An Efficient Technique for Calculating Exact Nearest-Neighbor Classification Accuracy

Matthew Mullin
Just Research
4616 Henry Street
Pittsburgh, PA 15213
mdm@justresearch.com

Rahul Sukthankar
Just Research &
The Robotics Institute, Carnegie Mellon
Pittsburgh, PA 15213
rahuls@cs.cmu.edu

Abstract

We present a technique for calculating exact nearest-neighbor classification accuracy. This is equivalent to averaging the results of an exponential number of trials (all test/train splits), yet it can be performed very efficiently. The technique is applied to each of four common classification experiment types. Complexity analysis and empirical results demonstrate the superiority of this algorithm over the customary approach of estimating accuracy by averaging several randomized test/train splits. This algorithm offers significant practical benefits to researchers in terms of vastly reduced computation time.

1 Introduction

In machine learning experiments, a pool of labeled data, S, is typically split into a training set, T, and a test set $S \setminus T$.[1] Items from the test set are presented to a classifier trained on T, and the empirical accuracy (fraction of test items classified correctly) is reported as the performance of the classifier. Given that the accuracy observed in such an experiment depends on which items from S appeared in the training set, the variance of this estimate can be significant [7]. Since calculating an expectation over all permissible test/train splits is normally impractical (the number of such experiments would grow exponentially with $|S|$), it is customary to create a lower variance estimator simply by averaging the accuracies reported from several trials, either from a random test/train split, or using n-fold cross validation. Unfortunately, running these experiments is still very time-consuming, particularly when an accurate estimate of the accuracy is desired. This paper presents a technique for calculating the *exact* sample classification accuracy[2] for the nearest-neighbor [2] family of classifiers. Not only is this accuracy identical to that obtained by an exhaustive enumeration of splits, it can be computed more efficiently than averaging results from several randomized trials.

The remainder of the paper is organized as follows. Section 2 applies our technique to four common classification experiment types, along with a complexity analysis. Section 3 presents empirical results that support these theoretical claims. Section 4 discusses applications and concludes with an outline of future research.

[1] $S \setminus T$ denotes the *set difference* of S and T: the elements of S that are not in T.

[2] This corresponds to *sample error* over the available data in [7].

2 Computing exact classification accuracy

In machine learning experiments, test/train splits are typically constrained by requirements on the size and structure in the training or test sets. Let $\mathcal{T}$ denote the set of training sets satisfying these constraints. The accuracy of a classifier in one trial (trained on a given $T \in \mathcal{T}$) is defined to be the fraction of items in the test set that are correctly identified. The exact sample accuracy of a classifier (which is independent of T) is defined to be the average of the accuracies, over $\forall T \in \mathcal{T}$. The goal of evaluation is to obtain this expected accuracy. This goal is normally intractable given the prohibitive size of $\mathcal{T}$. For instance, even for $|S| = 100$ (a small dataset) with the constraint $|T| = 50$ (half-half split), $|\mathcal{T}| = \binom{100}{50} \approx 10^{29}$. Thus, it is customary to estimate the accuracy of the classifier by averaging several (typically no more than 50) trials, each using a randomly-generated $T \in \mathcal{T}$ [7].

In this section, we show that it is not only feasible, but *efficient* to calculate the *exact* accuracies for certain classifiers. We demonstrate that, in the case of nearest-neighbor, this technique is also more accurate and more efficient than the standard approach outlined above. The following definitions are used in the remainder of the paper:

$c(x)$	The class label associated with x
C	Number of class labels in S
N	Size of S ($\|S\|$)
N_k	Number of elements in class k. $N_k = \|\{x : c(x) = k\}\|$

We note that the exact accuracy is obtained by dividing the total number of correct identifications, A, by the total number of possible classifications. The former is given by:

$$A = \sum_{T \in \mathcal{T}} \sum_{x \notin T} \text{correct}(x, T), \tag{1}$$

while the latter is simply $\sum_{T \in \mathcal{T}} |S \backslash T|$. The binary function, correct(x, T), returns 1 iff x is correctly identified[3] by a classifier trained on T. Equation 1 is a large sum (with $|\mathcal{T}|$ terms) of small inner sums (each with fewer than S terms). We focus on efficiently computing this seemingly-intractable sum. We begin by manipulating Equation 1 into a more manageable form:

$$A = \sum_{T \in \mathcal{T}} \sum_{x \notin T} \text{correct}(x, T) = \sum_{\substack{(x,T) \in S \times \mathcal{T} \\ x \notin T, T \in \mathcal{T}}} \text{correct}(x, T) = \sum_{x \in S} \underbrace{\sum_{\substack{T \in \mathcal{T} \\ T \not\ni x}} \text{correct}(x, T)}_{A_x}. \tag{2}$$

