Vectorization of Bias in Machine Learning Algorithms

Sophie Bekerman[1,*][a], Eric Chen[1,*][b], Lily Lin[2,*][c] and George D. Monta Nez[1][d]
[1]*AMISTAD Lab, Dept. of Computer Science, Harvey Mudd College, Claremont, CA, U.S.A.*
[2]*Department of Math and Computer Science, Biola University, La Mirada, CA, U.S.A.*

Keywords: Inductive Bias, Algorithmic Bias, Vectorization, Algorithmic Search Framework.

Abstract: We develop a method to measure and compare the inductive bias of classifications algorithms by vectorizing aspects of their behavior. We compute a vectorized representation of the algorithm's bias, known as the inductive orientation vector, for a set of algorithms. This vector captures the algorithm's probability distribution over all possible hypotheses for a classification task. We cluster and plot the algorithms' inductive orientation vectors to visually characterize their relationships. As algorithm behavior is influenced by the training dataset, we construct a Benchmark Data Suite (BDS) matrix that considers algorithms' pairwise distances across many datasets, allowing for more robust comparisons. We identify many relationships supported by existing literature, such as those between k-Nearest Neighbor and Random Forests and among tree-based algorithms, and evaluate the strength of those known connections, showing the potential of this geometric approach to investigate black-box machine learning algorithms.

1 INTRODUCTION

With the growing prevalence of black-box algorithms in machine learning, finding ways to evaluate these algorithms is crucial. We propose the inductive orientation vector, a geometric representation of bias, as a tool for analyzing learning algorithms. The inductive orientation vector captures the probability distribution of an algorithm's predictions. This vector originates within the *algorithmic search framework* (Montañez et al., 2019), and quantifies the structural manifestations of various assumptions contained in general algorithms. By empirically estimating this vector for various algorithms, researchers can compare the algorithms' inductive biases on particular benchmark datasets, much like how current model comparisons use accuracy, precision, and memorization capacity to compare algorithms (Caruana and Niculescu-Mizil, 2006; Osisanwo et al., 2017; Rong et al., 2021). In doing so, we can identify hidden connections between algorithms. By comparing the biases of new algorithms to known biases of existing algorithms, we provide a point of reference useful for determining whether the new algorithms share significant biases with previous ones or incorporate novel assumptions. We can also use the inductive orientation vector to compute other algorithm quantities such as algorithmic bias (Bashir et al., 2020), algorithmic capacity, and entropic expressivity (Lauw et al., 2020), which we will explore in future work. The inductive orientation vector provides an objective measure for bias in black-box algorithms, making it easier to choose the best algorithm for a problem without resorting to heuristics. Previously a strictly theoretical concept, this analysis allows us to evaluate the behavior of algorithms without knowing its underlying structure.

The remainder is organized as follows. In Section 2, we discuss prior work related to bias and vectorization. In Section 3, we summarize the algorithmic search framework and several relevant definitions. We then establish a process to estimate inductive orientation vectors and present results in Section 4. In Sections 5 and 6, we implement and discuss additional ways to analyze inductive orientation vectors.

2 RELATED WORK

The goal of the inductive orientation vector is to capture inductive bias. An algorithm's inductive bias is the set of assumptions the algorithm makes be-

[a] https://orcid.org/0000-0001-6497-3133
[b] https://orcid.org/0000-0002-0469-3858
[c] https://orcid.org/0000-0003-2500-6495
[d] https://orcid.org/0000-0002-1333-4611
*These authors contributed equally

Bekerman, S., Chen, E., Lin, L. and Monta Nez, G.
Vectorization of Bias in Machine Learning Algorithms.
DOI: 10.5220/0010845000003116
In *Proceedings of the 14th International Conference on Agents and Artificial Intelligence (ICAART 2022) - Volume 2*, pages 354-365
ISBN: 978-989-758-547-0; ISSN: 2184-433X

yond strict consistency with training data (Mitchell, 1980). This bias inheres in the algorithm, so it is distinct from bias in the training data. Bias is necessary for learning because, without it, a classification algorithm cannot generalize beyond training data (Mitchell, 1980). This result has been supported experimentally (Runarsson and Yao, 2005), and formally extended to show that algorithms must incorporate biasing assumptions to perform better than uniform random guessing on unseen data (Montañez et al., 2019). If inductive bias is responsible for an algorithm's successes and failures on various problems, then algorithms with similar inductive biases may behave similarly on a greater proportion of problems than algorithms with different biases. This predictive potential of inductive biases motivates the concept of the inductive orientation vector.

A vector representation of bias can allow geometric analysis of hidden connections between various algorithms. Over the past decade, vectorization has revolutionized machine learning. One prominent case is vector space word embeddings in natural language processing. Word embedding methods capture semantic notions geometrically (Gonen and Goldberg, 2019), allowing for "vector arithmetic" of words, such as `Madrid - Spain + France = Paris` (Mikolov et al., 2013). They are also useful in practice: using word embeddings, the language model GPT-3 recently achieved state-of-the-art results for many natural language benchmark tasks (Brown et al., 2020). Given the ubiquity and utility of vector representations of words, a vector representation of bias could be equally far-reaching.

3 THE SEARCH FRAMEWORK

3.1 The Search Problem

Inductive orientation vectors were initially introduced in the context of the algorithmic search framework, a learning-theory framework for analyzing machine learning and search algorithms (Montañez, 2017). Inductive orientation vectors were designed to compute specific theoretical quantities related to learning algorithm performance; estimating these vectors empirically can allow us to estimate those related quantities (Montañez et al., 2021). Within this framework, the process of learning is cast as a search for an ideal hypothesis (or hypotheses). Each search problem is represented by the 3-tuple, (Ω, T, F). The search space Ω is a finite set of all possible solutions (hypotheses) to a search problem. Within the set Ω is the target set T that contains the desirable and/or acceptable solutions. The target function $\mathbf{t}$ conveniently encodes the relation between the target set and its respective search space as a binary $|T|$-hot vector of length $|\Omega|$. Each entry in the target vector corresponds to a potential solution ω in the search space. At each index, the target vector takes on the value of 1 if $\omega \in T$ and 0 otherwise. The external information resource F provides initialization information (e.g., training data) and guides the algorithm in the search process by evaluating elements of the search space (through a loss or fitness function). The search framework accommodates many types of algorithms; for example, we can cast classification problems as search problems in the framework by considering classification as the search for a correct sequence of labels in the search space of all possible labels (Montañez, 2017).

To make this more concrete, consider a classification problem in which there are C different labels (classes). We train a classification model on a training set and evaluate its performance on a holdout (test) set of size N. Here, we are searching for a correct (or acceptable) labeling of the entire holdout set; thus, the search space Ω would be the set of all possible combinations of labeling the elements in the holdout set where $|\Omega| = C^N$. The target set T would consist of sequences of correct (or acceptable) labelings of the holdout set. Depending on the problem, we might require that a sequence of labels be completely correct to be included in the target set, or we might be satisfied with sequences that label the holdout set within a chosen threshold of accuracy. Finally, the external information resource F is the classification model's loss function and data used to train the model.

