Skinny-dip: Clustering in a Sea of Noise

Samuel Maurus, Claudia Plant
Helmholtz Zentrum München/Technische Universität München, University of Vienna, Vienna, Austria
samuel.maurus@helmholtz-muenchen.de, claudia.plant@univie.ac.at

ABSTRACT

Can we find heterogeneous clusters hidden in data sets with 80% noise? Although such settings occur in the real-world, we struggle to find methods from the abundance of clustering techniques that perform well with noise at this level. Indeed, perhaps this is enough of a departure from classical clustering to warrant its study as a separate problem. In this paper we present SkinnyDip which, based on Hartigan's elegant dip test of unimodality, represents an intriguing approach to clustering with an attractive set of properties. Specifically, SkinnyDip is highly noise-robust, practically parameter-free and completely deterministic. SkinnyDip never performs multivariate distance calculations, but rather employs insightful recursion based on "dips" into univariate projections of the data. It is able to detect a range of cluster shapes and densities, assuming only that each cluster admits a unimodal shape. Practically, its run-time grows linearly with the data. Finally, for high-dimensional data, continuity properties of the dip enable SkinnyDip to exploit multimodal projection pursuit in order to find an appropriate basis for clustering. Although not without its limitations, SkinnyDip compares favorably to a variety of clustering approaches on synthetic and real data, particularly in high-noise settings.

Keywords

Clustering; modality-testing; parameter-free; high-noise

1. INTRODUCTION

Depending on the nature of the measurements in a data set, it is not uncommon for patterns to be found swimming in a global sea of "clutter" or noise. In the clustering literature, however, we typically see methods showcased on data where the *majority* of objects shape the *signal*. Indeed, many clustering approaches are deficient of a clear "noise" concept. Our work hence begins with the question: what happens when there are *considerably* more "noise" objects in the data set than "clustered" objects?

KDD '16, August 13-17, 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2939672.2939740

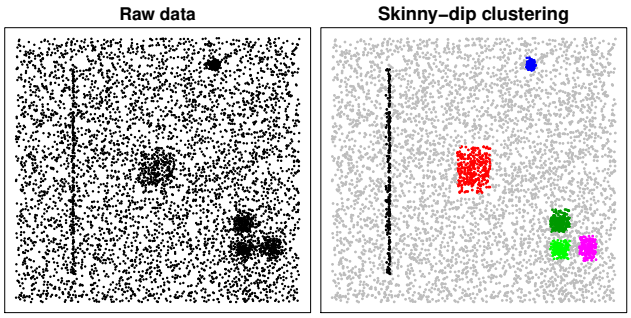

Figure 1: Our running example.

To tackle this question, we assume in this paper that the **clusters in a data set coincide with the modes of its multivariate distribution**, and design recursive heuristics based on the formal statistical *dip* test of unimodality in order to extract them. The resulting algorithm exhibits properties mirroring those of the dip: deterministic, parameter-free, highly robust to noise, free from the need to perform distance calculations, no strict model assumptions, and fast.

Consider with Figure 1 our illustrative running example[1]. It is clear that this is a rather turbulent data set: 80% of the objects are sampled from a uniform "noise" distribution. Despite this, our six clusters (Gaussians of varying size, a square, and a tall, thin rectangle) stand out as patterns. Our goal with this data set is to separate the signal (exposing each of the six shapes) from the noise.

An experienced data scientist would suspect centroid-based clustering approaches to be a poor choice here. They tend to lack a clear notion of "noise", so the Adjusted Mutual Information (AMI) value[2] (a clustering quality metric ranging from 0 at worst to 1 at best) of 0.18 for standard k-means, even when it is given the correct k, is unsurprising (visualization in supplement[1]). Model-based approaches like EM also fail because our cluster shapes do not follow a simple model (AMI of 0.22, Figure 2, reproducible[1]).

Suspecting that the clusters may have heterogeneous form, our scientist decides to begin with *spectral* clustering – a technique known to perform well with non-Gaussian clusters. The popular Self-Tuning [15] variant automatically detects six clusters and assigns labels as shown in Figure

[1]Supplement: github.com/samhelmholtz/skinny-dip
[2]Noise points are assigned a seventh class label. Normalized Mutual Information (NMI) measurements are provided for reference in the supplement.