This identity expresses the key insight of our technique: that the total number of correct identifications can be computed in two equivalent ways. In the standard method, the number of correct identifications for a particular test/train split is summed over all possible splits (LHS). Equivalently, one can count the number of *splits*, A_x, for which a particular item in the test set, x, is correctly identified and sum over every item in the data set (RHS). Computationally, this transforms the problem into a small sum of large inner sums. Fortunately, in the case of 1-nearest-neighbor, we can efficiently calculate A_x without explicit enumeration, enabling us to directly obtain exact classification accuracy.

2.1 Nearest-neighbor classification

The training phase of the basic nearest-neighbor classifier simply consists of storing all of the labeled training items in a list. In the testing phase, to classify an item x from the test set, the distance from x to each of the items in the training set is computed, and x

[3] It is implicit that $x \in S$, so that $x \notin T \Leftrightarrow x \in S \backslash T$.

is assigned the label of its closest training instance. Despite its obvious simplicity, the nearest-neighbor classifier has good theoretical properties [1], and has been successfully applied to a broad range of problems. See [2, 7] for more details on this classifier, and its common variants. We now examine the aspects of nearest-neighbor that enable efficient computation of exact accuracy.

The distance between two items is just a measure of similarity that may involve arbitrary transformations; it need not satisfy metric properties such as symmetry or the triangle inequality. However, since this distance is invariant over test/train splits, we can create a *fixed* ordering of items in S, sorted in increasing distance from x, for every $x \in S$. For a particular item, x, this ordered list looks like:[4]

$$\begin{array}{rl} x: & x_1\ x_2\ x_3\ \ldots\ x_i\ \ldots\ x_{N-2}\ x_{N-1}. \\ & \leftarrow \text{Nearest} \qquad\qquad \text{Farthest} \rightarrow \end{array}$$

This list contains all of the items in S; for a given split, some of the x_i items will be in the training set and the remainder will be in the test set. A given split correctly classifies x iff x and its nearest neighbor have the same class label. Note that x's nearest neighbor is not necessarily x_1; it is the leftmost x_i that has been assigned to the training set. Furthermore, the assignment of elements to the right of the nearest neighbor (into T or $S \setminus T$) does not affect whether the split correctly classifies x; all such splits give identical classification for x, and need not be individually enumerated. This observation enables us to avoid computing the large inner sum, A_x in Equation 2. We now demonstrate how our technique can be applied to four common experiment types, and in each case, derive an equivalent efficient expression for A_x.

2.2 Algorithm 1: class-independent test/train split

In this type of experiment, items from S are assigned to T, with uniform probability, without replacement, until the T reaches a specified size. For instance, the experimenter may choose to use half of the data for training, and save the remaining half for testing. Let α denote the required size of the training set. Now, since $|T| = \alpha$, $|\mathcal{T}| = \binom{N}{\alpha}$, which will be very large unless α is close to 0 or N.[5] From the ordering shown in Section 2.1, we determine how many training sets correctly classify x.

Consider the set of training sets, $\mathcal{E} \subset \mathcal{T}$, where x's nearest neighbor is x_i. If the $c(x) = c(x_i)$, then all classifiers using $T \in \mathcal{E}$ correctly classify x, else none of them do; thus, the number of total correct classifications should be incremented by $|\mathcal{E}|$ iff $c(x) = c(x_i)$. By examining the additional constraints on $\mathcal{E}$, we shall see that it is possible to obtain $|\mathcal{E}|$ without explicitly enumerating any element of $\mathcal{E}$. Our constraints are: (1) the items $x_1, \ldots, x_{i-1}$ cannot be in T, since x_i is the nearest neighbor; (2) aside from x_i, there are $\alpha - 1$ items in the training set, all of which must appear somewhere in the $N - i - 1$ available positions to the right of x_i (see the diagram in Section 2.1). Therefore, given x and x_i, there are $\binom{N-i-1}{\alpha-1}$ splits that will create training sets satisfying these constraints. Iterating over all possible positions x_i gives us:[6]

$$A_x = \sum_{i=1}^{N-1} I(x, x_i) \binom{N-i-1}{\alpha-1}, \tag{3}$$

where $I(x_i, x_j)$ is an indicator function that returns 1 if $(c(x_i) = c(x_j))$, and 0 otherwise. Equation 3 reduces the number of terms in Equation 2's intractable inner sum from $|\mathcal{T}| = \binom{N}{\alpha}$ to only $N - 1$. Unless $\alpha = 1$, or $\alpha = N - 1$, this improvement is very significant.

