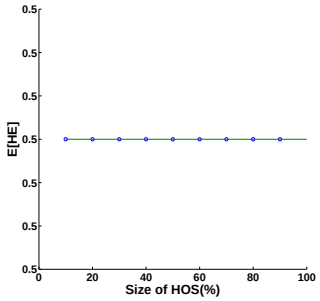


Figure 14: HE expectation

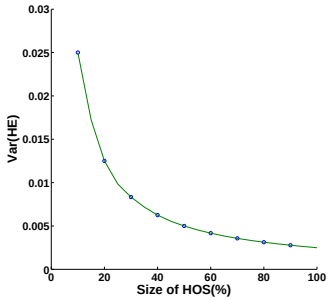


Figure 15: HE variance

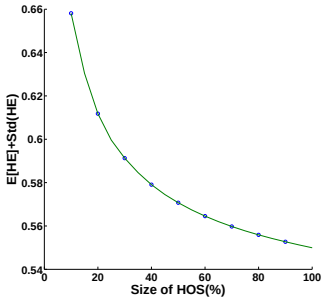


Figure 16: HE $E[] + Std()$

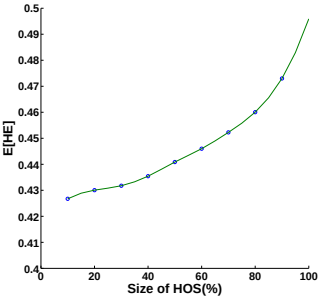


Figure 17: HE expectation

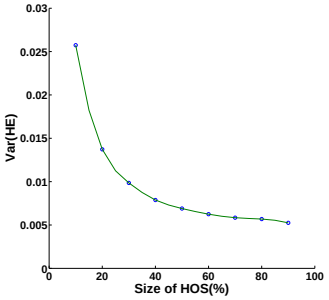


Figure 18: HE variance

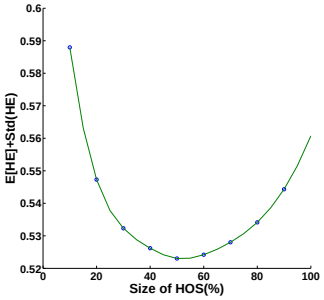


Figure 19: HE $E[] + Std()$

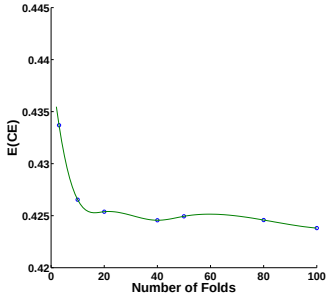


Figure 20: CE expectation

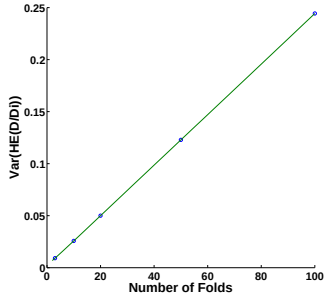


Figure 21: Individual run variance of CE

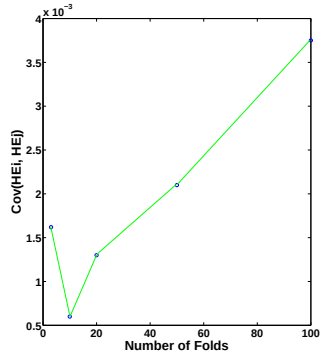


Figure 22: Pairwise covariances of CV.

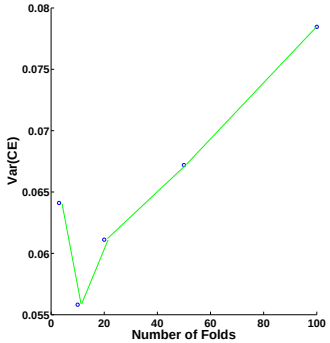


Figure 23: Total variance of cross validation.