3.2 The Search Algorithm

During the search process, a search algorithm $\mathcal{A}$ will induce a probability distribution P_i on the search space Ω; the algorithm will assign higher probability to elements in the search space that it perceives are likely to be in the target set. The search process is an iterative process; the number of iterations varies depending on the learning task. For each iteration, the search algorithm induces a probability distribution P_i over the search space Ω which is determined using the search history H. H contains a series of tuples $(\omega_i, F(\omega_i))$ where ω_i is a solution previously queried by the algorithm at the ith iteration (or time step) and its corresponding evaluation under the external information resource F. At each iteration i an algorithm queries an element ω_i from the search space based on the current probability distribution P_i, evaluates ω_i using the F, and adds the tuple $(\omega_i, F(\omega_i))$ into the search history H. The algorithm then adjusts the

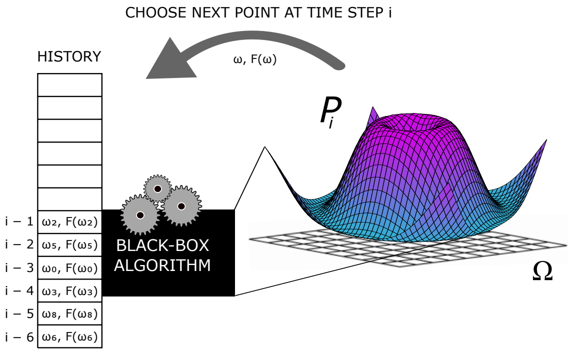


Figure 1: Black-box search algorithm. At time i the algorithm computes a probability distribution P_i over the search space Ω, using information from the history, and a new point is drawn according to P_i. The point is evaluated using external information resource F. The tuple $(\omega, F(\omega))$ is then added to the history at position i. Note, indices on ω elements do not correspond to time step in this diagram, but to sampled locations.

probability distribution on the search space, P_{i+1}, according to the search history. By the end of a search process (or run), a probability distribution sequence $\tilde{P}$ is produced. If, at the end of the run, the search history contains at least one element in the target set, the algorithm is successful; this is only evaluated after the search process is completed because the algorithm does not have access to the target set. Figure 1 provides a graphical representation of the search process.

3.3 Inductive Orientation Vector

Following Montañez, we use the *expected per-query probability of success* to measure an algorithm's performance (Montañez, 2017). Per-query normalization accounts for differences in the number of sampling steps (iterations) per run. Taking the expectation of multiple runs of the search process when trained on the same information resource F accounts for any stochastic differences between different runs. Mathematically, the expected per-query probability of success is defined as

$$q(T,F) = \mathbb{E}_{\tilde{P},H}\left[\frac{1}{|\tilde{P}|}\sum_{i=1}^{|\tilde{P}|} P_i(\omega \in T)\middle| F\right] \qquad (1)$$

where $\tilde{P}$ is a sequence of probability distributions P_i at each iteration i over the search space, T is the target set, F is the external information resource, and H is the search history. The number of queries made during a search is equal to the length of $|\tilde{P}|$. The expectation accounts for stochastic differences between multiple runs of the algorithm, while the inner quantity measures the expected probability of success of a single run (Lauw et al., 2020).

Previously, Montañez demonstrated a more convenient way of expressing the expected per-query probability of success as the inner product of the target function and a vector representing the expected probability distribution induced by the search algorithm $\mathcal{A}$ over multiple runs (Montañez, 2017). Let $\overline{\mathbf{P}}_F$ be the vector representation of this averaged probability distribution (conditioned on F) induced on Ω over multiple runs of the search process. Formally, define

$$\overline{\mathbf{P}}_F := \mathbb{E}_{\tilde{P},H}\left[\frac{1}{|\tilde{P}|}\sum_{i=1}^{|\tilde{P}|} P_i \middle| F\right]. \qquad (2)$$

This is the **inductive orientation vector** of $\mathcal{A}$ relative to a particular information resource F. The sum of all the probability mass on elements in the target set yields the probability of success, which is the probability that the algorithm queries an element in T. This sum, known as the single-query probability of success, is represented by $\mathbf{t}^\top \overline{\mathbf{P}}_F$, where $\mathbf{t}$ is a $|T|$-hot target function. This establishes the equivalence between the expected per-query probability of success over an entire search and the single-query probability of success (sampled from the averaged probability distribution induced by the search algorithm $\mathcal{A}$ in expectation), which is represented by $q(T,F) = \mathbf{t}^\top \overline{\mathbf{P}}_F$.

In the case that an algorithm is trained on different information resources F, its expected performance is calculated by finding an inductive orientation vector $\overline{\mathbf{P}}_{\mathcal{D}}$ relative to a data-generating distribution $\mathcal{D}$ on the space of information resources $\mathcal{F}$. Formally,

$$\overline{\mathbf{P}}_{\mathcal{D}} = \mathbb{E}_{\mathcal{D}}\left[\overline{\mathbf{P}}_F\right] = \mathbb{E}_{\mathcal{D}}\left[\mathbb{E}_{\tilde{P},H}\left[\frac{1}{|\tilde{P}|}\sum_{i=1}^{|\tilde{P}|} P_i \middle| F\right]\right] \qquad (3)$$

where F is distributed according to $\mathcal{D}$ (i.e., $F \sim \mathcal{D}$). Then, the expected per-query probability of success of an algorithm $\mathcal{A}$ when trained on information resources F generated by the data generating process $\mathcal{D}$ is $\mathbb{E}_{F\sim\mathcal{D}}[q(T,F)] = \mathbf{t}^\top \overline{\mathbf{P}}_{\mathcal{D}}$. Both measures of success, $q(T,F)$ and $\mathbb{E}_{F\sim\mathcal{D}}[q(T,F)]$, will be used.

3.4 Inductive Bias

Since an inductive orientation vector represents a learning algorithm's probability distribution over its search space, it captures aspects of the algorithm's inductive bias. Inductive bias is set of the assumptions built into a model, implicitly or explicitly, that allow it to generalize from training data. Although an algorithm's inductive bias determines how it interacts with data, it is a property of the algorithm that is independent of the data. However, the close relationship of inductive bias and data makes it difficult to isolate an algorithm's inductive bias. Thus, the inductive orientation vector, $\overline{\mathbf{P}}_{\mathcal{D}}$, aims to estimate this property relative to a specific distribution of information resources and search space. When comparing the inductive orientation vectors of different algorithms trained on the same set of information resources and search space, we can attribute their differences to differences between algorithms' inductive biases. In this way, the inductive orientation vector makes an algorithm's assumptions explicit and measurable.

4 ESTIMATING INDUCTIVE ORIENTATIONS

4.1 Method

We now present our method for estimating inductive orientation vectors. We begin by describing our data-generating process as well as some alternative methods, introducing the Labeling Distribution Matrix, and reviewing the inductive orientation vector.