[4] This technique is compatible with any reasonable tie-breaking criterion.

[5] Leave-one-out-cross-validation ($\alpha = N - 1$) avoids this combinatorial explosion of $|\mathcal{T}|$.

[6] When $n < r$, $\binom{n}{r} \equiv 0$.

2.2.1 Complexity analysis

The time required to calculate A_x using Equation 3 is composed of the following parts:

1. Computing distances from x to the other $N-1$ items: $O(N)$;
2. Sorting the $N-1$ items: $(N \log N)$;
3. Summing the $N-1$ terms in Equation 3: $O(N)$.

Therefore, the time required to obtain A_x is bounded by $O(N \log N)$. Since A_x is computed for each $x \in S$, the total time is:

$$O(N^2 \log N). \tag{4}$$

When a significant fraction of the data is to be placed in the training set, A_x can be computed more efficiently as follows. In Equation 3, note that when $i > N - \alpha$, the binomial coefficient $\binom{N-i-1}{\alpha-1} = 0$. So, rather than summing $N-1$ terms, we need only sum the first $N-\alpha$ terms. Also, rather than sorting all $N-1$ items, we need only determine the $N-\alpha$ nearest neighbors, which can be done with a priority queue in time $O(N \log(N-\alpha+1))$ [5]. This results in a total running time of $O(N^2 \log(N-\alpha+1))$, which can be as low as $O(N^2)$ when $N-\alpha$ is a small constant.

By comparison, a single trial of the standard algorithm takes $O(\alpha\ (N-\alpha))$.[7] The exhaustive computation (Equation 1) will therefore take $O(\binom{N}{\alpha}\alpha(N-\alpha))$, which is intractable for typical α. In fact, our algorithm is more efficient than averaging the results from $O(\log N)$ randomized trials.

2.3 Algorithm 2: ranked class-independent test/train split

The type of experiment described in Section 2.2 has a very strict criterion of correctness. In some domains with large numbers of classes, a classifier is judged to be "correct" iff at least one of the R closest items in the training set matches query item's class label. For instance, a face recognition system may be given a noisy photograph, x, and asked to return a list of the three closest images in its database. As long as the identity of one of the retrieved faces is correct, the system is judged to be correct. This results in a generalization of Algorithm 1 (which is recovered when $R = 1$). Our technique can be extended as follows.

We first define $f_i(k)$ to be the number of items from a given class, k, that are farther than a given item, x_i in S (equivalent to the number of items of the given class appearing to the right of x_i in the diagram in Section 2.1):

$$f_i(k) = |\{j : j > i \text{ and } c(x_j) = k\}|. \tag{5}$$

Let x_i be the nearest *correct* answer, but also the r-th nearest neighbor overall in the training set (i.e., there are exactly $r-1$ items in T that are nearer to x, and whose class is *not* $c(x)$). The expression for A_x in Equation 2 (after some algebra) becomes:

$$A_x = \sum_{i=1}^{N-1} I(x, x_i) \sum_{r=1}^{R} \binom{N-i-1}{\alpha-r} \binom{i - N_{c(x)} + f_i(c(x)) + 1}{r-1}. \tag{6}$$

The complexity of this algorithm may be derived by comparing Equations 3 and 6. Step 3 in Section 2.2.1 is affected as follows: (1) the single binomial coefficient has been replaced by a sum of R terms; (2) there is an additional cost of computing $\forall k\ f_i(k)$. The latter can be computed in constant time per summand since $f_{i+1}(k) = f_i(k) - 1$ when $k = c(x_{i+1})$, and is unchanged otherwise; for $N-1$ summands, this takes $O(N)$. Since step 3 can now take $O(RN)$ time, the total complexity of this algorithm is $O(N^2(R + \log N))$.

[7] A kd-tree [3] with suitable data can reduce average case expected time to $O(N \log \alpha)$.

2.4 Algorithm 3: class-constrained test/train splits

In some applications, the training set is constrained to contain a certain number of items from each class. For instance, the ORL face dataset [8] contains 10 images from each of 40 individuals. The face recognition experiments presented in [6, 9] examine accuracy by varying the size of the training set using 1, 3, or 5 images for each of the 40 classes. In general, we define α_k to be the number of items required from class k to be in the training set[8]. This section shows how our technique may be applied to this problem, by extending the algorithm described Section 2.2. Section 2.5 describes how this may be further extended to the top-R rank case.

