Information and Inference: A Journal of the IMA (2021) **10**, 261–283
doi:10.1093/imaiai/iaaa003
Advance Access publication on 20 April 2020

An iterative method for classification of binary data ®

Denali Molitor[†] and Deanna Needell
Department of Mathematics, University of California, Los Angeles, Los Angeles, CA 90095, USA
[†]Corresponding author: dmolitor@math.ucla.edu
deanna@math.ucla.edu

[Received on 14 November 2018; revised on 19 September 2019; accepted on 10 February 2020]

In today's data-driven world, storing, processing and gleaning insights from large-scale data are major challenges. Data compression is often required in order to store large amounts of high-dimensional data, and thus, efficient inference methods for analyzing compressed data are necessary. Building on a recently designed simple framework for classification using binary data, we demonstrate that one can improve classification accuracy of this approach through iterative applications whose output serves as input to the next application. As a side consequence, we show that the original framework can be used as a data preprocessing step to improve the performance of other methods, such as support vector machines. For several simple settings, we showcase the ability to obtain theoretical guarantees for the accuracy of the iterative classification method. The simplicity of the underlying classification framework makes it amenable to theoretical analysis.

Keywords: classification; iterative framework; binary data.

1. Introduction

Interpretability of algorithms and the ability to explain predictions is of increasing importance as machine learning algorithms are applied to an expanding range of problems in areas such as medicine, criminal justice and finance [5, 6, 17]. Decisions made based on algorithmic predictions can have profound repercussions for both participating individuals as well as society at large. A major drawback to complex models such as deep neural networks [14, 24, 29, 30] is that it is extremely difficult to explain how or why such algorithms arrive at a specific prediction; see e.g. [44, 46, 47] and references therein. Studying and advancing models for which model output can be understood will help to both improve methods that are more readily interpretable and develop tools for understanding more complex models. The aim of this paper is to continue developing a framework with these two simultaneous goals in mind. This will both build a simple yet robust and analysable approach, while also building a mathematical foundation upon which state of the art methods can be studied.

To this end, we consider the problem of performing classification using only binary measurements of the data. This situation may arise due to the need for extreme compression of data or in the interest of hardware efficiency [2, 18, 27, 28]. Despite this extremely coarse quantization of the data, one can still perform learning tasks, such as classification, with high accuracy. The authors of [36] recently proposed a classification method for binary data, which they show to be reasonably accurate and sufficiently simple to allow for theoretical analysis in certain settings. Additionally, the predicted class can be approximately understood as the class whose binarized training data most closely and frequently

® Symbol indicates reproducible data.

matches that of the test point. We will extend this approach to an iterative feedback framework that yields improved classification accuracy while also exhibiting more levels of abstraction akin to simple neural networks.

1.1 *Contributions*

We propose an extension of the simple classification method for binary data proposed in [36], which we will henceforth refer to as SCB. Specifically, we propose an iterative method that uses output from SCB as input to a subsequent application of SCB. We refer to this iterative method as ISCB. We find that the iterative extension, ISCB, often leads to improved performance over SCB. Additionally, we demonstrate that SCB can be used for dimension reduction or as a data preprocessing step to improve the performance of other models, such as support vector machines (SVM). While one can draw similarities between methods such as decision tree ensembles [3, 34, 35] and approximate nearest neighbors [22, 26, 48] with SCB, we focus here on the extension of SCB to an iterative framework. Due to the simplicity of the SCB framework, we can provide theoretical guarantees for the accuracy of the iterative extension in simple two-dimensional settings.

1.2 *Organization*

The paper is organized as follows. Section 2 introduces the problem statement and classification strategies of interest. Section 2.1 describes the SCB framework introduced in [36] and Section 2.2 our proposed iterative extension. In Section 3, we demonstrate the performance of the proposed approach on real and synthetic datasets. Section 4 discusses variations and practical considerations. We provide theoretical guarantees for the proposed iterative method in several simplified settings and provide intuition as to why ISCB generally outperforms the original SCB approach in Section 5. Lastly, Section 6 comments on how SCB can be adapted to serve as a data preprocessing and dimension reduction strategy for other methods applied to binary data.

2. Classification using binary data

We first introduce the problem and notation that will be used throughout. Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ be a random measurement matrix (e.g. typically it will contain i.i.d. standard normal entries). Let $\boldsymbol{X} = [\boldsymbol{x}_1 \cdots \boldsymbol{x}_p] \in \mathbb{R}^{n\times p}$ be the matrix of p data vectors $\boldsymbol{x}_i \in \mathbb{R}^n$ with labels $\boldsymbol{b} = (b_1, \cdots b_p)$. Let G be the number of groups or classes to which the data points belong, so that we may assume $b_i \in \{1, 2, \ldots, G\}$. Suppose we have the binary measurements of the data

$$\boldsymbol{Q} = \text{sign}(\boldsymbol{AX}),$$

where $\text{sign}(\boldsymbol{M})_{i,j} = \text{sign}(M_{i,j})$ and for a real number c the sign function simply assigns $\text{sign}(c) = 1$ if $c \geqslant 0$ and -1 otherwise. For a matrix $\boldsymbol{M}$, let $\boldsymbol{M}_{(j)}$ denote the jth column of $\boldsymbol{M}$.

The rows of the matrix $\boldsymbol{A}$ can be viewed as the normal vectors to randomly oriented hyperplanes, in which case the (i, j)th entry of $\boldsymbol{Q}$ denotes on which side of the ith hyperplane the jth data point $\boldsymbol{x}_j$ lies. In practice, the binary data $\boldsymbol{Q}$ may be obtained during processing or be provided as direct input from some other source. In the latter case, we may not have access to the data matrix $\boldsymbol{X}$ or the measurement matrix $\boldsymbol{A}$, but only the resulting binary data $\boldsymbol{Q}$. We refer to the binary information indicating the position of a data point relative to a set of hyperplanes as a *sign pattern*. In particular, for a column $\boldsymbol{Q}_{(j)}$ and any subset of its entries, the resulting vector indicates the sign pattern of the jth data point relative to that subset of hyperplanes.

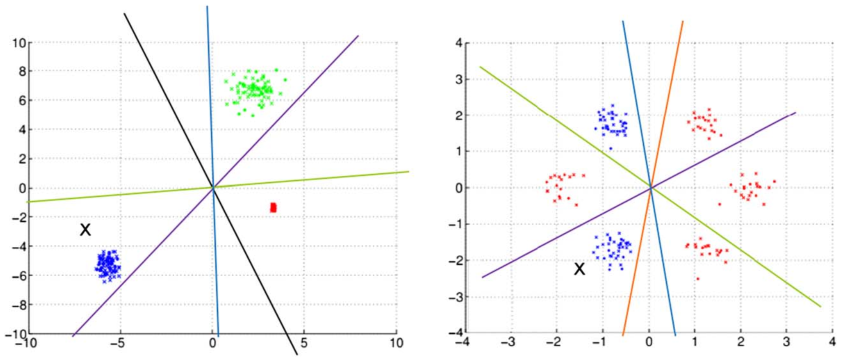

Fig. 1. A motivating example for using positions relative to hyperplanes for classification.

We aim to classify a data point $\boldsymbol{x}$ based only on the binary information contained in $\boldsymbol{Q}$. As a simple motivating example, consider the left plot of Fig. 1. The training data points each belong to one of three classes, red, blue or green. Consider the test point indicated by the black $\boldsymbol{x}$. Cycling through the hyperplanes, the green hyperplane indicates that the test point likely belongs to the blue or red class (since it lies on the same side as these clusters), the purple hyperplane indicates that the test point likely belongs to the blue or green class, the blue hyperplane indicates that the test point likely belongs to the blue class and the black hyperplane indicates that the test point likely belongs to the blue class. In aggregate, the test point matches the relative positions of the blue class to the hyperplanes most often. This prediction matches what we might predict visually.

For the data in the right plot of Fig. 1, there are both red and blue points on the same side as $\boldsymbol{x}$ for each hyperplane. However, if we consider sign patterns with respect to *pairs* of hyperplanes instead of only single hyperplanes, we can isolate data within cones or wedges as opposed to simply half-spaces. Comparing the sign patterns of the training data with respect to pairs of hyperplanes with that of the test point, we find that the test point $\boldsymbol{x}$ matches the sign patterns of the blue class most often. Thus, it may not be enough to consider hyperplanes individually, but in tuples. SCB of [36] uses this intuition as motivation.

Generating binary representations of data is a popular strategy for compressing and analysing high-dimensional data. Binary representations are commonly applied to text data and used in computer vision tasks for performing efficient search, detection and recognition [1, 12, 33, 39–43, 45]. Here, we focus not on generating binary representation of the data or specific applications, but on making classifications using binary data. While one could use more complicated methods to generate binary descriptors, in order to preserve the geometric interpretation of ISCB, we use random hyperplanes to generate the binary data as opposed to learned, data-dependent or domain-specific strategies. Similar strategies have also been considered in the context of approximate nearest neighbours search for high-dimensional data [13, 22, 26, 48]. Alternative methods for generating the binary representations could be used in the ISCB framework, for example, to exploit potential structure within the data.

2.1 *SCB*

In SCB, sign patterns of the data with respect to tuples of hyperplanes of various lengths are recorded and aggregated to arrive at a prediction. The length of the sign patterns, or the number of hyperplanes considered, is referred to as the *level*. For each level $\ell = 1, \cdots, L$, we choose m random combinations of ℓ hyperplanes. Each of the hyperplane-tuples provides a *measurement* of the data points. Fixing the number of hyperplane combinations considered, as opposed to considering all possible combinations, prevents the number of measurements from growing exponentially with the level.

Let t be a sign pattern for the ith measurement at the ℓth level and $P_{g|t}$ be the number of training points in class g with sign pattern t. This sign pattern information is then aggregated for the training data points in the membership function $\boldsymbol{r}(\ell, i, t, g)$, with

$$\boldsymbol{r}(\ell, i, t, g) := \frac{P_{g|t}}{\sum_{j=1}^{G} P_{j|t}} \frac{\sum_{j=1}^{G} \left| P_{g|t} - P_{j|t} \right|}{\sum_{j=1}^{G} P_{j|t}}, \tag{2.1}$$

where $\ell = 1, \cdots, L$, $i = 1, \cdots m$ and $g = 1, \cdots, G$. The first term in this formula gives the fraction of points with sign pattern t that belong to class g, while the second acts as a balancing term to account for differences in the relative sizes of different classes. Each value in this membership function gives an indication of how likely a data point is to belong to class g based on the fact that it has sign pattern t for the ith measurement at the ℓth level. Larger $\boldsymbol{r}(\ell, i, t, g)$ values indicate that a data point is more likely to belong to the gth class. Training is detailed in Algorithm 1, which simply consists of computing all of the $\boldsymbol{r}(\ell, i, t, g)$ quantities.