Given a dataset, we first split the dataset into a training set of n instances and a holdout set of size N. Note that we assume this dataset to be representative of its data-generating distribution $\mathcal{D}$. To generate a data resource F from the space, we sample a size m subset from the size n training set with replacement between subset selections. Note that each instance in subset F is drawn without replacement.

Since F is a subset of the training set that was drawn from $\mathcal{D}$, F itself approximates a sample from $\mathcal{D}$ (assuming m is large enough), being a subset of the original i.i.d. instances drawn from that distribution. Sampling m out of n instances is crucial, as this allows us to estimate $\overline{\mathbf{P}}_{\mathcal{D}}$ vectors even when we have more data than is computationally feasible for training an algorithm. Each bootstrap subsample is drawn from the empirical distribution which assigns to each instance in the training dataset a probability of $1/n$.

Having described our data-generating process, we provide the pseudocode (Algorithm 1) of how we generate a Labeling Distribution Matrix (LDM, first introduced by (Sandoval Segura et al., 2020)) relative to a learning algorithm $\mathcal{A}$ and a distribution $\mathcal{D}$. We introduce LDMs as they are our data structure for estimating inductive orientation vectors. Before explaining the details of this estimation, let us review how to construct an LDM and what its components are.

Let K be the number of independently selected subsets F of the training set. Let R be the number of times we repeat the process of training $\mathcal{A}$ on the same subsample F_k; this is to account for the stochastic nature of some algorithms. We let $\overline{\mathbf{P}}_{F_k}$ denote the averaged probability distribution induced over its search space Ω after training $\mathcal{A}$ on the subsample F_k R times. Together, the K simplex vectors $\overline{\mathbf{P}}_{F_k}$ form the columns of the LDM. Note that $\overline{\mathbf{P}}_{F_k}$ is the probability distribution vector averaged over only the final iteration of the search. In other words, $\mathbf{P}_{F_{k_r}}$ corresponds to the probability distribution induced over Ω at the last iteration in the search after being trained on F_k. Having computed the LDM, we compute the inductive orientation vector $\overline{\mathbf{P}}_{\mathcal{D}}$ relative to an algorithm $\mathcal{A}$ by simply taking the average across the columns of the LDM. In other words, we take the average of the $\overline{\mathbf{P}}_{F_k}$'s to be the inductive orientation vector relative to the data distribution $\mathcal{D}$ (using our bootstrapped approximation of it). In cases where multiple i.i.d. samples can be drawn from $\mathcal{D}$ directly, the estimation will become correspondingly more accurate.

Algorithm 1: Labeling Distribution Matrix (LDM).

```
for all $k = 1, \dots, K$ do
    $F_k \leftarrow$ Sample without replacement from training set
    for all $r = 1, \dots, R$ do
        Generate $\mathbf{P}_{F_{k_r}}$ after training $\mathcal{A}$ on $F_k$
        $\overline{\mathbf{P}}_{F_k} = \overline{\mathbf{P}}_{F_k} + \mathbf{P}_{F_{k_r}}$
    end for
    $\overline{\mathbf{P}}_{F_k} = \overline{\mathbf{P}}_{F_k}$ / R
    Store $\overline{\mathbf{P}}_{F_k}$ in LDM
end for
Return LDM
```

4.1.1 Experimental Setup

To estimate inductive orientation vectors, we selected a variety of machine learning algorithms from Python's scikit-learn library (version 0.22.2.post1) (Pedregosa et al., 2011). The algorithms, along with their parameters, are specified in Table 2. If a parameter is not specified, then the default value is used. Note that both Linear SVC and SVC with a linear kernel are included because we wanted to see if differences in implementation would affect algorithm behavior; Linear SVC is implemented using liblinear rather than libsvm, making it more scalable for larger datasets. Also, SGD Classifiers refer to algorithms that are optimized by stochastic gradient descent. We selected well-studied algorithms to evaluate our results against existing research. In general, our methods could be applied to any classification algorithm.

Since inductive orientation vectors are relative to an algorithm $\mathcal{A}$ and a data distribution $\mathcal{D}$, we selected 10 datasets, 9 of which were chosen from the UCI Machine Learning Repository (Dua and Graff, 2017; Moro et al., 2014; Cortez et al., 2009; Baati and Mohsil, 2020; Palechor and de la Hoz Manotas, 2019). One dataset (denoted Random dataset) was generated using `np.random` with RandomState set to 42. These datasets were chosen with the intention of having a collection of data of varying complexities to test the strength of inductive orientation similarities across diverse datasets. All datasets contain at least 1000 instances. For some datasets, feature engineering was used to encode categorical features as either numerical values or one-hot vectors. To shorten the estimation time for the inductive orientation vectors, non-binary classification datasets were converted into binary classification problems in a way that balanced the number of instances in each class.

Table 1: Datasets, their sizes, the size *m* of the subset used to train the learning algorithms, and dataset complexity ratios (CR, described in the main text).

Dataset	**Size**	**m**	**CR**
Obesity	2111	211	0.081
Letter Recognition	1609	260	0.254
Wine Quality	6496	500	0.345
Abalone	4176	417	0.747
Shopper’s Intention	12245	600	0.769
EEG Eye State	14980	600	0.780
Car Evaluation	1728	170	0.916
Bank Marketing	11162	500	0.980
Random	1609	260	1.004
Spam	4601	460	1.073

For each of the 10 datasets and for each algorithm selected, we estimated the corresponding inductive orientation vector using the following scheme: a holdout set of 5 instances, 800 subsets sampled from the training set (using the process outlined in Section 4.1), and 20 runs on the same data subset; namely, $N = 5$, $K = 800$, and $R = 20$. Empirically, we found that using 800 subsets led to more stable inductive orientation vectors with low variance, and increasing the number of subsets further did not significantly decrease variance. For a given dataset, all algorithms were trained on a subset of the data of the *same* size. The size of the training subset depends on the dataset (refer to Table 1 for details). For smaller datasets, the number of instances per subset was reduced to avoid significant overlap between subsets.

We measured each dataset’s complexity by computing the ratio of the average Euclidean distance between a point and its nearest neighbor of the same class to the average distance between a point and its nearest neighbor of a different class, as shown in Table 1. A lower ratio suggests the dataset is more structured and likely to be separable with a simple decision boundary. A higher ratio means the relationships between different classes are more complex. For large datasets, it is computationally expensive to compute the distance between every point, so a random subsample of 6,900 elements was used. Further discussion of this estimate of data complexity can be found in other sources (Rong et al., 2021) and Section 4.2.

4.2 Results and Discussion

We now present results regarding the basic characteristics of inductive orientation vectors as well as some analysis of their values. For a depiction of the inductive orientation vectors relative to the Bank Marketing dataset, refer to Figure 2.