Consider all training sets, T, where the nearest items to x in T is x_i. For each class, $k \neq c(x_i)$, we must select α_k items from a potential $f_i(k)$ candidates for inclusion into the training set T. For class $k = c(x_i)$, since we have already selected x_i, we only need an additional $\alpha_k - 1$ items from the $f_i(k)$ candidates. Since the choices for each class are independent, the number of training sets that correctly classify x is the product of the number of choices for each class. All possible training sets for item x can be generated by considering i from 1 to $N-1$, as above. In a manner analogous to Equation 3, we can efficiently calculate A_x as:

$$A_x = \sum_{i=1}^{N-1} I(x, x_i) \binom{f_i(c(x))}{\alpha_{c(x)} - 1} \prod_{\substack{k=1 \\ k \neq c(x)}}^{C} \binom{f_i(k)}{\alpha_k} \tag{7}$$

2.4.1 Complexity analysis

A similar analysis to the one presented in Section 2.2.1 can be performed to calculate the worst case bounds on running time for this algorithm. The time required to compute A_x using Equation 7 is composed of the following parts:

1. Distance computation and sorting (identical to Section 2.2.1);
2. Computing $f_i(k)$ as in Section 2.3: $O(N)$;
3. Computing the $N_{c(x)} - 1$ non-zero terms, each requiring $O(C)$ to multiply the binomial coefficients together: $O(CN_{c(x)})$.

Thus, the total time needed to compute A_x is $O(N \log N + CN_{c(x)})$. To compute A, we must do this for each $x \in S$; the total time is given by:

$$\begin{aligned} \sum_{x \in S} O(CN_{c(x)} + N \log N) &= \sum_{k=1}^{C} \sum_{c(x)=k} O(CN_k + N \log N) \\ &= O(N^2(C + \log N)). \end{aligned} \tag{8}$$

By comparison, a single trial of the standard algorithm takes $O(\alpha(N - \alpha))$, where $\alpha \equiv \sum_{k=1}^{C} \alpha_k$. Therefore, the exhaustive computation will take $O(\alpha(N - \alpha) \prod_{k=1}^{C} \binom{N_k}{\alpha_k})$, which is prohibitive. Our algorithm is also more efficient than averaging the results of $O(\log N)$ random trials for typical datasets; in the worst possible case, with pathological data and $\alpha(N - \alpha) \approx N$, running our algorithm is more advantageous than averaging $O(N^2)$ random splits.

[8] It is easy to compute α_k for experiments that require a particular fraction of items per class to be in the training set. Similarly, α_k can be computed when a certain number (or fraction) of items per class is required to be in the test set.

2.5 Algorithm 4: ranked class-constrained test/train split

The final algorithm shows how our technique can be applied to the rank-variant of the class-constrained experiment (combining Sections 2.3 and 2.4). As in Section 2.3, let x_i be the nearest correct item that is also the r-th nearest neighbor overall in the training set. We introduce the new variables, $\beta_1, \ldots, \beta_C$, where β_k is the number of items in class k that are to the left of x_i in the diagram shown in Section 2.1. We need to place β_k items into the $N_k - f_i(k)$ slots to the left of x_i; also, we must assign $\alpha_k - \beta_k$ items into $f_i(k)$ possible slots to the right of x_i. Therefore, A_x is:

$$\sum_{i=1}^{N-1} I(x, x_i) \sum_{r=1}^{R} \binom{f_i(c(x))}{\alpha_{c(x)} - 1} \underbrace{\sum_{\beta_1=0}^{r-1} \sum_{\beta_2=0}^{r-1-\beta_1} \cdots \sum_{\beta_C=0}^{r-1-\sum_{k=1}^{C-1}\beta_k}}_{\text{Omit the sum over } \beta_{c(x)}} \prod_{\substack{k=1 \\ k \neq c(x)}}^{C} \binom{N_k - f_i(k)}{\beta_k} \binom{f_i(k)}{\alpha_k - \beta_k},$$

Generating functions [4] allow us to express the inner sums over β_k as the coefficients of the following polynomial in z (details omitted for space considerations):

$$z \prod_{\substack{k=1 \\ k \neq c(x)}}^{C} \left\{ \sum_{\beta} \binom{N_k - f_i(k)}{\beta} \binom{f_i(k)}{\alpha_k - \beta} z^{\beta} \right\},$$

where the coefficient of z^s is the value of these sums at $r = s$. The expression for A_k now consists of a product of $C - 1$ polynomials, but since we are only interested in terms up to z^R, we need not evaluate the higher order terms. Each of the $C - 1$ steps involves multiplying a running product of degree $\leq R$, with a polynomial of degree $\min(R, \alpha_k)$; this can be performed in time $O(R^2)$, thus the sums can be computed in $O(CR^2)$. Comparing this with step 3 in Section 2.4.1, we see that the total complexity is $O(N^2(CR^2 + \log N))$.