Given a test point $\boldsymbol{x}$, with binary data $\boldsymbol{q} = \text{sign}(\boldsymbol{A}\boldsymbol{x})$, for each level ℓ, measurement i and associated sign pattern t^* we find the corresponding $\boldsymbol{r}(\ell, i, t^*, g)$ value and keep a running sum for each group g, stored in the vector $\widetilde{\boldsymbol{r}}$ (note that the vector $\widetilde{\boldsymbol{r}}$ depends on the data point $\boldsymbol{x}$, but we notationally ignore this dependence for tidiness, and will write $\widetilde{\boldsymbol{r}}(g)$ for a class g or data point $\boldsymbol{y}$ when clarification is needed). If t^* does not match any of the sign patterns observed in the training data, then no update to $\widetilde{\boldsymbol{r}}$ is made. The testing procedure is detailed in Algorithm 2. In [36], the authors showed that this classification method works well on both artificial and real datasets (e.g. MNIST [31], YaleB [9–11, 25]).

Algorithm 1 SCB training from [36]

```
1. **Input:** binary training data $\boldsymbol{Q}$, training labels $\boldsymbol{b}$, number of classes $G$, number of levels $L$.
2. for $\ell$ from 1 to $L$, $i$ from 1 to $m$ do
3.     Randomly select $\ell$ hyperplanes.
4.     for all observed sign patterns $t$ and classes $g$ from 1 to $G$ do
5.         Compute $\boldsymbol{r}(\ell, i, t, g)$ as in (2.1).
6.     end for
7. end for
```

2.2 *ISCB*

We now introduce a novel iterative extension to SCB, which we refer to as ISCB. First, we motivate the extension through an example. Consider Fig. 2, which plots the values of $\widetilde{\boldsymbol{r}}$ from Algorithm 2 for the task of classifying the digits 0–4 of the MNIST dataset (where we will use class labels $0, 1, \ldots, 4$). Note that test images of the digit 0 typically have lower $\widetilde{\boldsymbol{r}}(1)$ values than do other digits. Similarly, test images of the digit 1 typically have lower $\widetilde{\boldsymbol{r}}(0)$ than do test images of the digits 1–4. Indeed, it is not only likely that

$$\hat{b}_{\boldsymbol{x}} = argmax_{g \in \{1, \cdots, G\}} \widetilde{\boldsymbol{r}}(g)$$

Algorithm 2 SCB classification from [36]

```
1.  Input: binary testing data $\boldsymbol{q}$, number of classes $G$, number of levels $L$, learned parameters
    $\boldsymbol{r}(\ell, i, t, g)$ and hyperplane tuples from Algorithm 1.
2.  Initialize $\widetilde{\boldsymbol{r}}(g) = 0$ for $g = 1, \cdots, G$.
3.  for $\ell$ from 1 to $L$, $i$ from 1 to $m$ do
4.      Identify the sign pattern $t^*$ to which $\boldsymbol{q}$ corresponds for the $i$th $\ell$-tuple of hyperplanes.
5.      for $g$ from 1 to $G$ do
6.          $\widetilde{\boldsymbol{r}}(g) = \widetilde{\boldsymbol{r}}(g) + \boldsymbol{r}(\ell, i, t^*, g)$.
7.      end for
8.  end for
9.  Set $\widetilde{\boldsymbol{r}}(g) = \frac{\widetilde{\boldsymbol{r}}(g)}{Lm}$ for $g = 1, \cdots, G$.
10. Classify $\hat{b} = argmax_{g\in\{1,\cdots,G\}}\widetilde{\boldsymbol{r}}(g)$.
```

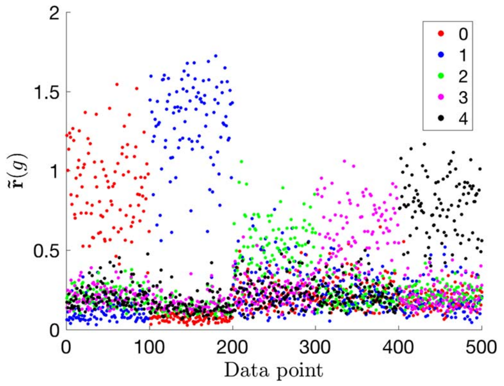

FIG. 2. The $\widetilde{\boldsymbol{r}}(g)$ values from SCB trained to classify digits 0–4 from the MNIST dataset are plotted. Five digits are considered to ease visualization. One-hundred test points from each digit are used with points 1–100 corresponding to 0s, 101–200 corresponding to 1s, etc. $\widetilde{\boldsymbol{r}}(0)$ values are plotted in red, $\widetilde{\boldsymbol{r}}(1)$ in blue, $\widetilde{\boldsymbol{r}}(2)$ in green, $\widetilde{\boldsymbol{r}}(3)$ in magenta and $\widetilde{\boldsymbol{r}}(4)$ in black.

corresponds to the true digit label, but in addition the $\widetilde{\boldsymbol{r}}$ vectors for testing images from different digits contain different *patterns*. Thus, we expect that using a method more advanced than simply predicting the class corresponding to the maximum of the $\widetilde{\boldsymbol{r}}$ vector may improve classification accuracy, specifically a strategy that makes use of the distribution of the values contained in $\widetilde{\boldsymbol{r}}$.

One could make predictions based on the $\widetilde{\boldsymbol{r}}$ vectors in a variety of ways. We mention a few such options here. Drawing intuition from simple neural network architectures such as multilayer perceptron [16] and boosting algorithms such as AdaBoost [19, 20], we first consider using iterative applications of SCB, where $\widetilde{\boldsymbol{r}}$ values of the training data from previous iterations are used as input training data for the following iteration. In particular, this strategy is reminiscent of the structure of a single neuron in a neural network in which information only propagates forward as opposed to throughout the whole network. In contrast to deep neural networks, the output at each iteration, $\widetilde{\boldsymbol{r}}$, can be interpreted as a vector indicating to which class a data point $\boldsymbol{x}$ is likely to belong. This iterative strategy also relates to boosting in that subsequent iterations train on the shortcomings of previous iterations. Specifically, if points from a given class are misclassified, but produce similarly structured $\widetilde{\boldsymbol{r}}$ vectors, this pattern may be corrected in the next iteration.

The training and testing phases of the proposed ISCB method are detailed in Algorithms 3 and 4. To ease notation, denote $\boldsymbol{r}_k$, $\widetilde{\boldsymbol{r}}_k$ and $\boldsymbol{A}^{(k)}$ as $\boldsymbol{r}$, $\widetilde{\boldsymbol{r}}$ and $\boldsymbol{A}$ from the kth application of SCB (Algorithms 1 and 2). During training, the first iteration in ISCB is executed as in Algorithm 1. We collect the data $\boldsymbol{X} = [\widetilde{\boldsymbol{r}}_1(\boldsymbol{x}_1)\cdots\widetilde{\boldsymbol{r}}_1(\boldsymbol{x}_p)] \in \mathbb{R}^{G\times p}$, which will be used as training data for the next iteration, where $\boldsymbol{x}_i$ are training data points. In contrast to SCB, the iterative algorithm calculates $\widetilde{\boldsymbol{r}}$ values for both the training and test data. Note that the dimension of the data points is fixed at G after the first application of SCB. For high-dimensional data, we will typically have $G \ll n$. This reduction in dimension reduces the computational cost of some of the required computations, such as $\boldsymbol{Q} = \operatorname{sign}(\boldsymbol{AX})$. One could also make use of the same measurement matrix $\boldsymbol{A}$ for all iterations after the first. Since the dimension is much smaller after the first iteration of SCB, one may also need fewer levels for accurate classification. We leave an exhaustive study of the many possible variations for future work, and focus here on establishing the mathematical framework of this iterative approach.

After each application of Algorithm 1, we collect sign information of our data with respect to a new set of random hyperplanes. Although the dimension of the data for the subsequent applications lies in $\mathbb{R}^G$ and thus we expect the size of this data to be manageable, there are several motivations for taking binary measurements of the data at each application. First, we can still take advantage of methods for efficient storage of and computation with binary data. Second, the binary measurements roughly preserve angular information about the data. For the $\widetilde{\boldsymbol{r}}$ values, we are generally interested in the relative sizes of the components, since these represent the likelihood that a point belongs to a given class. The overall magnitude of the $\widetilde{\boldsymbol{r}}$ values is of less importance and, thus, binary measurements retain the significant information pertaining to the data. Third, considering binary measurements of the data at each application maintains consistency between the applications. Specifically, the inputs to each application are binary measurements of the data and SCB thus applies a similar transformation to the data at each iteration. This makes the iterative method more interpretable, amenable to theoretical analysis, and more in line with the layered structure of sophisticated deep neural net architectures.

Since the components of $\widetilde{\boldsymbol{r}}$ are always non-negative, we restrict the random hyperplanes to intersect this space after the first application. For example, we can ensure the hyperplanes intersect this region by requiring that the normal vectors have at least one positive and one negative coordinate. We do not recenter the data after each application, as the structure of $\widetilde{\boldsymbol{r}}$ can lead to poor performance with recentering. For an example, consider Fig. 3, in which the $\widetilde{\boldsymbol{r}}$ values follow a roughly linear trend for later applications of the method.

For testing, we calculate $\widetilde{\boldsymbol{r}}_k(g)$ values for each iteration k by aggregating the $\boldsymbol{r}(\ell, i, t^*, g)$ values for each sign pattern t^* of the test point. Finally, at the last iteration K, we make the prediction

$$\hat{b} = \underset{g\in\{1,\cdots,G\}}{\operatorname{argmax}}\, \widetilde{\boldsymbol{r}}_K(g).$$

We pause to make a few remarks regarding ISCB and its relation to SCB and neural networks. For both SCB and ISCB, one must specify the number of levels used. The level indicates the maximal size of random subsets of hyperplanes used for matching sign patterns and determining the $\boldsymbol{r}_k$ parameters.