Inductive orientation vectors can differ greatly between datasets and between algorithms. For example, the inductive orientation vectors relative to the datasets Car Evaluation, Letter Recognition, Obesity, and Wine Quality are more sparse than those relative to Abalone, Bank Marketing, EEG Eye State, Random, and Shopper’s Intention. This sparseness occurs when an algorithm consistently predicts the same element in Ω even when trained on different subsets of the original training data. An algorithm that predicts in a similar manner across varying training subsets suggests that the algorithm is able to generalize to data outside its training subset, since each training subset is randomly generated and typically different.

Sparse inductive orientation vectors tend to correspond with datasets of lower complexity (cf. Table 1). This trend suggests algorithms do not capture the general trends in the data for datasets of high complexity, leading to more varied predictions and a spread out probability distribution. While this trend generally holds true, there are some deviations. For example, while the dataset Spam has a complexity ratio of 1.073 (which exceeds even the Random dataset), its inductive orientation vectors are more sparse than that of Random. This is most likely because the complexity ratio is not a perfect measure of dataset complexity. The same can be said for the spareness of inductive orientation vectors relative to the Car Evaluation dataset. Based on this ratio, points in the dataset Spam are closer to points of a different class than points of the same class. Even so, a relationship could still be detected in the data, causing the vectors to be more sparse than those of the Random dataset. A randomly generated dataset, however, clearly has no existing pattern, so its inductive orientation vectors are uniform. Since most other datasets match our expectations, we generally attribute sparseness in inductive orientation vectors to lower complexity in datasets.

We also confirm that different instances of the same algorithm typically have closely related inductive orientation vectors, despite having different hy-

Table 2: Machine learning algorithms and their parameters. Note that SGD Classifiers refer to classifiers that are optimized by Stochastic Gradient Descent. The base algorithm of SGD (Hinge) Classifier is a linear support vector machine and that of SGD (Log) is logistic regression.

Algorithm Name	**Abbreviation**	**Hyperparameters**
Adaboost	Adaboost	
Decision Tree	Dec. Tree	
Gradient Boosting	Grad. Boost.	
k-Nearest Neighbors	KNN	n_neighbors (*k*): 1, 3, 11, 15, 19, 25, 49, 51
Logistic Regression	Log. Reg.	max_iter:2000
Multi-layer Perceptron	MLP(1,3)	max_iter:2000, hidden_layer_sizes:(100), (150,100,50)
Guassian Naive Bayes	N. Bayes	
Quadratic Discriminant Analysis	QDA	
Random Forest	RF	n_estimators:1,5,10,25,100
Stochastic Gradient Descent Classifier	SGD (Hinge, Log)	max_iter:2000, loss:‘hinge’, ‘log’
Linear Support Vector Classifier	Linear SVC	max_iter:2000
Support Vector Classifier	SVC (Linear/RBF)	max_iter:2000, kernel:‘linear’, ‘rbf’

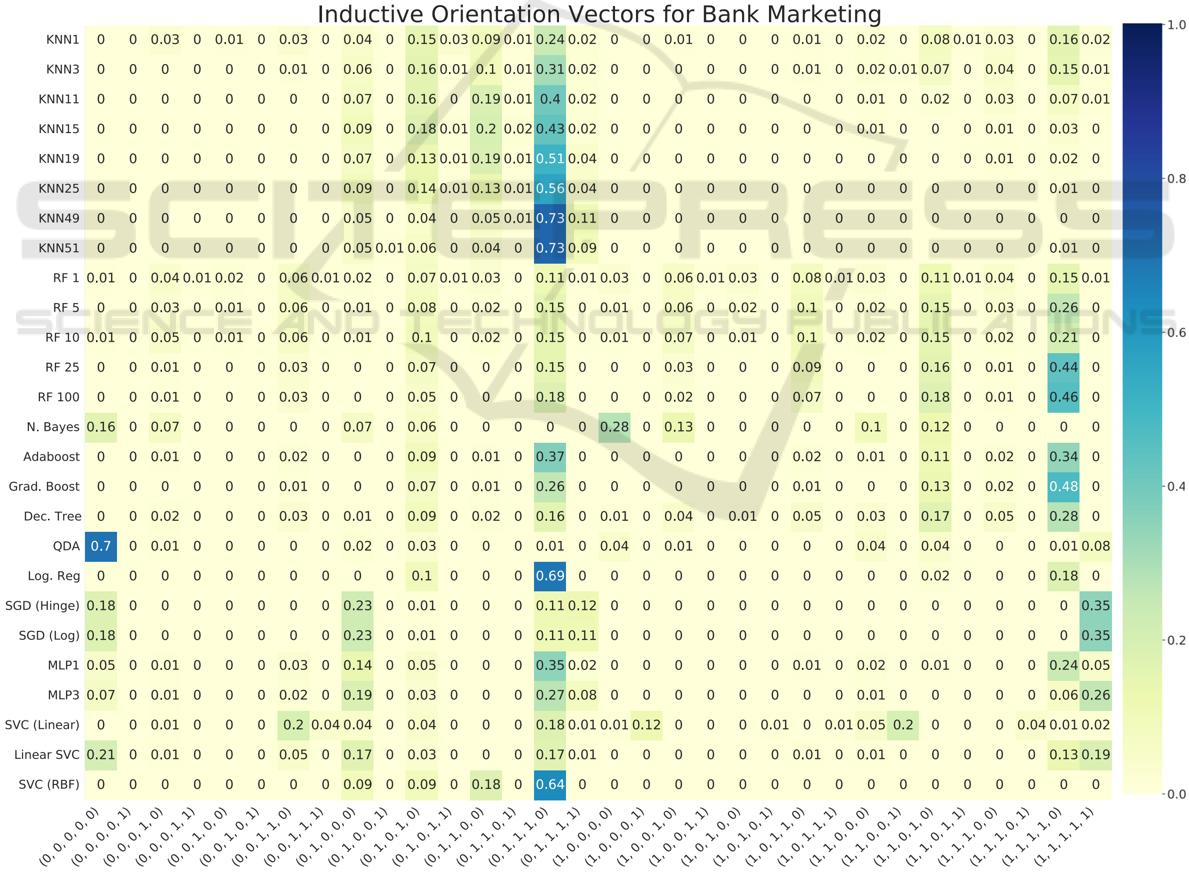


Figure 2: Inductive orientation vectors of algorithms trained on the Bank Marketing Dataset (Note that each 5-tuple corresponds to a particular way of classifying the elements in the holdout set. Each 5-tuple is in the search space Ω).