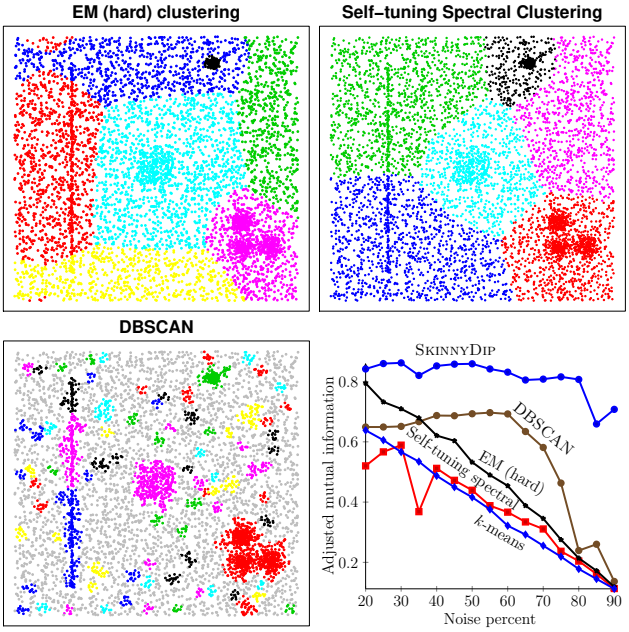

Figure 2: Further results for our running example.

2. The result, which again sees clusters composed purely of noise points, is not surprising given the known limitations of spectral clustering in the presence of significant noise.

Undiscouraged, our scientist wonders if density-based approaches can help. The popular DBSCAN method is known to robustly discover clusters of varying shape in noisy environments. After fine-tuning its parameters[3] (requiring graphical inspection), the DBSCAN result in Figure 2 is produced. This result is an improvement, however leaves much to be desired. DBSCAN detects 95 clusters in this case (AMI 0.24), primarily because there are various areas in which, through the randomness of noise, the local density threshold is exceeded. We need this threshold to prevent fragmentation of our tall rectangular cluster, but it causes our three neighboring Gaussians to merge. No parameter combination for DBSCAN can solve these problems.

Enter SkinnyDip[4], which pairs the formal dip test with efficient recursive heuristics. Requiring **practically no parameters** other than a threshold α for statistical significance (which for *all* our work is fixed at $\alpha = 0.05$), SkinnyDip detects all six clusters plus noise (back in Figure 1) to achieve an AMI of 0.81. SkinnyDip does this **deterministically and quickly**. It requires **no manual intervention, no convergence-based processes, no multivariate distance calculations, and no model-order selection techniques** (like the Bayesian Information Criterion).

Although we focus more on experiments in Section 6, we can briefly see in Figure 2 that SkinnyDip is able to defend its result the longest on this running-example data as we vary the noise percentage up to 90% (where nine of the ten-thousand data set objects are noise points).

Situations involving oddly-shaped clusters embedded in a sea of noise are not only of theoretical interest. For a **real-world example** we need look no further than Figure 3, which shows a simplified and annotated version of the North Jutland (Denmark) Road Network data from the UCI repository (data in the supplement). The data points represent road segments, which are generally denser in highly-populated areas (cities like Aalborg, Hjørring and Frederikshavn – with populations over 20,000). The majority of the road segments, however, could be termed "noise": long arterials connecting cities, or less-dense road networks in the relatively sparsely-populated countryside. We also appreciate the heterogeneity of cluster shapes, especially around the coastal and island regions, stemming from the topographical restrictions (lakes, shorelines) on habitation.

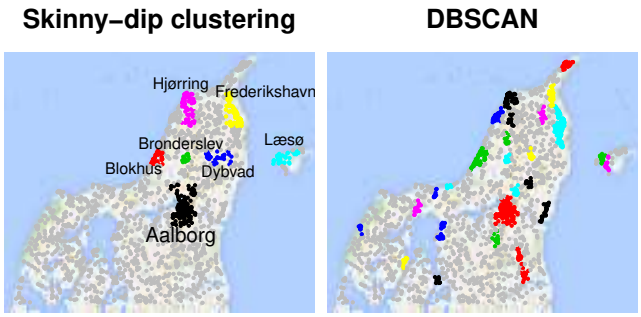

Figure 3: Clustering road segments in North Jutland. Background courtesy of Google Maps.

Without having to tune any parameters, SkinnyDip quickly produces an intuitive result in which the aforementioned three major cities of the region are found. The island of Læsø and particularly dense areas around Brønderslev (population over 12,000) are also found. The best result we were able to achieve by tuning the parameters of DBSCAN is displayed to its right ($minPts$= 10, ϵ = 0.03, replicable[1]). It detects over 20 additional and insignificant "countryside" (noise) clusters.

Contributions:
We consider the (often neglected) problem of clustering in extreme-noise settings, showing real examples.
We present SkinnyDip to cluster such data. SkinnyDip is deterministic and requires no distance computations, strict model assumptions, obscure parameters or convergence processes. As far as we know, it is the first clustering algorithm with this combination of properties.
We show that the core recursive heuristics on which SkinnyDip is based can be used to complement existing statistical tests of unimodality in order to answer the additional question of *where are the modes?* (not just: *is it multimodal?*) for arbitrary continuous samples.
We show the interpretation benefits that come from pursuing the maximum-dip directions in the data and clustering with them as a basis.