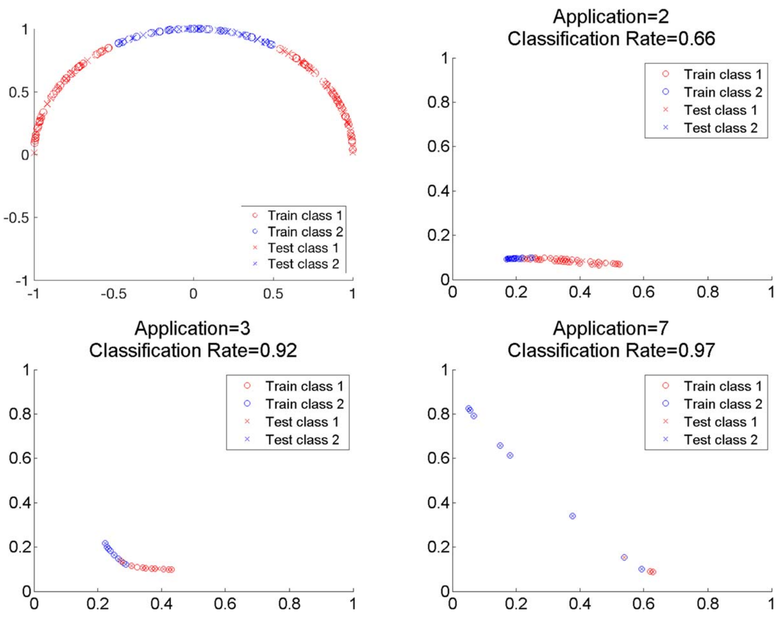

FIG. 3. The $\widetilde{\boldsymbol{r}}_k$ vectors are plotted for ISCB with varying numbers of iterations k. The original training and testing data are shown in the upper left plot. Circles indicate training data and crosses indicate testing data. One level is used for each application of SCB and the subsequent plots give the $\widetilde{\boldsymbol{r}}_k$ vectors for $k = 1, 3$ and 7 respectively.

In this way, the level indicates the granularity at which the sign patterns are compared and must match in order to contribute to the learned $\boldsymbol{r}_k$ parameters. We will see that the performance generally improves when using a larger number of levels. Using more levels, however, requires more computation and storage and the performance gain typically levels off. For ISCB, one must also indicate the number of applications to consider. The $k + 1^{\text{st}}$ application uses the $\tilde{\boldsymbol{r}}_k$ values from the kth application. With each application, data points that were misclassified at the previous application have the opportunity to be reclassified in the next application. As we will see in the experiments, using more than one application of SCB typically leads to improved performance, but this improvement in accuracy levels off after only a few applications.

While we draw inspiration from the multilayer structure of neural networks and one can draw several similarities between the two, the models differ in significant ways. When binarizing the data $\boldsymbol{X}$ in each application of SCB via $\boldsymbol{Q} = \operatorname{sign}(\boldsymbol{AX})$, a linear transformation is applied to the data followed by the nonlinear sign function. This step is followed by an aggregation of information through the calculations of the parameters $\boldsymbol{r}_k$ via (2.1). In contrast to neural networks, however, the parameters $\boldsymbol{r}_k$ for each application k of SCB are not learned via a gradient descent variant or in order to minimize a specific loss function. Instead, the learned parameters are determined via (2.1) by using counts of the training data points with specific sign patterns. Another major difference between ISCB and neural networks is the usage of a fixed and random linear transformation $\boldsymbol{A}$, whose rows correspond to random hyperplanes as opposed to optimized, learned weights as is typical for neural networks. In summary, while ISCB is inspired by some aspects of neural networks, we do not claim the methods are in the same family of approaches. Indeed, SCB and

ISCB are rooted in the natural geometric interpretation demonstrated in Fig. 1 and subsequently well suited to analysis.

Algorithm 3 ISCB training

```
1.  Input: binary training data $\boldsymbol{Q} \in \mathbb{R}^{m\times p}$, training labels $\boldsymbol{b}$, number of classes $G$, number of levels $L$, number of applications $K$.
2.  for $k$ from 1 to $K$, do
3.      Train learned parameters $\boldsymbol{r}_k(\ell, i, t, g)$ as in Algorithm 1, with input: $\boldsymbol{Q}$, $\boldsymbol{b}$, $G$ and $L$.
4.      Set $\boldsymbol{X} = 0 \in \mathbb{R}^{G\times p}$.
5.      for $j$ from 1 to $p$ do
6.          Apply Algorithm 2 to $\boldsymbol{Q}_{(j)}$ using learned parameters $\boldsymbol{r}_k(\ell, i, t, g)$ to calculate $\widetilde{\boldsymbol{r}}_k$.
7.          Set $\boldsymbol{X}_{(j)} = \widetilde{\boldsymbol{r}}_k$.
8.      end for
9.      Form the random measurement matrix $\boldsymbol{A}^{(k)} \in \mathbb{R}^{m\times G}$.
10.     Set $\boldsymbol{Q} = \text{sign}(\boldsymbol{A}^{(k)}\boldsymbol{X})$.
11. end for
12. Output:$\boldsymbol{r}_k(\ell, i, t, g)$, and $\boldsymbol{A}^{(k)}$ for $k$ from 1 to $K$.
```

Algorithm 4 ISCB testing

```
1.  Input: binary test data $\boldsymbol{q} \in \mathbb{R}^m$, number of classes $G$, levels $L$ and iterations $K$, learned parameters $\boldsymbol{r}_k(\ell, i, t, g)$, hyperplane tuples and $\boldsymbol{A}^{(k)}$ from Algorithm 3.
2.  for $k$ from 1 to $K$ do
3.      Set $\widetilde{\boldsymbol{r}}_k = 0$.
4.      for $\ell$ from 1 to $L$, $i$ from 1 to $m$, do
5.          Identify the pattern $t^*$ to which $\boldsymbol{q}$ corresponds for the $i$th $\ell$-tuple of hyperplanes.
6.          for $g$ from 1 to $G$ do
7.              Update $\widetilde{\boldsymbol{r}}_k(g) = \widetilde{\boldsymbol{r}}_k(g) + \boldsymbol{r}_k(\ell, i, t^*, g)$.
8.          end for
9.      end for
10.     Set $\boldsymbol{q} = \text{sign}(\boldsymbol{A}^{(k)}\widetilde{\boldsymbol{r}}_k)$.
11. end for
12. Classify $b = argmax_{g\in\{1,\cdots,G\}}\widetilde{\boldsymbol{r}}_K(g)$.
```

TABLE 1 *Flops required for inference via a single application of SCB*

Flops	Operation
$2m(d-1)$	Calculate $q = \text{sign}(\boldsymbol{A}x)$
G	Initialize $\tilde{r}$
$m\sum_{\ell=1}^{L}\lvert T_{\ell,i}\rvert\ell$ bit-wise comparisons	Identify the sign pattern (worst case)
mGL	Update $\tilde{r}(g)$ for each class and level
$G+1$	Scale
G	Predict $\hat{b}_x = argmax_{g\in\{1,\cdots,G\}}\tilde{r}(g)$

2.3 *Computational complexity*

Given a test data point $x \in \mathbb{R}^d$, inference via a single application of SCB (see Algorithm 2) requires approximately

$$m\left(\sum_{l=1}^{L}\lvert T_{\ell,i}\rvert + GL + 2d - 1\right) + 3G + 1 \tag{2.2}$$

flops. Flop counts for each step in Algorithm 2 are given in Table 1. Identifying the sign pattern typically dominates the testing cost of SCB. This step requires comparing a maximum of 2^ℓ binary vectors of length ℓ for the ℓth level. Fortunately, these binary comparisons can be implemented efficiently when the number of levels is reasonably small (for example less than 32 bits). For the data considered here, we find that choosing the number of levels to be between 10 and 20 is sufficient. For longer binary vectors, however, one could incorporate fast binary search strategies used in computer vision applications [23, 37, 38]. Additionally, typically fewer than 2^ℓ sign patterns actually occur within the training data. In (2.2) we assume that testing whether two binary strings of length ℓ match requires a single flop. The total cost for ISCB is roughly the sum of the costs of SCB at each iteration where the dimension of the data d changes to the number of classes G after the first iteration. Since ISCB performs SCB at each iteration, one could potentially reduce the total cost of ISCB by using fewer levels for some iterations.

3. Experimental results

We test ISCB on synthetic and image datasets. The synthetic datasets demonstrate why the iterative method is effective for certain simple settings and how the data transforms between iterations. ISCB is also tested on the MNIST dataset of handwritten digits [31], the YaleB dataset for facial recognition [9–11, 25] and the NORB dataset for classification of images of various toys [32].

3.1 *Two-dimensional synthetic data*

We further motivate ISCB through examples with two-dimensional synthetic data. For two-dimensional data with two classes, the dimension of the input data for all applications of SCB is two-dimensional and so we can easily visualize the effect of the iterations on both the training and testing data. We find that the more easily discerned data points are pushed out towards the boundary of the positive quadrant, while data points that are closer to the class boundary linger closer to the line $y = x$ in subsequent iterations and thus have a higher likelihood of being predicted as the other class at the next iteration.