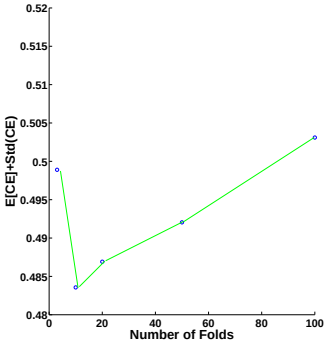


Figure 24: $E[] + \sqrt{\mathrm{Var}(())}$ of CV

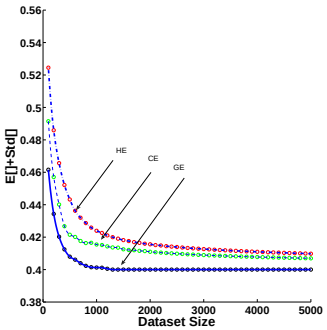


Figure 25: Convergence behavior.

empirical studies were carried out on the $2 \times 2 \times 2$ cube in Figure 11(i.e. 2 attributes each having two values and 2 classes).

9.1 HOS

In our first study we set all the joint cell probabilities to be equal. For N=100 we wanted to observe the behavior of the moments of HE under varying HOS size. We see from Figure 14 that the E[HE] is 0.5 throughout. This is the case since each class is equally likely and thus the training set carries little information that can assist the NBC in classification. This also affects the variance which reduces as the test size increases(Figure 15). The reason for this being that the stability of the training set is a non-issue here and so larger the test set a better estimate one gets of the error. Combining these two results we see from Figure 16 that the estimate is more accurate for larger HOS sizes.

In our next study we set the joint probabilities in class 1 to 0.1 and those in class 2 to 0.15. In this case we observe qualitatively similar results as those obtained for the 1 dimensional scenario. The E[HE] increases as the training set size reduces(Figure 17). The best performance is observed for HOS size of 50%(Figure 19).

9.2 Cross Validation

In our third study we observed the behavior of CV with varying number of folds. We set the joint probabilities the same as our previous study. Here too we observe the same qualitative results we saw for the 1 dimensional case. The variance of the individual runs increases with the number of folds(Figure 21). The covariances between errors of two runs is high initially, later drops and then increases again. The reasoning behind this is the same as described before. The best performance is observed when the number of folds is 10.

9.3 Comparison of GE, HE, and CE

In our final study we observe the behavior of the three forementioned errors with increase in dataset size. We again set the joint probabilities the same as our last two studies. We set the HOS size to 50%, the number of folds in CV to 10 as these proved to be the best in our previous studies. From Figure 25 we observe that consistently 10 fold CV is better than HOS, though as the dataset size increases this difference in performance reduces.

This type of study can be used to observe the non-asymptotic convergence behavior of errors.

10. Conclusion

In this section we summarize the major developments in this work and mention a future line of research.

The major contributions in this paper are: (a) we developed general relationships between the moments of GE, HE, CE, (b) we reduced the granularity by shifting the focus from looking at the entire input at one time to looking at only particular inputs, (c) we developed efficient formulations for the moments of the errors for the NBC using the fact in (b), (d) we proposed a plethora of strategies to efficiently compute cdf's required for the moment computation in higher dimensions for the NBC, thus making our method scalable,

and (e) we plotted our formulations to be able interpret what they mean in different circumstances. The major challenge in future work is to extend the analysis to more *sophisticated* classification models such as decision trees, neural nets, SVM's etc. We have however developed a characterization for the NBC which is important in its own right. It is a model which is extensively used in industry, due to its robustness outperforming its more sophisticated counterparts in some real world applications(eg. spam filtering in Mozilla Thunderbird and Microsoft Outlook, bio-informatics etc.).

In conclusion, we proposed a methodology to observe and understand non-asymptotic behavior of errors for the classification model at hand by making as much progress as possible in theory in view of reducing the computation required for experiments, and then performed experiments to study specific situations that we are interested in. The extent to which this methodology can be used for studying learning methods is a part of our current investigation, however we feel the method has promise.