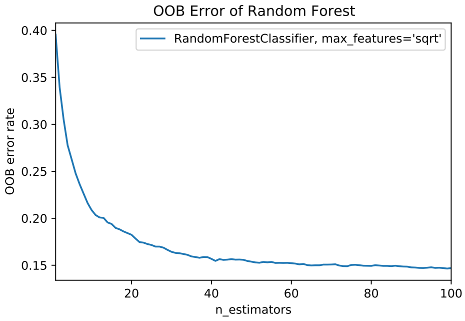


Figure 3: Out-of-bag error of Random Forest decreases as number of trees increase for the Bank Marketing Dataset.

perparameters. We see this with the $\overline{\mathbf{P}}_{\mathcal{D}}$ vectors relative to the Bank Marketing dataset, with the exception of the SVCs. For example, KNNs and Random Forests concentrate their probability mass on the labeling (0,1,1,1,0), SGD Classifiers concentrate their probability mass on (1,1,1,1,1), and MLPs concentrate their probability mass on (0,1,1,1,0). Although changing the hyperparameters does lead to changes in the probability distribution over Ω, the overall shapes of the distributions across different instances of the same algorithm are relatively stable, which means that $\overline{\mathbf{P}}_{\mathcal{D}}$ vectors of instances of the same algorithm are generally distinct from that of other algorithms. This suggests that, in many cases, changing the hyperparameters of an algorithm may not *fundamentally* change the algorithm.

Analyzing the $\overline{\mathbf{P}}_{\mathcal{D}}$ vectors of KNN relative to the Bank Marketing dataset, we notice a positive correlation between the number of neighbors k and the amount of probability mass concentrated at the labeling (0,1,1,1,0). As k increases from 1 to 51, the probability mass on (0,1,1,1,0) increases from 0.24 to 0.73, which means that KNN becomes more consistent in its predictions for larger values of k. This is because, as KNN considers increasingly larger neighborhoods of data to make its prediction, it becomes less affected by changes in data as well as noise. Taken to the extreme, when it considers the entire training subset as its neighborhood, KNN will be biased towards the majority class, if the KNN instance assigns uniform weighting to each neighbor. In contrast, KNNs with smaller values of k are highly sensitive to changes in data, causing variations in their predictions. This is reflected in the $\overline{\mathbf{P}}_{\mathcal{D}}$ vectors of KNN1 and KNN3 whose probability mass are distributed more evenly across Ω. These conclusions based on the inductive orientation vectors confirm that the choice of k plays a key role in how KNN will perform given a dataset.

A similar trend can be seen with the $\overline{\mathbf{P}}_{\mathcal{D}}$ vectors of Random Forest. Random Forest builds a forest of decision trees by selecting a random subset of training features and a different bootstrap sample of the training data for every tree. Thus, between different runs of Random Forest 1, which consists of only one tree, the bootstrap sample and training features selected are likely to differ, making the algorithm highly sensitive to variations in data. However, as the number of trees in a Random Forest model increases, it is able to generalize better and, thus, predict more consistently. From Figure 3, we see that the out-of-bag (OOB) error (a measure of accuracy of a Random Forest model) decreases as the number of trees increases, indicating that Random Forest models with more trees are more accurate. This correlates with our observation of the $\overline{\mathbf{P}}_{\mathcal{D}}$ vectors that as the number of trees increases, Random Forest concentrates increasingly more probability mass on the label it believes is correct. Thus, the inductive orientation vectors are consistent with existing knowledge of these algorithms and can provide new insights for algorithms that are less well-studied.

5 CLUSTERING

5.1 Method

Examining and comparing inductive orientation vectors can become difficult and impractical when the number of classes and/or holdout set size increases. Thus, a more effective way of identifying similarities between algorithms is by analyzing cluster plots.

Cluster plots are generated in three steps: clustering, dimensionality reduction, and plotting. We cluster at the original dimension $\mathbb{R}^{32}$ to preserve relationships between inductive orientation vectors. We used Meanshift as our clustering algorithm, avoiding the need to specify the number of clusters or set any hyperparameters; using Scikit-learn's defaults (Pedregosa et al., 2011) generally produced clusters consistent with our expectations. Another advantage of Meanshift is its ability to handle high-dimensional data using Locality Sensitive Hashing (Cui et al., 2011) which may be useful when working with inductive orientation vectors of higher dimensions, although it was not used in this study. To visualize the vectors, we reduce the vectors using Principal Component Analysis (PCA) to two dimensions. We use PCA instead of other dimensionality reduction methods, such as t-SNE and UMAP, because PCA is not stochastic, making it simpler to analyze. Having reduced the vectors to two dimensions, we plot them and label them according to how they were clustered in the original dimension. Note that the resulting clusters in the plots may diverge slightly from how we might cluster the vectors in two dimensions because the clusters were first formed in the original space.

5.2 Results and Discussion

Across cluster plots, we observe that instances of Random Forest, Decision Tree, Adaboost, and Gradient Boosting are consistently clustered together. This is expected because these algorithms are all well-known tree-based algorithms, extending and combining various implementations of decision trees. They also all have axis-aligned decision boundaries, as seen in Figure 5. Thus, it appears that similarities between their inductive orientation vectors are a result of their shared tree-based implementation. Furthermore, most or all of the instances of the KNN and Random Forest are clustered together in every plot. Existing research supports this relationship, as Lin and Jeon classify both algorithms as variants of weighted neighborhood schemes (Lin and Jeon, 2006). In Figure 5, we also see that they have somewhat similar decision boundaries. These findings suggest that cluster plots can uncover similarities between algorithms and, thus, are a useful tool for directing further research.

We note that datasets with more structure like Obesity and Letter Recognition have visually tighter clusters, compared to unstructured datasets like Random. Such analysis is limited since the scale of the axes is generated by PCA for a specific collection of inductive orientation vectors and, thus, not directly comparable between plots.

6 BENCHMARK DATA SUITE (BDS)

6.1 Method

While cluster plots are an effective way of visually comparing inductive orientation vectors, the Benchmark Data Suite (BDS) matrix is a more precise comparison metric. This metric is based on pairwise distances between inductive orientation vectors estimated relative to the same dataset.

Definition 6.1 (Pairwise Distance)**.** For a pair of inductive orientation vectors $\mathbf{P}_{\mathcal{D},\mathcal{A}}$ and $\mathbf{P}_{\mathcal{D},\mathcal{B}}$, the pairwise distance is the square of the Euclidean distances between the vectors. Formally,

$$d(\overline{\mathbf{P}}_{\mathcal{D},\mathcal{A}},\overline{\mathbf{P}}_{\mathcal{D},\mathcal{B}}) = \sum_{i=1}^{|\Omega|}(\overline{\mathbf{P}}^{[i]}_{\mathcal{D},\mathcal{A}} - \overline{\mathbf{P}}^{[i]}_{\mathcal{D},\mathcal{B}})^2 \tag{9}$$

where $\mathcal{D}$ is the data-generating distribution, i is the index, and $|\Omega|$ is the size of the search space and length of the inductive orientation vector. Note that $\mathcal{A}$ and $\mathcal{B}$ can be two separate learning algorithms or the same learning algorithm with different parameterizations.