Consider the data given in the upper left plot of Fig. 3. There are two times as many points from the red class considered, both in the training and testing set. Half of the red points in testing and training

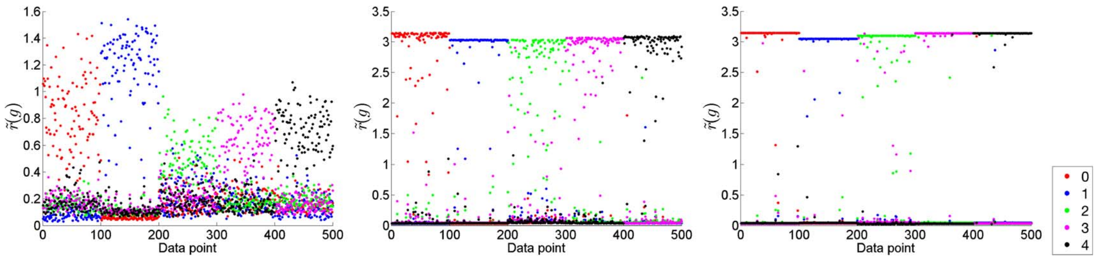

FIG. 4. The $\widetilde{\boldsymbol{r}}_k(g)$ values for testing data from ISCB trained to classify digits 0–4 from the MNIST dataset are plotted for application $k = 1, 3, 5$ from left to right. Five digits are considered to ease visualization. One hundred testing points from each digit are used with points 1–100 corresponding to 0s, 101–200 corresponding to 1s, etc. $\widetilde{\boldsymbol{r}}_k(0)$ values are plotted in red, $\widetilde{\boldsymbol{r}}_k(1)$ in blue, $\widetilde{\boldsymbol{r}}_k(2)$ in green, $\widetilde{\boldsymbol{r}}_k(3)$ in magenta and $\widetilde{\boldsymbol{r}}_k(4)$ in black.

lie on either side of the blue data points. Thus, applying SCB using a single level leads to all of the blue test points being misclassified as red. The abhorrent misclassification of the blue points is caused by the fact that we use only a single level ($L = 1$) and since for any hyperplane at least as many red points as blue lie on either side of it. The $\widetilde{\boldsymbol{r}}_1$ values, plotted in the upper right plot of Fig. 3, have a much nicer distribution in terms of ease of classification; in fact, they are nearly linearly separable. The separation in the $\widetilde{\boldsymbol{r}}_1$ values between the blue and red points occurs since $\widetilde{\boldsymbol{r}}_1$(red) is generally larger for red points than for the misclassified blue points. That is, the points that truly belong to the red class are more 'confidently' classified as red than are the blue points. If we consider the $\widetilde{\boldsymbol{r}}_1$ values as data, applying SCB now classifies the data with much higher accuracy (92% as compared to 66% for the original training data), while still only using a single level. By the seventh iterative application of SCB, the accuracy increases to 97%. If we perform the same experiment, but include a higher density of blue points so that the total number of red and blue points are the same, we achieve higher accuracy at the first application of SCB, but again see improved accuracy for later iterations.

We find that the $\widetilde{\boldsymbol{r}}_k(g)$ values for different classes spread out with additional applications of SCB for more sophisticated data as well. Recall Fig. 2, which plots the $\widetilde{\boldsymbol{r}}(g)$ values from SCB trained to classify digits 0–4 from the MNIST dataset. We can visualize the separation between the data points from different classes at later applications through similar plots. Figure 4 shows $\widetilde{\boldsymbol{r}}_k(g)$ values of testing data with number of applications k of ISCB as $k = 1, 3$ and 5. As the number of applications k increases, the $\widetilde{\boldsymbol{r}}_k(g)$ values for a testing point in class c generally approach $\widetilde{\boldsymbol{r}}_k(g) = 0$ for $g \neq c$, while $\widetilde{\boldsymbol{r}}_k(c)$ approaches a positive value, leading to separation of data from different classes. After a few applications k, the $\widetilde{\boldsymbol{r}}_k(g)$ values are pushed close to the boundary of $\mathbb{R}^G_+$, the space of vectors of length G with non-negative entries.

3.2 *Image datasets*

We test ISCB on the MNIST dataset of handwritten digits [31], the YaleB dataset for facial recognition [9–11, 25] and the NORB dataset for classification of images of various toys [32]. Results are shown in Fig. 5. We generally find both that increasing the number of levels used in each SCB application of ISCB and increasing the number of applications leads to improved performance. The classification accuracies typically level off after only a few applications of SCB, with the largest improvement typically occurring between the first and second application. These trends are less clear in the YaleB dataset, but this may be in part due to the limited amount of training data available for this dataset. For comparison, a

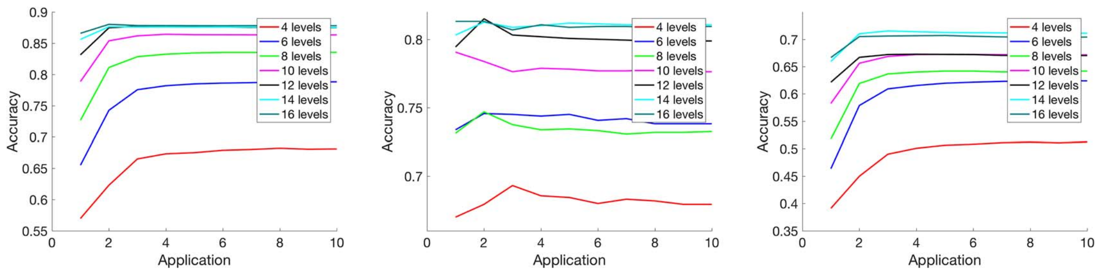

FIG. 5. Accuracies for classifying MNIST data among 10 classes (left plot), YaleB data among eight classes (middle plot) and NORB data among five classes (right plot) are given in terms of the number of applications of SCB used. For the MNIST dataset, the model is trained with $p = 1000$ images of each class and tested on 100 images from each class. The model for the YaleB dataset is trained using $p = 40$ training images from each class and applied to 20 test images from each class. For the NORB dataset, the model is trained on $p = 1000$ training images and is applied to 200 test images for each class. In each model, $m = 500$ measurements are used. Results are averaged over 10 trials.

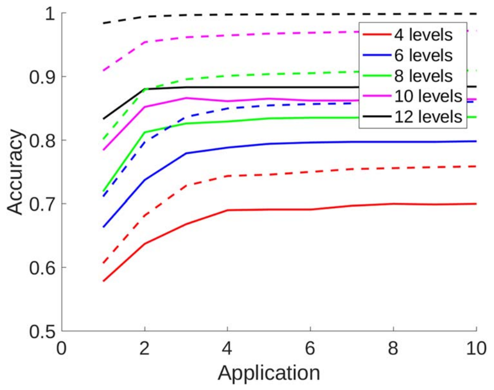

FIG. 6. Accuracies for classifying MNIST data among 10 classes are given in terms of the number of applications of SCB used for both training (dashed) and testing data (solid). The models are trained with $p = 1000$ images of each class, are tested on 100 images from each class and use $m = 500$ measurements. Results are averaged over 10 trials.

multilayer perceptron with a single hidden layer of 20 nodes, ReLU activations and trained via stochastic gradient descent to a tolerance of 10^{-4} achieves an accuracy of 91.77%. With two hidden layers of 20 nodes, it achieves 92.21% for the same set of MNIST data used in Fig. 5 for ISCB. As is discussed in Section 2.2, there are significant differences between ISCB and neural networks, making a direct comparison unnatural.

The accuracy of ISCB applied to the training data as opposed to the testing data achieves higher accuracies as expected (see Fig. 6). Increasing the number of levels and applications of SCB improves the accuracy of the model for both training and testing data. The model performance applied to the training data is still limited by the compression via the random hyperplanes.

4. Alternative iterative method

We remark here briefly about our choice in defining $\widetilde{\boldsymbol{r}}_k(g)$ and mention a natural alternative. We motivated ISCB by noting that the $\widetilde{\boldsymbol{r}}$ vectors from Algorithm 2 for test data from the same classes share

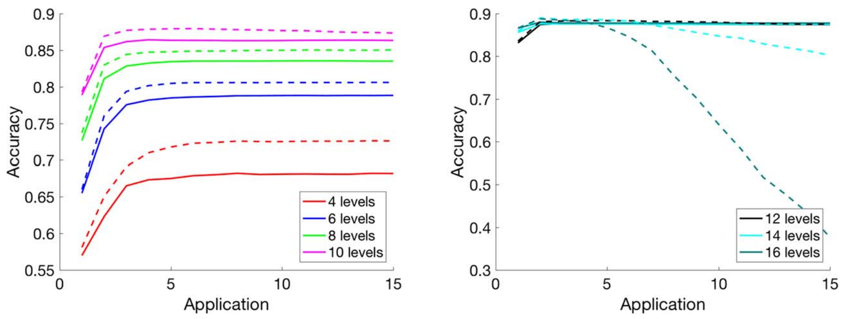

FIG. 7. The performance of ISCB using $\widetilde{\boldsymbol{r}}$, presented in Section 2.2, (solid) is compared with that of the alternative version of ISCB using $\hat{\boldsymbol{r}}$, presented in Section 4, (dashed) on the MNIST dataset. $p = 1000$ training and 100 testing images are used for each digit. Each method uses $m = 500$ binary measurements of the data at each application of Algorithm 1. The number of levels L used with each method is indicated in the legend.

similar structures. We additionally find that the contributions to the $\widetilde{\boldsymbol{r}}$ values coming from different *levels* admit different patterns as well. We could thus choose to use

$$\hat{\boldsymbol{r}}_k(\ell, g) = \sum_{i=1}^{m} \sum_{t^*} \boldsymbol{r}_k(\ell, i, t^*, g)$$

as data for the kth application of SCB instead of $\widetilde{\boldsymbol{r}}_k(g)$ as is done in Algorithm 3. Here, t^* ranges over all observed sign patterns for the ith ℓ-tuple of hyperplanes. We refer to this method as *ISCB with* $\hat{\boldsymbol{r}}$. Note that we have the following relation between $\widetilde{\boldsymbol{r}}_k$ and $\hat{\boldsymbol{r}}_k$:

$$\widetilde{\boldsymbol{r}}_k(g) = \sum_{\ell=1}^{L} \hat{\boldsymbol{r}}_k(\ell, g).$$

After the first application of SCB, the dimension of the data for ISCB with $\hat{\boldsymbol{r}}$ is then $\mathbb{R}^{LG}$.