11. Appendix

Proposition 3 *The polynomial* $(x+a)^r x^2 y + (y+a)^r y^2 x > 0$ *iff* $x > 0$ *and* $y > 0$*, where* $x, y \in [-a, -a+1, ..., a]$*,* $r = max_b \lfloor \frac{ln[b]}{ln[\frac{a+1}{a-b}]} \rfloor + 1$*,* $a \in \mathbb{N}$*,* $1 < b < a$ *and* $b \in \mathbb{N}$*.*

Proof One direction is trivial. If $x > 0$ and $y > 0$ then definitely the polynomial is greater than zero for any value of r. Now lets prove the other direction i.e. if $(x+a)^r x^2 y + (y+a)^r y^2 x > 0$ then $x > 0$ and $y > 0$ where $x, y \in [-a, ..., a]$, $r = max_b \lfloor \frac{ln[b]}{ln[\frac{a+1}{a-b}]} \rfloor + 1$, $1 < b < a$ and $b \in \mathbb{N}$. In other words, if $x \leq 0$ or $y \leq 0$ then $(x+a)^r x^2 y + (y+a)^r y^2 x \leq 0$. We prove this result by forming cases.

Case 1: Both x and y are zero.
The value of the polynomial is zero.

Case 2: One of x or y is zero.
The value of the polynomial again is zero since each of the two terms separated by a sum have xy as a factor.

Case 3: Both x and y are less than zero.
Consider the first term $(x+a)^r x^2 y$. This term is non-positive since $x + a$ is always non-negative(since $x \in [-a, ..., a]$) and x^2 is always positive but y is non-positive. Analogous argument for the second term and so it too is non-positive. Thus their sum is non-positive.

Case 4: One of x or y is negative and the other is positive. Assume w.l.o.g. that x is positive and y is negative.

$$\begin{aligned} (x+a)^r x^2 y + (y+a)^r y^2 x &\leq 0 \\ only\ if \quad (x+a)^r x + (y+a)^r y &\geq 0 \\ only\ if \quad r &\geq \frac{ln[\frac{-y}{x}]}{ln[\frac{x+a}{y+a}]} \end{aligned}$$

On fixing the value of y the value of x at which the right hand side of the above achieves maximum is 1(since lower the value of x higher the right handside but x is positive by our

assumption and $x \in [-a, ..., a]$). Thus we have the above inequality true only if,

$$r \geq \frac{ln[-y]}{ln[\frac{a+1}{y+a}]}$$

Let $b = -y$ then $1 \leq b \leq a$ since y is negative. Hence, if r satisfies the inequality for all possible allowed values of b then only will it imply that the polynomial is less than or equal to zero in the specified range. For r to satisfy the inequality for all allowed values of b, it must satisfy the inequality for the value of b that the function is maximum. Also, for $b = 1$ and $b = a$ the right handside is zero. So the range of b over which we want to find the maximum is $1 < b < a$. With this the minimum value of r that satifies the above inequality in order for the polynomial to be less than or equal to zero is,

$$r = max_b \lfloor \frac{ln[b]}{ln[\frac{a+1}{a-b}]} \rfloor + 1$$

The 4 cases cover all the possibilities and thus we have shown that if $x \leq 0$ or $y \leq 0$ then $(x + a)^r x^2 y + (y + a)^r y^2 x \leq 0$. Having also shown the other direction we have proved the proposition. ■

References

Elisseeff A and Pontil M. *Learning Theory and Practice*, chapter Leave-one-out error and stability of learning algorithms with applications. IOS Press, 2003.

Kotrlik J. W. Bartlett J. E. and Higgins C. Organizational research: Determining appropriate sample size for survey research. *Information Technology, Learning, and Performance Journal*, 19(1):43–50, 2001. URL `citeseer.ist.psu.edu/goutte97note.html`.

Y. Bengio and Y. Grandvalet. No unbiased estimator of the variance of k-fold cross validation. *Journal of Machine Learning Research*, 2003.

D. Bertsimas and I. Popescu. Optimal inequalities in probability theory: a convex optimization approach. Technical report, Dept. Math. O.R., Cambridge, Mass 02139, 1998. URL `citeseer.ist.psu.edu/bertsimas00optimal.html`.

Avrim Blum, Adam Kalai, and John Langford. Beating the hold-out: Bounds for k-fold and progressive cross-validation. In *Computational Learing Theory*, pages 203–208, 1999. URL `citeseer.ist.psu.edu/blum99beating.html`.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge, 2004.