An algorithm's BDS matrix is a matrix whose columns are made up of pairwise distances between the inductive orientation vector of the algorithm in question and a set of learning algorithms all trained on a particular dataset. The number of columns is equal to the number of benchmark datasets, and each column is relative to a different dataset. To balance the influence of each dataset while preserving the relative distance between algorithms' inductive orientation vectors for each dataset, we normalize each column of the BDS matrix by dividing by the corresponding maximum pairwise distance. These scaling factors are stored in a normalization vector. By multiplying the columns of a normalized BDS matrix with its corresponding scaling factor in the normalization vector, we can recover the unnormalized BDS matrix.

Given a set of learning algorithms **A** and a set of datasets **B**, we present the following pseudocode to find the BDS of a fixed algorithm $\mathcal{A}^* \in \mathbf{A}$.

Algorithm 2: Benchmark Data Suite.

```
for all $b \in \mathbf{B}$ do
    for all $\mathcal{A} \in \mathbf{A}$ do
        Compute the $\overline{\mathbf{P}}_{\mathcal{D},\mathcal{A}}$ when trained on subsets of the dataset b.
    end for
    for all $\mathcal{A} \in \mathbf{A} \setminus \mathcal{A}^*$ do
        Compute $d(\overline{\mathbf{P}}_{\mathcal{D},\mathcal{A}},\overline{\mathbf{P}}_{\mathcal{D},\mathcal{A}^*})$ and insert into matrix.
    end for
    BDS $\leftarrow$ Normalize the pairwise distances.
end for
Return BDS
```

6.1.1 Mean Pairwise Distance Vector

From the BDS matrix, we take the average across the columns to produce a mean pairwise distance vector. This aggregate vector reveals which relationships between inductive orientations hold across datasets and which hold true for only a dataset. We sort this mean distance vector by the average distance to more conveniently identify relationships between algorithms.

6.2 Results

Using the inductive orientation vectors presented in Section 4.2, we generate mean pairwise distance vectors for each of our algorithms.

The first two columns of Table 3 show the sorted mean pairwise distance vector of KNN11 which is mostly representative of other instances of KNN. We observe that, on average, the algorithms closest to

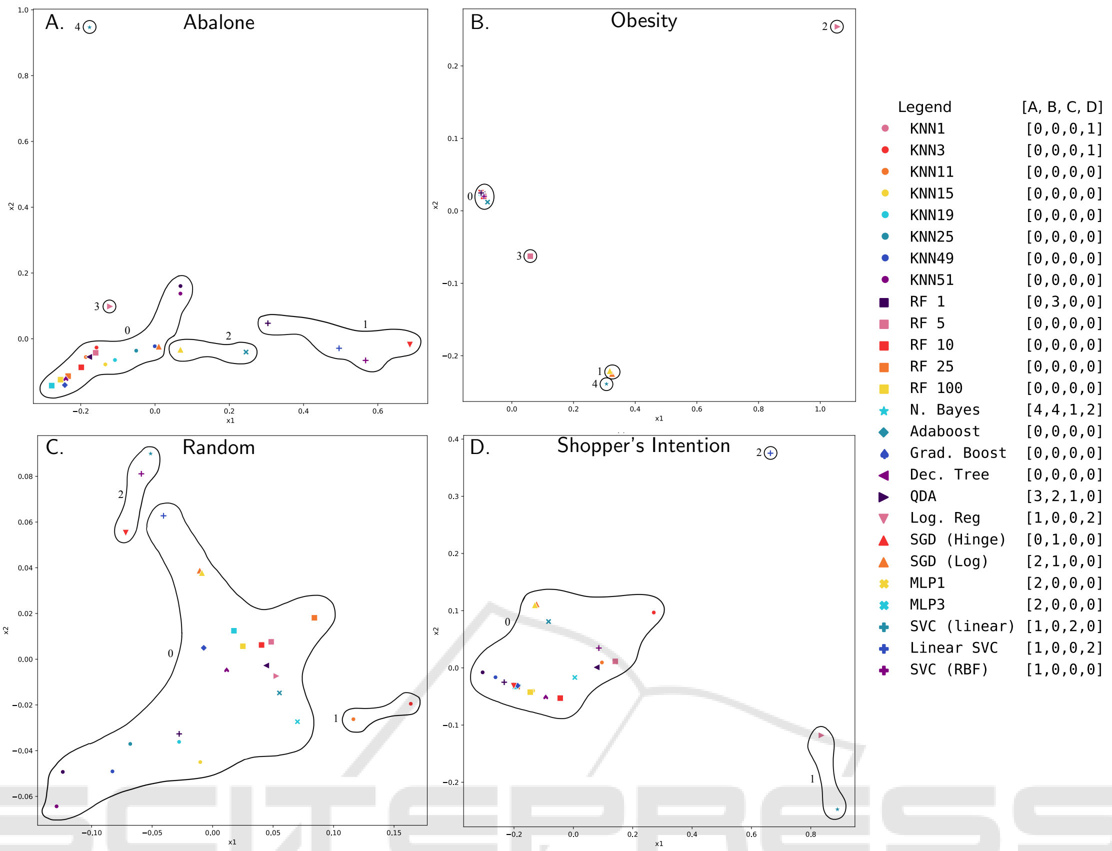
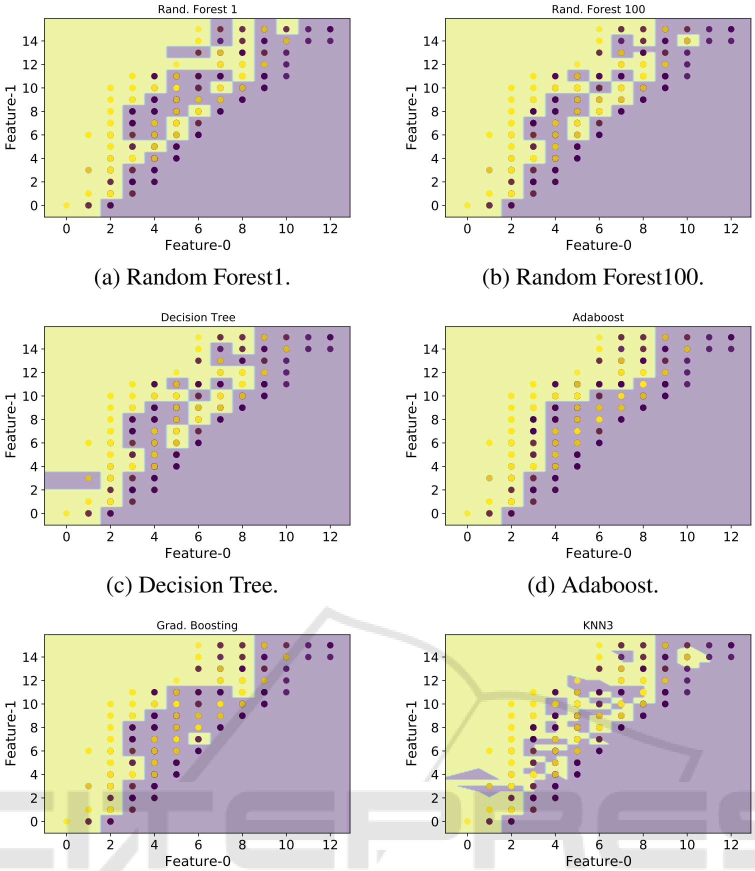


Figure 4: Cluster plots of inductive orientation vectors using the Meanshift algorithm to cluster in the original dimensions and PCA for dimensionality reduction. Boundaries around the plotted inductive orientations were added manually according to the labels generated by Meanshift.

the $\overline{\mathbf{P}}_{\mathcal{D}}$ of the KNN of interest are other instances of KNN, specifically those with similar values of k. Even so, not all instances of KNN may be close to the KNN of interest. For example, KNN1 is one of the furthest algorithms from KNN11. In general, instances of KNN, Random Forest and other tree based algorithms are the closest, followed by MLPs, and then a mixture of SVCs and SGDs. The furthest algorithms from KNN are QDA and Naive Bayes, which we also observed in Section 5.2.