In certain settings, ISCB with $\hat{\boldsymbol{r}}$ performs better than ISCB of Section 2.2. Typically, using $\hat{\boldsymbol{r}}_k(\ell, g)$ as opposed to $\widetilde{\boldsymbol{r}}_k(g)$ as input to the subsequent applications of Algorithm 1 performs better when the number of levels L used is small. Unfortunately, for higher numbers of levels L we see drastic declines in performance for later applications when using $\hat{\boldsymbol{r}}_k(\ell, g)$, as this method is more prone to overfit. These trends are illustrated in Fig. 7 for the MNIST dataset. In the left plot of Fig. 7, ISCB with $\hat{\boldsymbol{r}}$ leads to improved performance over ISCB with $\widetilde{\boldsymbol{r}}$. As the number of levels L used increases from 4 to 10, however, this difference diminishes. For greater than 14 levels, using ISCB with $\hat{\boldsymbol{r}}_k(\ell, g)$ leads to decreasing performance in the number of applications of SCB (seen in the right plot of Fig. 7). The same decrease in performance does not occur when using the $\widetilde{\boldsymbol{r}}_k(g)$ values as data for the next iteration, suggesting this choice may be more robust.

5. Theoretical analysis

We next offer some simple theoretical insights demonstrating why we expect performance to improve through multiple applications of SCB. We consider several scenarios that are simple, yet highlight the intuition behind the approach. At a high level, the iterative framework has the opportunity to train on the output from previous iterations and correct misclassifications that occur when the model from the previous iteration is applied to the training data. Qualitatively, as the number of iterations increases, we find that the data points that are more easily identifiable as belonging to a single class are pushed towards extreme points of the range of outputs, while data points that are more difficult to classify fall in the interior of the range and have the chance to be classified correctly at the next iteration.

5.1 *Binary classification of point masses*

As a simple but illustrative example, consider a classification task between two classes, where the training and testing data for each class is concentrated at a single point, i.e. a point mass. We consider only a single level L and let j be the number of hyperplanes that separate the two point masses in the first application of SCB. In expectation, $\frac{j}{m}$ gives an indication of the angle separating the two point masses, where m is the number of rows in the measurement matrix. Let A_1 be the number of points in class 1 and A_2 be the number of points in class 2. For testing data in class 1,

$$\widetilde{\boldsymbol{r}}_1(1) = \sum_{i=1}^{m} \boldsymbol{r}(\ell, i, t^*, 1) = j + (m-j)\frac{A_1|A_1 - A_2|}{(A_1 + A_2)^2} \text{and}$$

$$\widetilde{\boldsymbol{r}}_2(2) = \sum_{i=1}^{m} \boldsymbol{r}(\ell, i, t^*, 2) = (m-j)\frac{A_2|A_1 - A_2|}{(A_1 + A_2)^2}.$$

For testing data in class 2,

$$\widetilde{\boldsymbol{r}}_1(1) = \sum_{i=1}^{m} \boldsymbol{r}(\ell, i, t^*, 1) = (m-j)\frac{A_1|A_1 - A_2|}{(A_1 + A_2)^2} \text{and}$$

$$\widetilde{\boldsymbol{r}}_2(2) = \sum_{i=1}^{m} \boldsymbol{r}(\ell, i, t^*, 2) = j + (m-j)\frac{A_2|A_1 - A_2|}{(A_1 + A_2)^2}.$$

Note that the data for the second application of the method are again two-dimensional. Let $\widetilde{\boldsymbol{g}}_1$ be the $\widetilde{\boldsymbol{r}}_1$ vector for data points in class 1 and $\widetilde{\boldsymbol{g}}_2$ be the $\widetilde{\boldsymbol{r}}_1$ vector for data points in class 2. The following formula gives the angle θ between the two point masses at the second application,

$$\theta = \cos^{-1}\left(\frac{\langle \widetilde{\boldsymbol{g}}_1, \widetilde{\boldsymbol{g}}_2 \rangle}{||\widetilde{\boldsymbol{g}}_1||_2 \cdot ||\widetilde{\boldsymbol{g}}_2||_2}\right). \tag{5.1}$$

Figure 8 shows the angle that separates the point masses of the training data at the second application in terms of $\frac{j}{m}$ for various ratios $c = \frac{A_1}{A_2}$. We find that if A_1 and A_2 are similar in size, then the expected angle separating the two point masses increases for the second application, making the point masses 'easier' to separate in later applications.

In particular, if the two classes contain the same number of points, i.e. $A_1 = A_2$, and at least one hyperplane separates the two point masses initially, then at the next iteration, the angle between data points of classes 1 and 2 is $\pi/2$ (the best possible). Since the data are two-dimensional and we restrict the hyperplanes to intersect the positive quadrant after the first SCB application, then if the model classifies

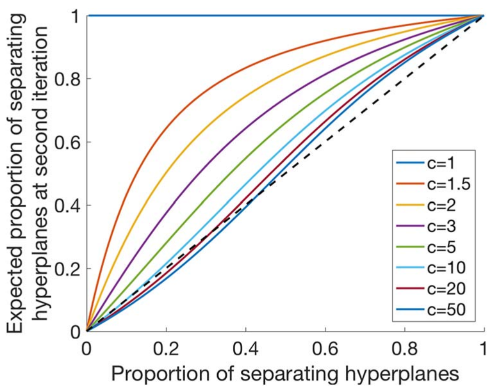

FIG. 8. This plot shows the expected proportion of hyperplanes that separate data at the second iteration of ISCB given the fraction of separating hyperplanes at the first application of SCB. The relative sizes of the two classes, given by A_1 and A_2, are varied as well, as is indicated by the parameter $c = \frac{A_1}{A_2}$ given in the legend.

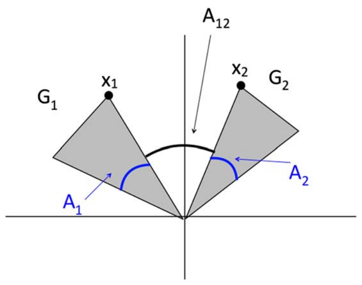

FIG. 9. Illustration of data setup for Section 5.2.

the point masses correctly at the first iteration, it will classify correctly at all subsequent iterations as well.

5.2 *Probabilistic bounds for an angular model*

We next consider an analogue to Theorem 2 of [36], in which the authors provided a lower bound on the probability that a data point will be classified correctly for data points of two different classes that reside in separated angular wedges. Consider two-dimensional data with two classes. Suppose that the data from each class is distributed within the disjoint wedges, G_1 and G_2, with angles A_1 and A_2 respectively. This setup is illustrated in Fig. 9. Consider the data points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, which lie on the inside edge of each wedge. Let A_{12} be the angle between these two points. We aim to find a lower bound on the angle between the $\widetilde{\boldsymbol{r}}_1$ vectors for $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ after a single application of SCB with a single level L. Again, since we only use a single level, $\hat{\boldsymbol{r}}_1 = \widetilde{\boldsymbol{r}}_1$ for all points $\boldsymbol{x}$.

Theorem 2 of [36] shows that a larger separating angle A_{12} between the classes leads to a higher probability of correct classification for a data point from one of the classes. A lower bound on the angle between the different classes after an application of SCB then indicates the probability of correct classification at the next application, through Theorem 2 of [36].

TABLE 2 *Contributions to the membership index parameter* $\boldsymbol{r}$ *for the point* $\boldsymbol{x}_1$ *and for hyperplanes of various types. The variables u and u′ are i.i.d. random variables uniformly distributed between zero and one, indicating the angle at which random hyperplanes intersect the wedges* G_1 *and* G_2*. The angles* A_1 *and* A_2*, wedges* G_1 *and* G_2 *and data points* $\boldsymbol{x}_1$ *and* $\boldsymbol{x}_2$ *are as shown in Fig. 9*

Hyperplane case	Number in event	Class	Value of $\boldsymbol{r}(1,i,t,g)$
Separates $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$	j	1	1
		2	0
Does not separate $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$	$m-j-k_1-k_2$	1	$\frac{A_1\lvert A_1-A_2\rvert}{(A_1+A_2)^2}$
or intersect G_1 or G_2		2	$\frac{A_2\lvert A_1-A_2\rvert}{(A_1+A_2)^2}$
Intersects G_2	k_2	1	$\frac{A_1\lvert A_1-A_2u'\rvert}{(A_1+A_2u')^2}$
		2	$\frac{A_2u'\lvert A_1-A_2u'\rvert}{(A_1+A_2u')^2}$
Intersects G_1	k_1	1	$\frac{A_1u\lvert A_1u-A_2\rvert}{(A_1u+A_2)^2}$
		2	$\frac{A_2\lvert A_1u-A_2\rvert}{(A_1u+A_2)^2}$

Assume that the data is distributed with uniformly random angles within G_1 and G_2. Let k_1 and k_2 be the number of hyperplanes that intersect G_1 and G_2, respectively, and let j be the number of hyperplanes that separate G_1 and G_2. Note that

$$\mathbb{E}k_1 = \frac{A_1}{\pi}, \quad \mathbb{E}j = \frac{A_{12}}{\pi} \quad \text{and} \quad \mathbb{E}k_2 = \frac{A_2}{\pi}.$$

Assume that the hyperplanes are also distributed with uniformly random angles within these wedges. We can then replace $P_{g|t}$ with angular measures, specifically, A_iu_h, where $u_h \in [0,1]$ and depends on the angle at which the hyperplane h intersects G_i. Since the hyperplanes are uniformly distributed at random within each region, the u_h are uniform random variables between zero and one.

The contribution to the membership index parameter $\boldsymbol{r}$ for SCB with a single level L and for each possible type of hyperplane in this setup are summarized in Table 2 for the point $\boldsymbol{x}_1$. To simplify calculations, assume that $A_1 = A_2$. With this assumption, the membership index parameters no longer depend on A_1 or A_2. Summing over all hyperplanes, for $\boldsymbol{x}_1$ we have

$$\widetilde{\boldsymbol{r}}_1(1) = \sum_{i=1}^{m} \boldsymbol{r}(1,i,t_i^*,1) = j + \sum_{h=1}^{k_1} \frac{u_h(1-u_h)}{(u_h+1)^2} + \sum_{h=1}^{k_2} \frac{1-u'_h}{(1+u'_h)^2}$$

and

$$\widetilde{\boldsymbol{r}}_1(2) = \sum_{i=1}^{m} \boldsymbol{r}(1,i,t_i^*,2) = \sum_{h=1}^{k_1} \frac{1-u_h}{(u_h+1)^2} + \sum_{h=1}^{k_2} \frac{u'_h(1+u'_h)}{(1+u'_h)^2}.$$

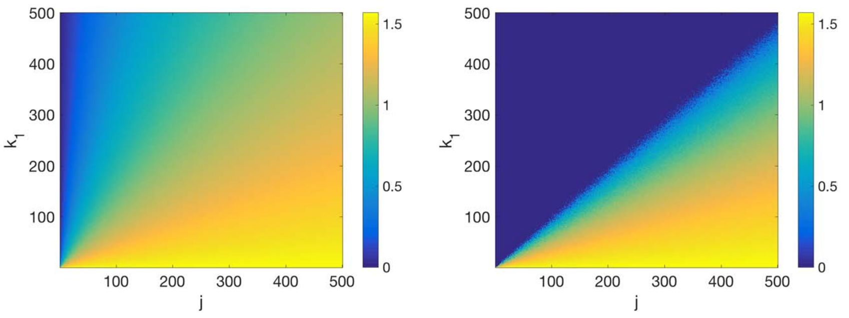

FIG. 10. For various values of $k_1 = k_2$ (the number of hyperplanes intersecting the wedges G_1 and G_2, respectively) and j (the number of hyperplanes separating the wedges G_1 and G_2), the left plot indicates the true angle (in radians) between $\widetilde{\boldsymbol{g}}_1$ and $\widetilde{\boldsymbol{g}}_2$ as given in (5.2) and (5.3). The right plot indicates the angle using the upper bound for $\cos(\theta)$ given in (5.4).