2. THE DIP TEST

Relatively unknown in the data-mining community, the *dip* [9] is a prominent formal statistical test for measuring the unimodality of a sample from a continuous random variable. Before introducing SkinnyDip, we briefly review the dip's fundamentals (rigorous treatment in [9]).

[3]Here $minPts$= 10 and $\epsilon = 0.019$.
[4]So named because it uses the dip test and is "skinny" in the sense that it is lean (few assumptions/parameters).

Consider Figure 4, which shows the histogram we obtain after projecting our running-example data[5] onto its horizontal axis. To its right is the sample's empirical cumulative distribution function (ECDF), denoted F. The colored regions correspond to the modes (**A**-**E**) of the sample, and we can see that F's gradient is larger where modes exist. Note that the histogram is for visual comparison only – the dip only needs F (which requires no bin-width parameter).

To measure the unimodality of the sample, the dip uses a common definition of unimodality: a distribution is unimodal if its ECDF takes a *convex* form up to its mode (or *modal interval* in general) and a *concave* form after it. The dip hence tries to assert that F is unimodal by fitting a piecewise-linear function G of this form onto it and taking a measure for its "goodness of fit". This statistic, the "dip" $d \in (0, \frac{1}{4}]$, is defined as the minimum achievable vertical offset for two copies of F (one above, one below, shown as dashed lines in the bottom graphic) such that G does not violate its unimodal rules ("convex then concave").

With our depicted analogy we imagine that the two offset copies of F form the edges of a racing track (birds-eye view). The racer, who follows the trajectory of G, *must* drive as though she were navigating a chicane, that is, "first steer to the left, then optionally straight, then to the right"[6]. The smallest possible width of the track which still enables the driver to follow these rules and remain on the track[7] is twice the dip statistic d. The farther F strays from unimodality, the larger the required width of the track.

[5]For visual clarity we use the version with 30% noise.

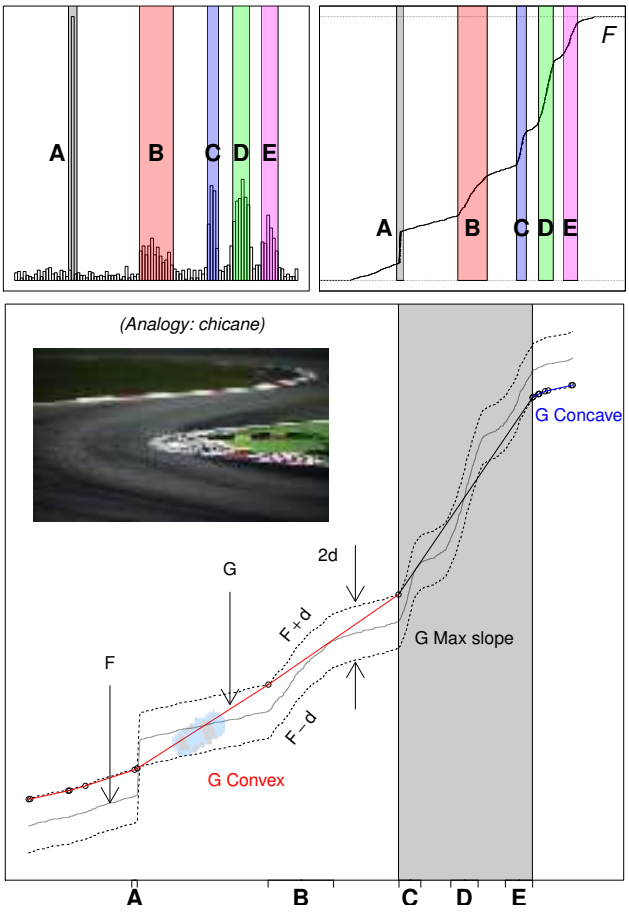

Figure 4: Histogram, ECDF (top) and dip solution (bottom) for the horizontal-axis projection of our running-example data.

After comparing d with that of a suitable reference unimodal distribution (null distribution), a p-value is reported for the null hypothesis H_0 ="F is unimodal". For our example we have $p < 10^{-10}$. Using the common threshold of $\alpha = 0.05$ we hence reject H_0 and favor the alternative ("F is *at least* bimodal").