Considering Random Forest 100, we find its nearest algorithms are other instances of Random Forests and Gradient Boosting. While both are types of boosting, Gradient Boosting is consistently closer to instances of Random Forest, excluding Random Forest 1, than Adaboost, suggesting greater similarities between the inductive biases of Gradient Boosting and Random Forest. Of all instances of Random Forest, Random Forest 1 is always furthest away from the Random Forest of interest. This is likely because Random Forest 1 makes its predictions based on a single tree, unlike the other more extensive instances of Random Forest. Besides a mixture of tree-based algorithms, KNNs are relatively close, followed by MLPs, Logistic Regression, and SVCs. Similar to the results from KNN 11, QDA and Naive Bayes are the farthest away from Random Forest 100. Although their mean distance vectors are not shown, Adaboost and Gradient Boosting are closer to each other than to Random Forest, likely because both are boosting algorithms.

Lastly, we notice that Naive Bayes, similar to SVC with a linear kernel, is relatively far from all other algorithms since the distances in the mean distance vector, on average, are much greater than those of other algorithms. Of all algorithms, QDA's inductive orientation is most similar to that of Naive Bayes; even so, their pairwise distance is still relatively large. Both Naive Bayes and QDA are probabilistic models leveraging Bayes' Theorem. However, Naive Bayes makes an assumption that all input features are conditionally independent, while QDA does not. Essentially, Naive Bayes is a simplified version of QDA (Ghojogh and Crowley, 2019), which explains the large pairwise distance when the input features are correlated and thus not conditionally independent.



(a) Random Forest1.

(b) Random Forest100.

(c) Decision Tree.

(d) Adaboost.

(e) Gradient Boosting.

(f) KNN3.

Figure 5: Example decision boundaries for two selected features for axis-aligned algorithms and KNN3. The data is generated using the first two features (x and y position of the box bounding the handwritten letter).

6.3 Discussion

Overall, the sorted mean pairwise distance vectors match our conclusions from analyzing the raw inductive orientation vectors and cluster plots. As mentioned previously, the mean distance vectors reveal that different instances of the same algorithm tend to be similar. Furthermore, cluster memberships are often reflected in the mean distance vectors. Most notably, Naive Bayes and QDA, both of which are often clustered on their own, also appear distant to other algorithms according to the pairwise distance vectors.

Calculated based on pairwise distances, the mean distance vector is limited in its ability to reflect changes in how probability mass is distributed over Ω. In Section 4.2, we noted an increase in the amount of probability mass concentrated on a particular labeling as the number of neighbors (for KNN) and the number of trees (for Random Forest) increased. This trend, revealing how different hyperparameter values affect the distribution over Ω, is difficult to observe when analyzing the sorted mean distance vector consisting of only pairwise distances. For example, consider again the mean distance vector of KNN11. Although we can conclude that KNN15, KNN19, and KNN25 are closer to KNN11 than KNN3, KNN49, and KNN51, we are unaware of why instances of KNN are not all close, in terms of pairwise distances. On the other hand, if we examine the raw inductive orientations, we see that, KNN49 and KNN51 concentrate more mass at (0,1,1,1,0) than KNN11, whereas KNN3 has less. Furthermore, KNN1, according to the mean distance vector, would be considered to be unrelated to KNN11 because its relationship to KNN11 is overshadowed by its large pairwise distance to KNN11. Thus, the mean distance vector obscures some subtle patterns in the algorithms' behavior. Even so, the limitations of mean pairwise vectors can be overcome when used in conjunction with cluster plots and analysis of the inductive orientation vectors themselves.

Regardless, the mean distance vector is a powerful tool that provides a ranking of how similar other algorithms are to an algorithm of interest based on

Table 3: Sorted mean pairwise distance vectors for KNN11, Random Forest 100, and Naive Bayes. Algorithms with smaller distances are sorted to be near the top.

KNN 11		**Random Forest 100**		**Naive Bayes**	
Algorithm Name	$d(\overline{\mathbf{P}}_{\mathcal{D}_\mathcal{A}},\overline{\mathbf{P}}_{\mathcal{D}_\mathcal{B}})$	**Algorithm Name**	$d(\overline{\mathbf{P}}_{\mathcal{D}_\mathcal{A}},\overline{\mathbf{P}}_{\mathcal{D}_\mathcal{B}})$	**Algorithm Name**	$d(\overline{\mathbf{P}}_{\mathcal{D}_\mathcal{A}},\overline{\mathbf{P}}_{\mathcal{D}_\mathcal{B}})$
KNN15	0.007	RF 25	0.007	QDA	0.272
KNN19	0.020	Grad. Boost.	0.038	RF 5	0.320
KNN25	0.037	RF 5	0.044	Linear SVC	0.344
RF 25	0.064	RF 10	0.053	MLP3	0.350
RF 5	0.072	KNN11	0.080	MLP1	0.353
RF 100	0.075	KNN15	0.082	SGD (Hinge)	0.360
Grad. Boost.	0.085	Adaboost	0.091	SGD (Log)	0.364
RF 10	0.098	KNN19	0.111	KNN11	0.365
Adaboost	0.102	Dec. Tree	0.126	RF 10	0.370
MLP1	0.115	KNN25	0.138	RF 25	0.374
KNN3	0.118	KNN3	0.143	KNN15	0.377
KNN51	0.119	MLP1	0.143	Dec. Tree	0.378
MLP3	0.123	MLP3	0.157	SVC (Linear)	0.379
KNN49	0.124	KNN51	0.227	KNN19	0.387
Dec. Tree	0.144	KNN49	0.239	Grad. Boost.	0.393
Log. Reg.	0.202	Log. Reg.	0.241	Adaboost	0.398
Linear SVC	0.233	Linear SVC	0.249	KNN3	0.402
SVC (RBF)	0.241	RF 1	0.253	KNN25	0.405
SGD (Hinge)	0.250	SGD (Hinge)	0.270	RF 100	0.409
SGD (Log)	0.251	SGD (Log)	0.272	RF 1	0.432
RF 1	0.262	KNN1	0.309	KNN51	0.446
KNN1	0.287	SVC (RBF)	0.309	KNN49	0.451
SVC (Linear)	0.362	SVC (Linear)	0.385	Log. Reg.	0.494
QDA	0.430	QDA	0.454	KNN1	0.502
N. Bayes	0.454	N. Bayes	0.463	SVC (RBF)	0.507