The calculation for $\boldsymbol{x}_2$ is similar. Let $\widetilde{\boldsymbol{g}}_1$ and $\widetilde{\boldsymbol{g}}_2$ be the $\widetilde{\boldsymbol{r}}_1$ vectors corresponding to $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, respectively. Then at the next application, we have

$$\widetilde{\boldsymbol{g}}_1 = \left(j + \sum_{h=1}^{k_1} \frac{u_h(1-u_h)}{(u_h+1)^2} + \sum_{h=1}^{k_2} \frac{1-u_h'}{(1+u_h')^2}, \sum_{h=1}^{k_1} \frac{1-u_h}{(u_h+1)^2} + \sum_{h=1}^{k_2} \frac{u_h'(1-u_h')}{(1+u_h')^2} \right) \tag{5.2}$$

and

$$\widetilde{\boldsymbol{g}}_2 = \left(\sum_{h=1}^{k_1} \frac{u_h(1-u_h)}{(u_h+1)^2} + \sum_{h=1}^{k_2} \frac{1-u_h'}{(1+u_h')^2}, j + \sum_{h=1}^{k_1} \frac{1-u_h}{(u_h+1)^2} + \sum_{h=1}^{k_2} \frac{u_h'(1-u_h')}{(1+u_h')^2} \right). \tag{5.3}$$

The angle between these two vectors is again given by (5.1). The resulting angles from simulations for various $k_1 = k_2$ and j are given in the left plot of Fig. 10. We make the simplification $k_1 = k_2$ to ease visualization. Unsurprisingly, as j increases so does the separation between $\widetilde{\boldsymbol{g}}_1$ and $\widetilde{\boldsymbol{g}}_2$. As k_1 and k_2 increase, for fixed j, the separation between $\widetilde{\boldsymbol{g}}_1$ and $\widetilde{\boldsymbol{g}}_2$ decreases.

Ideally, we would like to find a lower bound on the angle θ between $\widetilde{\boldsymbol{g}}_1$ and $\widetilde{\boldsymbol{g}}_2$ that depends on k_1, k_2 and j. Unfortunately, the explicit form of the resulting angle is relatively complicated. We can simplify the denominator of (5.1) by using the bounds $||\widetilde{\boldsymbol{g}}_i||_2 \geqslant j$. We expect this bound to be quite loose, if not trivial, when j is small, but to provide a reasonable bound for larger j. With this simplification,

$$\begin{aligned}\cos(\theta) \leqslant\ & \frac{\left(j + \sum_{h=1}^{k_1} \frac{u_h(1-u_h)}{(u_h+1)^2} + \sum_{h=1}^{k_2} \frac{1-u_h'}{(1+u_h')^2}\right)\left(\sum_{h=1}^{k_1} \frac{u_h(1-u_h)}{(u_h+1)^2} + \sum_{h=1}^{k_2} \frac{1-u_h'}{(1+u_h')^2}\right)}{j^2} \\ & + \frac{\left(\sum_{h=1}^{k_1} \frac{1-u_h}{(u_h+1)^2} + \sum_{h=1}^{k_2} \frac{u_h'(1-u_h')}{(1+u_h')^2}\right)\left(j + \sum_{h=1}^{k_1} \frac{1-u_h}{(u_h+1)^2} + \sum_{h=1}^{k_2} \frac{u_h'(1-u_h')}{(1+u_h')^2}\right)}{j^2}.\end{aligned} \tag{5.4}$$

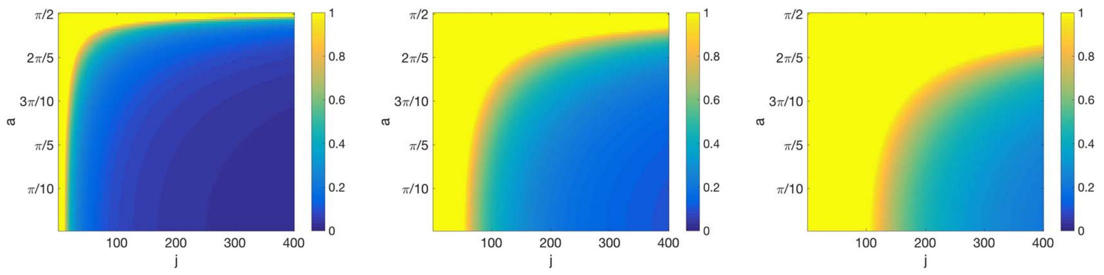

FIG. 11. For various values of $k_1 = k_2$ (the number of hyperplanes intersecting the wedges G_1 and G_2, respectively), j (the number of hyperplanes separating the wedges G_1 and G_2) and angles a, we plot the bound for $\mathbb{P}(\theta \leqslant a)$ given by Theorem 5.1. From left to right, the plots use $k_1 = 10, 50$ and 100, respectively.

For this simplified bound, taking an expectation is a straightforward calculation; see Appendix A for details. We eventually arrive at the bound

$$\begin{aligned}\mathbb{E}(\cos(\theta)) \leqslant \frac{(k_1+k_2)(2\log 2-1)}{j} + \frac{(k_1^2+k_2^2)(10(\log 2)^2-14\log 2+5)}{j^2} \\ + \frac{4k_1k_2(1-\log 2)(3\log 2-2)+(k_1+k_2)(-2/3+8\log 2-10(\log 2)^2)}{j^2}.\end{aligned} \tag{5.5}$$

Using Markov's inequality, for $a \in (0, \pi/2)$,

$$\mathbb{P}(\theta \leqslant a) = \mathbb{P}\left[\cos(\theta) \geqslant \cos(a)\right] \leqslant \frac{\mathbb{E}(\cos(\theta))}{\cos(a)}. \tag{5.6}$$

Although this bound is relatively loose, for sufficiently small a and large j, the probability that $\theta \leqslant a$ is small. We summarize this result in Theorem 5.1. More visually appealing, Fig. 11 gives the probabilities that result from combining (5.5) and (5.6) for a variety of hyperplane combinations and angles a.

THEOREM 5.1 Suppose data is distributed as in Fig. 9, where points from classes 1 and 2 are distributed with uniformly random angles within the wedges G_1 and G_2, respectively. Suppose that the angles A_1 and A_2 are equal. Let k_1 and k_2 be the number of hyperplanes that intersect G_1 and G_2, respectively. Let j be the number of hyperplanes that separate G_1 and G_2. Consider the points $\boldsymbol{x}_1$ in class 1 and $\boldsymbol{x}_2$ in class 2 as shown in Fig. 9. The angle θ between the $\widetilde{\boldsymbol{r}}$ vectors for $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ after a single iteration of SCB with one level L satisfies the following inequality:

$$\mathbb{P}(\theta \leqslant a) \leqslant \frac{C_1 j(k_1+k_2) + C_2(k_1^2+k_2^2) + C_3k_1k_2 + C_4(k_1+k_2)}{j^2\cos(a)},$$

where

$$C_1 = 2(\log 2) - 1, \qquad C_2 = 10(\log 2)^2 - 14\log 2 + 5,$$

$$C_3 = 4(1-\log 2)(3\log 2 - 2), \qquad C_4 = -10(\log 2)^2 + 8\log 2 - 2/3.$$

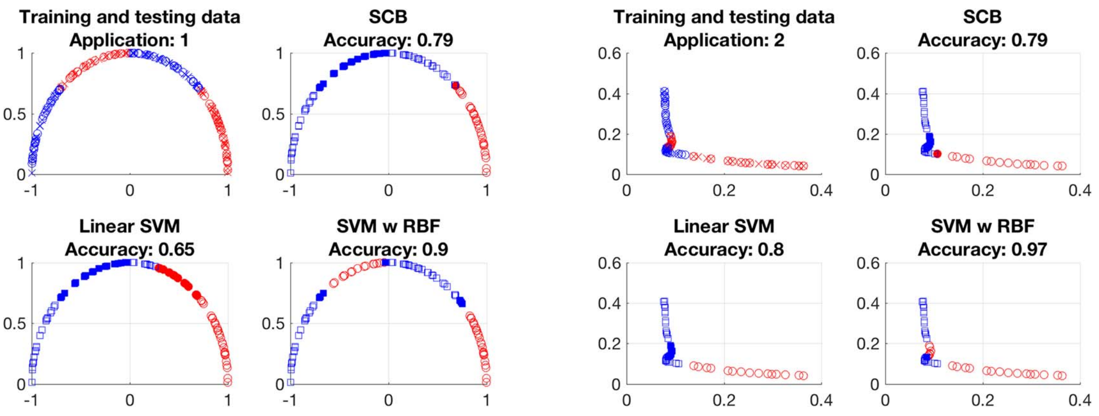

FIG. 12. The four plots on the left display accuracies and predictions made via various methods for the data given in the upper left-most plot. In the plots of the training and testing data, circles indicate training data and crosses indicate test data. Filled markers indicate that a given method misclassified that particular data point. The methods considered are SCB and SVM with both a linear and RBF kernel. The right set of four plots display accuracies and predictions made via the same set of methods applied to the $\widetilde{\boldsymbol{r}}$ values from a single application of SCB with a single level ($L = 1$) and $m = 100$ measurements.