3 Empirical results

Table 1 summarizes exact and estimated accuracy with 95% confidence intervals of the estimate (computed over several independent sets), and timing results[9] from a set of four experiments that compare our technique (denoted "Exact") against runs of the standard algorithm (denoted S-n, where n is the number of random trials averaged). All experiments used a nearest-neighbor classifier with Euclidian distance function and unscaled features. To avoid problems with overflow and accuracy (A can get very large), these experiments were implemented using PARI[10], an arbitrary precision math library.

Table 1: Results: Empirical accuracy (%) with 95% (2σ) intervals, and timing (seconds).

#	Exact	time	S-1	time	S-10	time	S-50	time
1	60.7	8.2	61.1±6.3	1.2	60.6±2.0	11.8	60.6±0.7	59.3
2	96.3	24.3	96.4±3.4	1.2	96.2±1.1	12.5	96.5±0.4	62.2
3	88.8	115.3	89.2±3.9	25.5	88.9±0.8	248.6	88.8±0.8	1248.2
4	94.0	153.9	93.5±3.3	25.0	94.2±1.1	246.6	93.9±0.7	1262.1

Experiments 1 and 2 used the *liver-disorders* dataset from the UCI Machine Learning Repository (345 items, each with 6 numeric values from 2 classes); items were selected

[9] Timing results (seconds) were obtained from a Pentium II 300MHz running Linux 2.2.1.

[10] PARI: <http://hasse.mathematik.tu-muenchen.de/ntsw/pari/view/>.

uniformly without replacement. Experiment 1 corresponds to Section 2.2 (Algorithm 1: $\alpha = 172$); Experiment 2 corresponds to Section 2.3 (Algorithm 2: $\alpha = 172, R = 5$). Experiments 3 and 4 used the ORL dataset [8], with images reduced to 16×16 size (400 items, each with 256 numeric values; 10 items in each of 40 classes). Experiment 3 corresponds to Section 2.4 (Algorithm 3: $\forall k\ \alpha_k = 3$). Experiment 4 corresponds to Section 2.5 (Algorithm 4: $\forall k\ \alpha_k = 3, R = 3$).

These experiments reflect the theoretical results presented above. The accuracies reported by our technique are consistent with those obtained by the standard method. Also, as expected, the variance in the estimates for the standard method drop as more trials are averaged. The timing information shows that it is advantageous to use our algorithm if, to obtain the desired level of confidence, more than 5 to 20 trials of the standard method (depending on the problem) would be required.

4 Conclusion

Nearest-neighbor methods appear frequently (in disguise) in real applications, typically as the final stage of a complex system. Our technique, which is straightforward to implement, is immediately applicable; for instance, Algorithm 3 reduced the time required to evaluate the face recognition systems described in [9] from several hours to a few minutes. This technique may also be used as a fast cross-validation component in other machine-learning algorithms. We are currently incorporating it into a gradient-descent-based method for feature weighting with encouraging preliminary results. We plan to extend our technique in two main directions: (1) directly extracting exact higher-order statistics, such as variance; (2) applying the technique to a broader set of classifiers, such as k-nearest-neighbor.

Acknowledgments

Thanks to Rich Caruana, Dayne Freitag, Terence Sim, and Gita Sukthankar for valuable feedback on this paper.

References

[1] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13, 1967.

[2] B. Dasarathy. *Nearest Neighbor Pattern Classification Techniques.* IEEE Computer Society Press, 1991.

[3] J. Friedman, J. Bentley, and R. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3), 1977.

[4] R. Graham, D. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, 1994.

[5] D. Knuth. *The Art of Computer Programming: Sorting and Searching*. Addison-Wesley, 1973.

[6] S. Lawrence, C. Giles, A. Tsoi, and A. Back. Face recognition: A hybrid neural network approach. Technical Report UMIACS-TR-96-16, University of Maryland, 1996.

[7] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[8] F. Samaria and A. Harter. Parametrisation of a stochastic model for human face identification. In *Proceedings of IEEE Workshop on Applications on Computer Vision*, 1994. ORL database is available at: `<www.cam-orl.co.uk/facedatabase.html>`.

[9] T. Sim, R. Sukthankar, M. Mullin, and S. Baluja. High-performance memory-based face recognition for visitor identification. Technical Report JPRC-TR-1999-001-1, Just Research, 1999.