L. Breiman. Heuristics of instability and stabilization in model selection. *The Annals of Statistics*, 1996.

Ronald W. Butler and Richard K. Sutton. Saddlepoint approximation for multivariate cumulative distribution functions and probability computations in sampling theory and outlier testing. *Journal of the American Statistical Association*, 93(442):596–604, 1998.

Pedro Domingos and Michael J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

Alan Edelman and H. Murakami. Polynomial roots from companion matrix eigenvalues. *Mathematics of Computation*, 64(210):763–776, 1995. URL `citeseer.ist.psu.edu/edelman95polynomial.html`.

Cyril Goutte. Note on free lunches and cross-validation. *Neural Computation*, 9(6):1245–1249, 1997. URL `citeseer.ist.psu.edu/goutte97note.html`.

Peter Hall. *The Bootstrap and Edgeworth Expansion*. Springer-Verlag, 1992.

K. Isii. The extrema of probability determined by generalized moments(i) bounded random variables. *Ann. Inst. Stat. Math*, 12:119–133, 1960.

K. Isii. On the sharpeness of chebyshev-type inequalities. *Ann. Inst. Stat. Math*, 14:185–197, 1963.

S. Karlin and L.S. Shapely. Geometry of moment spaces. *Memoirs Amer. Math. Soc.*, 12, 1953.

Michael J. Kearns and Dana Ron. Algorithmic stability and sanity-check bounds for leave-one-out cross-validation. In *Computational Learing Theory*, pages 152–162, 1997. URL `citeseer.ist.psu.edu/kearns97algorithmic.html`.

R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pages 1137–1143. San Mateo, CA: Morgan Kaufmann, 1995.*, 1995. URL `overcite.lcs.mit.edu/kohavi95study.html`.

Samuel Kutin and Partha Niyogi. Almost-everywhere algorithmic stability and generalization error. In *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 275–282, San Francisco, CA, 2002. Morgan Kaufmann Publishers.

Chambers R L and Skinner C J. *Analysis of Survey Data*. Wiley, 1977.

L. Gyor L. Devroye and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer-Verlag, 1996.

John Langford. Filed under: Prediction theory, problems. http://hunch.net/index.php?p=29, 2005.

Pat Langley and Stephanie Sage. Tractable average-case analysis of naive Bayesian classifiers. In *Proc. 16th International Conf. on Machine Learning*, pages 220–228. Morgan Kaufmann, San Francisco, CA, 1999. URL `citeseer.ist.psu.edu/article/langley99tractable.html`.

Bruce Levin. A representation for multinomial cumulative distribution functions. *The Annals of Statistics*, 9(5):1123–1126, 1981.

Andrew W. Moore and Mary S. Lee. Efficient algorithms for minimizing cross validation error. In *International Conference on Machine Learning*, pages 190–198, 1994. URL `citeseer.ist.psu.edu/moore94efficient.html`.

Stephane Boucheron Olivier Bousquet and Gabor Lugosi. Introduction to statistical learning theory. http://www.kyb.mpg.de/publications/pdfs/pdf2819.pdf, 2005.

M.E. Plutowski. Survey: Cross-validation in theory and in practice. www.emotivate.com/CvSurvey.doc, 1996.

Andras Prekopa. The discrete moment problem and linear programming. RUTCOR Research Report, 1989.

Jun Shao. Linear model selection by cross validation. *Journal of the American Statistical Association*, 88, 1993.

Vapnik. *Statistical Learning Theory*. Wiley & Sons, 1998.

Robert C. Williamson. Srm and vc theory(statistical learning theory). http://axiom.anu.edu.au/ williams/papers/P151.pdf, 2001.

Wolfram-Research. Mathematica. http://www.wolfram.com/.

Shao-Po Wu and Stephen Boyd. Sdpsol: a parser/solver for sdp and maxdet problems with matrix structure. http://www.stanford.edu/ boyd/SDPSOL.html, 1996.

Huaiyu Zhu and Richard Rohwer. No free lunch for cross validation. *Neural Computation*, 8(7):1421–1426, 1996. URL `citeseer.ist.psu.edu/zhu96no.html`.