6. ISCB for data preprocessing and dimension reduction

We remark here briefly about another potential strategy using the output of the SCB approach. Although this is not the focus of the current work, it may lead to fruitful future directions. The idea is to use the output from SCB and then apply other established classification methods such as SVM [15] to the $\widetilde{\boldsymbol{r}}$ vectors. Considering SVM specifically, we find that this strategy can perform better than SVM applied directly to the data.

First, consider a simple example with the synthetic data shown in the upper left plot of Fig. 12. Applying SVM with a linear kernel [15, 21] unsurprisingly performs poorly, achieving an accuracy of 65%. A radial basis function (RBF) kernel [8, 21] SVM performs much better, achieving an accuracy of 90%. Applying SVM instead to the $\widetilde{\boldsymbol{r}}_1$ values of the training data produced via SCB with a single level L and $m = 100$ measurements leads to 80% accuracy using a linear kernel and 97% accuracy using an RBF kernel. Thus, applying SVM to the $\widetilde{\boldsymbol{r}}_1$ values as opposed to the original data leads to an improvement in accuracy of 15% for SVM with a linear and 7% for SVM with an RBF kernel.

For the same initial data, if we increase the number of levels L used in SCB to four and the number of measurements to $m = 200$, the accuracies of SVM trained on the resulting $\widetilde{\boldsymbol{r}}_1$ values are 97% with a linear kernel and 94% with an RBF kernel (Fig. 13). The respective accuracies are improved by 21 and 4% respectively as compared to SVM applied to the original data. This increase in the number of levels L and measurements m also leads to improved performance for both SCB and ISCB with two applications. Note that if SCB is able to perfectly classify the training data points, then SVM with a linear kernel trained on the $\widetilde{\boldsymbol{r}}_1$ values of the training points will also perfectly classify the training data points, as the $\widetilde{\boldsymbol{r}}_1$ values of the training points will be linearly separable.

7. Conclusions and avenues for future work

The ISCB framework offers many directions for extensions and further analyses. In Section 5, we provide theoretical analyses for simple settings. These analyses could be extended to other more

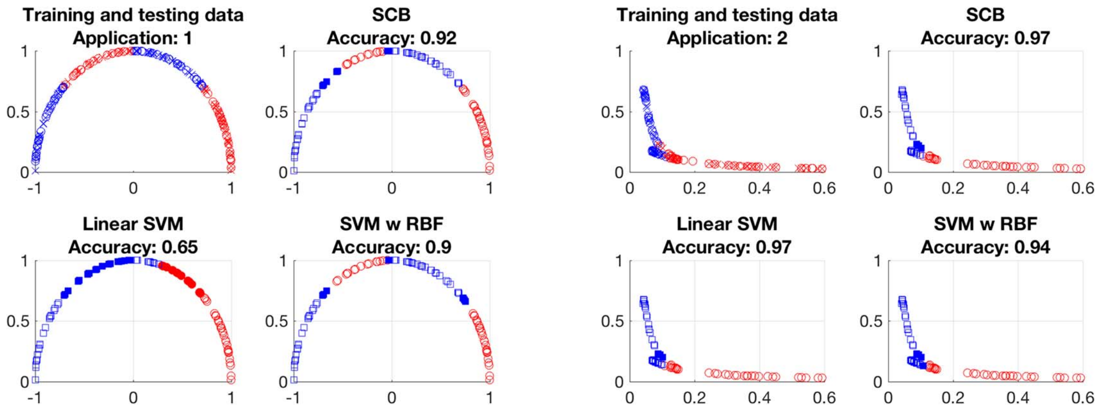

FIG. 13. The four plots on the left display accuracies and predictions made via various methods for the data given in the upper left-most plot. In the plots of the training and testing data, circles indicate training data and crosses indicate test data. Filled markers indicate that a given method misclassified that particular data point. The methods considered are SCB and SVM with both a linear and RBF kernel. The right set of four plots display accuracies and predictions made via the same set of methods applied to the $\tilde{\boldsymbol{r}}$ values from SCB at the first application. $L = 4$ levels and $m = 200$ measurements are used for each application of SCB.

sophisticated settings or to produce stronger guarantees. For example, while Theorem 5.1 gives probabilistic bounds on the separating angle between classes after a single application of SCB, Theorem 2 of [36] cannot be applied directly to give a probability of the correct classification at the next iteration, as the transformed data produced by the SCB application will no longer be distributed uniformly within the respective wedge for each class. Explicit guarantees for when ISCB has a higher probability of correct classification than SCB could provide further insights on the performance of ISCB. One could also investigate guarantees when the measurements are noisy, leading to potential sign flips in the binary matrix $\boldsymbol{Q}$. Incorporating dithers or bias terms in the random linear measurements so that $\boldsymbol{Q} = \text{sign}(\boldsymbol{AX} + b)$ is another possible extension, although determining the scale for the values of b adds additional complexity [4, 27]. Ideas from 1-bit compressive sensing could potentially be used to exploit sparsity in the data as well [4, 7].

One could incorporate ideas from other machine learning models such as neural networks within the ISCB framework. For example, one could use informed or learned separating hyperplane measurements as opposed to entirely random hyperplanes. One could also use weighted linear combinations of the $\boldsymbol{r}_k$ parameters with learned weights in order to make predictions or compute the $\tilde{\boldsymbol{r}}_k$ for the subsequent application of SCB.

A variety of methods exist for summarizing and representing data with binary descriptors [1, 12, 33, 39–43, 45]. These methods could be used as alternatives to generating the binary matrix $\boldsymbol{Q}$ as opposed to using binarized linear measurements of the data as is considered here and in [36]. Such alternative strategies could be particularly effective when applied to image or video data, as many methods for generating binary descriptors have been developed specifically for applications within the field of computer vision. The natural geometric interpretation of the binarized linear measurements $\boldsymbol{Q} = \text{sign}(\boldsymbol{AX})$ is advantageous, however, for deriving performance guarantees and maintaining the geometric interpretation of SCB and ISCB.

We have illustrated that iterative applications of SCB of [36] lead to improved classification accuracies as compared to the original single iteration approach in a variety of settings. Numerical experiments on the MNIST, YaleB and NORB datasets support this claim. For a simple angular model, we provide a bound on the probability that the separation between the transformed data of two classes is small after an application of SCB. Experiments and theoretical analyses on synthetic data in simple settings demonstrate the effects of multiple iterations on the data and predictions. These examples also highlight simple situations in which the ISCB framework excels.

Acknowledgements

The authors would like to thank Tina Woolf for help with the initial code used for the SCB method.

Funding

National Science Foundation Division of Mathematics (CAREER #1348721); National Science Foundation (BIGDATA #1740325).

References

1. Alahi, A., Ortiz, R. & Vandergheynst, P. (2012) Freak: fast retina keypoint. *2012 IEEE Conference on Computer Vision and Pattern Recognition*. Providence, RI, USA: IEEE, pp. 510–517.

2. Aziz, P. M., Sorensen, H. V. & Vn der Spiegel, J. (1996) An overview of sigma-delta converters. *IEEE Signal Process. Mag.*, **13**, 61–84.

3. Banfield, R. E., Hall, L. O., Bowyer, K. W. & Kegelmeyer, W. P. (2007) A comparison of decision tree ensemble creation techniques. *IEEE Trans. Pattern Anal. Mach. Intell.*, 173–180.

4. Baraniuk, R. G., Foucart, S., Needell, D., Plan, Y. & Wootters, M. (2017) Exponential decay of reconstruction error from binary measurements of sparse signals. *IEEE Trans. Inf. Theory*, **63**, 3368–3385.

5. Barocas, S., Bradley, E., Honavar, V. & Provost, F. (2017) Big data, data science, and civil rights. https://cra.org/ccc/resources/ccc-led-whitepapers/.

6. Barocas, S. & Selbst, A. D. (2016) Big data's disparate impact. *Calif. Law Rev.*, **104**, 671.

7. Boufounos, P. T. & Baraniuk, R. G. (2008) 1-bit compressive sensing. *2008 42nd Annual Conference on Information Sciences and Systems*. Princeton, NJ, USA: IEEE, pp. 16–21.

8. Buhmann, M. D. (2003) *Radial Basis Functions: Theory and Implementations*, vol. **12**. Cambridge, UK: Cambridge University Press.

9. Cai, D., He, X. & Han, J. (2007) Spectral regression for efficient regularized subspace learning. *International Conference on Computer Vision*, Rio de Janeiro, Brazil.

10. Cai, D., He, X., Han, J. & Zhang, H.-J. (2006) Orthogonal Laplacianfaces for face recognition. *IEEE Trans. Image Process.*, **15**, 3608–3614.

11. Cai, D., He, X., Hu, Y., Han, J. & Huang, T. (2007) Learning a spatially smooth subspace for face recognition. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Minnesota, USA.

12. Calonder, M., Lepetit, V., Strecha, C. & Fua, P. (2010) Brief: binary robust independent elementary features. *European Conference on Computer Vision*. Heraklion, Crete, Greece: Springer, pp. 778–792.

13. Cao, Y., Qi, H., Zhou, W., Kato, J., Li, K., Liu, X. & Gui, J. (2018) Binary hashing for approximate nearest neighbor search on big data: a survey. *IEEE Access*, **6**, 2039–2054.

14. Collobert, R. & Weston, J. (2008) A unified architecture for natural language processing: deep neural networks with multitask learning. *Proceedings of the 25th International Conference on Machine Learning*. Helsinki, Finland: ACM, pp. 160–167.

15. Cortes, C. & Vapnik, V. (1995) Support-vector networks. *Mach. Learn.*, **20**, 273–297.

16. Cybenko, G. (1989) Approximation by superpositions of a sigmoidal function. *Math. Control Signals Syst.*, **2**, 303–314.

17. Munoz, C., Smith, M. & Patil, D. (2016) Big data: a report on algorithmic systems, opportunity, and civil rights. Executive Office of the President. https://obamawhitehouse.archives.gov/sites/default/files/microsites/ostp/2016_0504_data_discrimination.pdf.