pairwise distances aggregated across datasets. Since the inductive biases of algorithms are captured in their respective inductive orientations (used in calculating pairwise distances), the mean distance vector allows us to gauge whether (and/or which) algorithms have similar inductive biases as the algorithm of interest. Accounting for all relationships across datasets, the mean distance vector is a reliable and quantitative way of comparing algorithms. This is because an algorithm must be consistently close, in terms of pairwise distance, to the algorithm of interest across all datasets for it to also be close in the resulting mean distance vector. Additionally, connections between algorithms, such as QDA and Naive Bayes, would be difficult to establish through a cluster plot, especially since the coordinate distances between algorithms can be distorted in the process of dimensionality reduction. These difficulties, however, are avoided when using mean distance vectors.

Lastly, although not explored in-depth here, the BDS matrix is a versatile tool of which many aspects of the BDS matrix can be modified, namely, the normalization method, set of benchmark datasets, set of algorithms, aggregation method, and even the pairwise distance metric. In these ways, the BDS matrix can be adapted for many other applications.

7 CONCLUSION

We develop a method to estimate the inductive orientation vector of a classification algorithm using a modified Labeling Distribution Matrix (LDM). The inductive orientation captures an algorithm's inductive bias relative to a certain dataset, providing an empirical basis for algorithm comparison. These vectors allow us to confirm several known similarities among existing algorithms, and this method therefore holds promise for characterizing novel algorithms. Inductive orientation vectors allow us to study the effects of algorithm hyperparameterization in a consistent way across algorithms. We explore inductive orientations visually using cluster plots and numerically using the Benchmark Data Suite, a particularly helpful foundation for studying algorithmic relationships across many diverse datasets.

Importantly, the LDM can used to calculate inductive orientation vectors for arbitrary black-box classification algorithms. While our estimation procedure was developed in this case for binary classification algorithms, regression and general search algorithms also produce inductive orientation vectors (Montañez, 2017). Adapting these methods for non-classification algorithms is the subject of future work. By comparing the biases of new algorithms to known biases of existing algorithms, we can provide a point of reference for comparing algorithm biases and inductive assumptions. Making biases explainable and measurable is of growing importance, given the increasing use of complex, overparameterized models such as deep neural networks. Inductive orientation vectors provide a quantitative tool for measuring and comparing inductive biases across algorithms.

ACKNOWLEDGEMENTS

This research was supported in part by the National Science Foundation under Grant No. 1950885. Any opinions, findings, or conclusions are those of the authors alone, and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Baati, K. and Mohsil, M. (2020). Real-Time Prediction of Online Shoppers' Purchasing Intention Using Random Forest. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 43–51. Springer.

Bashir, D., Montañez, G. D., Sehra, S., Sandoval Segura, P., and Lauw, J. (2020). An Information-Theoretic Perspective on Overfitting and Underfitting. *Australasian Joint Conference on Artificial Intelligence (AJCAI 2020)*.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165*.

Caruana, R. and Niculescu-Mizil, A. (2006). An Empirical Comparison of Supervised Learning Algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168.

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., and Reis, J. (2009). Modeling Wine Preferences by Data Mining from Physicochemical Properties. *Decision support systems*, 47(4):547–553.

Cui, Y., Cao, K., Zheng, G., and Zhang, F. (2011). An Adaptive Mean Shift Algorithm Based on LSH. *Procedia Engineering*, 23:265–269.

Dua, D. and Graff, C. (2017). UCI Machine Learning Repository.

Ghojogh, B. and Crowley, M. (2019). Linear and Quadratic Discriminant Analysis: Tutorial. *arXiv preprint arXiv:1906.02590*.

Gonen, H. and Goldberg, Y. (2019). Lipstick on a Pig: Debiasing Methods Cover up Systematic Gender Biases in Word Embeddings But do not Remove Them. *arXiv preprint arXiv:1903.03862*.

Lauw, J., Macias, D., Trikha, A., Vendemiatti, J., and Montañez, G. D. (2020). The Bias-Expressivity Trade-off. In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence - Volume 2*, pages 141–150. SCITEPRESS.

Lin, Y. and Jeon, Y. (2006). Random Forests and Adaptive Nearest Neighbors. *Journal of the American Statistical Association*, 101(474):578–590.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Mitchell, T. M. (1980). *The Need for Biases in Learning Generalizations*. Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ.

Montañez, G. D. (2017). The Famine of Forte: Few Search Problems Greatly Favor Your Algorithm. In *Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on*, pages 477–482. IEEE.

Montañez, G. D., Bashir, D., and Lauw, J. (2021). Trading Bias for Expressivity in Artificial Learning. In *Agents and Artificial Intelligence*, pages 332–353, Cham. Springer International Publishing.

Montañez, G. D., Hayase, J., Lauw, J., Macias, D., Trikha, A., and Vendemiatti, J. (2019). The Futility of Bias-Free Learning and Search. In *32nd Australasian Joint Conference on Artificial Intelligence*, pages 277–288. Springer.

Moro, S., Cortez, P., and Rita, P. (2014). A Data-Driven Approach to Predict the Success of Bank Telemarketing. *Decision Support Systems*, 62:22–31.

Osisanwo, F., Akinsola, J., Awodele, O., Hinmikaiye, J., Olakanmi, O., and Akinjobi, J. (2017). Supervised Machine Learning Algorithms: Classification and Comparison. *International Journal of Computer Trends and Technology (IJCTT)*, 48(3):128–138.

Palechor, F. M. and de la Hoz Manotas, A. (2019). Dataset for Estimation of Obesity Levels Based on Eating Habits and Physical Condition in Individuals from Colombia, Peru and Mexico. *Data in brief*, 25:104344.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Rong, K., Khant, A., Flores, D., and Montañez, G. D. (2021). The Label Recorder Method: Testing the Memorization Capacity of Machine Learning Models. In *The Seventh International Conference on Machine Learning, Optimization, and Data Science (LOD 2021)*.

Runarsson, T. P. and Yao, X. (2005). Search Biases in Constrained Evolutionary Optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(2):233–243.

Sandoval Segura, P., Lauw, J., Bashir, D., Shah, K., Sehra, S., Macias, D., and Montañez, G. D. (2020). The Labeling Distribution Matrix (LDM): A Tool for Estimating Machine Learning Algorithm Capacity. In *Proceedings of the 12th International Conference on Agents and Artificial Intelligence - Volume 2*, pages 980–986. SCITEPRESS.