18. Fang, J., Shen, Y., Li, H. & Ren, Z. (2014) Sparse signal recovery from one-bit quantized data: an iterative reweighted algorithm. *Signal Process.*, **102**, 201–206.

19. Freund, Y. & Schapire, R. E. (1997) A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, **55**, 119–139.

20. Freund, Y., Schapire, R. & Abe, N. (1999) A short introduction to boosting. *Japan. Soc. Artif. Intell.*, **14**, 1612.

21. Friedman, J., Hastie, T. & Tibshirani, R. (2001) *The Elements of Statistical Learning*, vol. **1**. Springer Series in Statistics New York. NY, USA: Springer.

22. Gionis, A., Indyk, P. & Motwani, R. (1999) Similarity search in high dimensions via hashing. *Proceedings of the 25th International Conference on Very Large Data Bases*, San Francisco, California, Morgan Kaufmann Publishers, vol. **99**, pp. 518–529.

23. Grauman, K. & Fergus, R. (2013) Learning binary hash codes for large-scale image search. *Machine Learning for Computer Vision*. Berlin, Heidelberg, Germany: Springer, pp. 49–87.

24. He, K., Zhang, X., Ren, S. & Sun, J. (2016) Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Las Vegas, NV, USA: IEEE, pp. 770–778.

25. He, X., Yan, S., Hu, Y., Niyogi, P. & Zhang, H.-J. (2005) Face recognition using Laplacianfaces. *IEEE Trans. Pattern Anal. Mach. Intell.*, **27**, 328–340.

26. Indyk, P. & Motwani, R. (1998) Approximate nearest neighbors: towards removing the curse of dimensionality. *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. Dallas, Texas, USA: ACM, pp. 604–613.

27. Jacques, L., Laska, J., Boufounos, P. & Baraniuk, R. (2013) Robust 1-bit compressive sensing via binary stable embeddings of sparse vectors. *IEEE Trans. Inf. Theory*, **59**, 2082–2102.

28. Laska, J. N., Wen, Z., Yin, W. & Baraniuk, R. G. (2011) Trust, but verify: fast and accurate signal recovery from 1-bit compressive measurements. *IEEE Trans. Signal Process.*, **59**, 5289–5301.

29. LeCun, Y., Bengio, Y. & Hinton, G. (2015) Deep learning. *Nature*, **521**, 436.

30. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998) Gradient-based learning applied to document recognition. *Proc. IEEE*, **86**, 2278–2324.

31. LeCun, Y., Cortes, C. & Burges, C. (2010) MNIST handwritten digit database. *AT&T Labs*, 2. http://yann.lecun.com/exdb/mnist.

32. LeCun, Y., Huang, F. J. & Bottou, L. (2004) Learning methods for generic object recognition with invariance to pose and lighting. *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 2*. Washington, DC, USA: IEEE, pp. II–104.

33. Leutenegger, S., Chli, M. & Siegwart, R. (2011) BRISK: binary robust invariant scalable keypoints. *International Conference on Computer Vision*. Barelona, Spain: IEEE, pp. 2548–2555.

34. Murthy, S. K., Kasif, S. & Salzberg, S. (1994) A system for induction of oblique decision trees. *J. Artif. Intell. Res.*, **2**, 1–32.

35. Murthy, S. K., Kasif, S., Salzberg, S. & Beigel, R. (1993) OC1: a randomized algorithm for building oblique decision trees. *Association for the Advancement of Artificial Intelligence, vol. 93*, pp. 322–327. Citeseer.

36. Needell, D., Saab, R. & Woolf, T. (2018) Simple classification using binary data. *J. Mach. Learn. Res.*, **19**, 2487–2516.

37. Norouzi, M., Punjani, A. & Fleet, D. J. (2012) Fast search in hamming space with multi-index hashing. *2012 IEEE Conference on Computer Vision and Pattern Recognition*. Providence, RI, USA: IEEE, pp. 3108–3115.

38. Persson, A. & Loutfi, A. (2016) Fast matching of binary descriptors for large-scale applications in robot vision. *Int. J. Adv. Robot. Syst.*, **13**, 58.

39. Rublee, E., Rabaud, V., Konolige, K. & Bradski, G. R. (2011) ORB: an efficient alternative to SIFT or SURF. *International Conference on Computer Vision, vol. 11, p. 2*. Barcelona, Spain, Citeseer.
40. Strecha, C., Bronstein, A., Bronstein, M. & Fua, P. (2012) LDAHash: improved matching with smaller descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, **34**, 66–78.
41. Trzcinski, T., Christoudias, M., Fua, P. & Lepetit, V. (2013) Boosting binary keypoint descriptors. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Portland, OR, USA: IEEE, pp. 2874–2881.
42. Trzcinski, T., Christoudias, M. & Lepetit, V. (2014) Learning image descriptors with boosting. *IEEE Trans. Pattern Anal. Mach. Intell.*, **37**, 597–610.
43. Trzcinski, T. & Lepetit, V. (2012) Efficient discriminative projections for compact binary descriptors. *European Conference on Computer Vision*. Florence, Italy: Springer, pp. 228–242.
44. Ventura, C., Masip, D. & Lapedriza, A. (2017) Interpreting CNN models for apparent personality trait regression. *IEEE Conference on Computer Vision and Pattern Recognition Workshops*. Honolulu, HI, USA: IEEE, pp. 1705–1713.
45. Yang, X. & Cheng, K.-T. (2012) LDB: an ultra-fast feature for scalable augmented reality on mobile devices. *2012 IEEE International Symposium on Mixed and Augmented Reality*. Atlanta, Georgia, USA: IEEE, pp. 49–57.
46. Zhang, Q., Nian Wu, Y. & Zhu, S.-C. (2018) Interpretable convolutional neural networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT, USA: IEEE, pp. 8827–8836.
47. Zhang, Q.-S. & Zhu, S.-C. (2018) Visual interpretability for deep learning: a survey. *Front. Inform. Tech. Electron. Eng.*, **19**, 27–39.
48. Zhao, K., Lu, H. & Mei, J. (2014) Locality preserving hashing. *Twenty-eighth AAAI Conference on Artificial Intelligence*, Palo Alto, CA, USA, AAAI Press.

Appendix A. Detailed calculations for Section 5.2

In this section, we provide details for calculating (5.5). Let

$$K_{11} = \sum_{h=1}^{k_1} \frac{u_h(1-u_h)}{(u_h+1)^2},\ K_{12} = \sum_{h=1}^{k_1} \frac{1-u_h}{(u_h+1)^2},\ K_{21} = \sum_{h=1}^{k_2} \frac{1-u'_h}{(1+u'_h)^2},\ K_{22} = \sum_{h=1}^{k_2} \frac{u'_h(1-u'_h)}{(1+u'_h)^2},$$

where u_h and u'_h are i.i.d. uniformly random variables between zero and one. We can then rewrite (5.4) as

$$\begin{aligned}\cos(\theta) &\leqslant \frac{(j+K_{11}+K_{21})(K_{11}+K_{21}) + (j+K_{12}+K_{22})(K_{12}+K_{22})}{j^2} \\ &= \frac{j(K_{11}+K_{21}+K_{12}+K_{22}) + K_{11}^2 + 2K_{11}K_{21} + K_{21}^2 + K_{12}^2 + 2K_{12}K_{22} + K_{22}^2}{j^2}. \end{aligned} \quad \text{(A.1)}$$

We then require the expectation of each term in the numerator. Since u_h and u'_h are i.i.d., $\mathbb{E}K_{11}K_{21} = \mathbb{E}K_{11}\mathbb{E}K_{21}$. Straightforward integral calculations lead to the following expected values:

$$\mathbb{E}\left(\frac{u_h(1-u_h)}{(u_h+1)^2}\right) = 3\log 2 - 2, \qquad \mathbb{E}\left(\frac{1-u_h}{(u_h+1)^2}\right) = 1 - \log 2,$$

$$\mathbb{E}\left(\frac{u_h^2(1-u_h)^2}{(u_h+1)^4}\right) = 25/6 - 6\log 2, \qquad \mathbb{E}\left(\frac{(1-u_h)^2}{(u_h+1)^4}\right) = 1/6.$$

We then have the following expectations:

$$\begin{aligned}
\mathbb{E}K_{11} &= k_1(3\log 2 - 2)\\
\mathbb{E}K_{12} &= k_1(1 - \log 2)\\
\mathbb{E}K_{11}^2 &= k_1(k_1 - 1)(3\log 2 - 2)^2 + k_1(25/6 - 6\log 2)\\
\mathbb{E}K_{12}^2 &= k_1(k_1 - 1)(1 - \log 2)^2 + k_1(1/6).
\end{aligned}$$

$\mathbb{E}K_{22}, \mathbb{E}K_{21}, \mathbb{E}K_{22}^2$ and $\mathbb{E}K_{21}^2$ take the same forms with k_2 replacing k_1.

Taking the expectation of (A.1),

$$\begin{aligned}
\mathbb{E}(\cos(\theta)) \leqslant\ & \frac{(k_1 + k_2)(2\log 2 - 1)}{j}\\
&+ \frac{(k_1^2 - k_1 + k_2^2 - k_2)(3\log 2 - 2)^2 + (k_1^2 - k_1 + k_2^2 - k_2)(1 - \log 2)^2}{j^2}\\
&+ \frac{4k_1k_2(1 - \log 2)(3\log 2 - 2) + (k_1 + k_2)(1/6 + 25/6 - 6\log 2)}{j^2}\\
\leqslant\ & \frac{(k_1 + k_2)(2\log 2 - 1)}{j}\\
&+ \frac{(k_1^2 - k_1 + k_2^2 - k_2)(10(\log 2)^2 - 14\log 2 + 5)}{j^2}\\
&+ \frac{4k_1k_2(1 - \log 2)(3\log 2 - 2) + (k_1 + k_2)(13/3 - 6\log 2)}{j^2}\\
\leqslant\ & \frac{(k_1 + k_2)(2\log 2 - 1)}{j} + \frac{(k_1^2 + k_2^2)(10(\log 2)^2 - 14\log 2 + 5)}{j^2}\\
&+ \frac{4k_1k_2(1 - \log 2)(3\log 2 - 2) + (k_1 + k_2)(-2/3 + 8\log 2 - 10(\log 2)^2)}{j^2},
\end{aligned}$$

providing the desired bound.