The dip test hence gives us a measure for F's *departure from unimodality* – a useful insight if we assume that clusters are the modes of a distribution. Importantly though, the dip communicates one more piece of information: the location of the *modal interval* $[x_L, x_U]$. This is the interval in which G is prescribed maximum slope (shown as the gray region). As we will see shortly, SkinnyDip exploits this information when searching for all modes of a distribution.

Based purely on the geometry of the ECDF, the dip has convenient properties. Firstly, it is nonparametric – any continuous distributions with this abstract "convex then concave" shape are supported (Gaussian, beta, uniform and so on). The dip is invariant to shifting and scaling and, as we have hinted, is particularly robust to noise and outliers. It requires no input parameters. Finally, it is deterministic and *fast* – the standard algorithm on a sorted sample requires $\mathcal{O}(n)$ operations in the worst case. These properties mirror much of what we would like to see in a clustering technique.

Taken in its existing form, however, the dip test is of little use for clustering. The reasons for this are twofold: 1) it is only applicable to univariate data[8], and 2) it can only help us accept or reject the hypothesis that *the distribution is unimodal*[9], and thus cannot help us find the count and location of *all* modes in a distribution.

Despite this, we propose that the test is a promising and intriguing formal foundation from which we can commence exploring novel heuristics for the task of clustering.

3. DIP-BASED UNIVARIATE CLUSTERING

We take the first step towards our goal by considering a simpler problem: how can we cluster noisy *univariate* data? In this section we propose a solution in the form of a dip-based heuristic. Our approach is a recursive one, and at a high level is analogous to the traversing mechanism used by the dip in computing its statistic. In our case, however, it is used to "pick off" one mode at a time from the sample. We refer to this core part of SkinnyDip with the name UniDip and describe it now for our running example with the aid of Figure 4 and Algorithm 1.

We begin by "dipping" over the sorted univariate ($m = 1$) sample $x_1 \leq \ldots \leq x_n$ of size n (line 2). The dip result tells us two things. Firstly, from the p-value of $p < 10^{-10}$ we learn (based on the significance threshold of $\alpha = 0.05$) that the distribution has *at least* two modes. Secondly, we learn

[6]Strictly speaking, the steering must be such that the car's trajectory follows the exact restrictions imposed on G.

[7]The driver can leave the track inside the modal interval.

[8]A measure of distance on the space, as well as a means for generalizing empirical distribution functions, would be required for the multivariate case.

[9]The alternate hypothesis is hence the relatively unspecific statement of *the distribution is at least bimodal*.

the location of the modal interval $[x_L, x_U]$ for this sample. In the context of the dip this represents the interval for G's prescribed maximum constant slope. In our concrete case this first modal interval spans our modes **C**,**D** and **E** (the gray region in Figure 4). Intuitively, the dip lumps these into one modal interval because they lie close together in the ECDF (mathematically, doing so allows it to achieve a minimum d). We recurse into the modal interval to extract these three modes (line 5), which gives us three individual modal intervals $[x_{L_\mathbf{C}}, x_{U_\mathbf{C}}], [x_{L_\mathbf{D}}, x_{U_\mathbf{D}}], [x_{L_\mathbf{E}}, x_{U_\mathbf{E}}]$.

Knowing that there is at least one more mode, our search is not over. To find the next, we recognize that it must lie outside the gray interval $[x_L, x_U]$. At this point we heuristically assert that if a mode were to exist to its right, then dipping over $[x_{L_\mathbf{E}}, x_n]$ (that is, mode **E** and everything to its right) would yield a significant result. The choice of this interval is key. Consider if we were instead to dip over $[x_U, x_n]$ (everything to the right of the gray region). For our example, where that region only contains uniform noise, the dip would give a non-significant (unimodal) result. However, we would *equally* get a unimodal result if a single "cluster" (another Gaussian, for example) resided in that region. Clearly we need to be able to distinguish between these cases, so we include mode **E**. This way, if the dip concludes unimodality, we can assume that the single mode to which it refers is simply mode **E** again, and thus there is nothing "interesting" to its right. If the dip concludes multimodality, we know there *is* something interesting to its right. At the other end, if a mode were to exist to the *left* of x_L, then dipping analogously over $[x_1, x_{U_\mathbf{C}}]$ (this time including mode **C**) would yield a significant result.

For our example we find that the right part $[x_{L_\mathbf{E}}, x_n]$ is unimodal ($p \approx 0.97$, line 10), so we need not search right of x_U. The left part $[x_1, x_{U_\mathbf{C}}]$ *is* significantly multimodal ($p < 10^{-10}$, line 9), so we recurse into the region $[x_1, x_L]$ to continue searching. Through this recursive call the intervals $[x_{L_\mathbf{A}}, x_{U_\mathbf{A}}], [x_{L_\mathbf{B}}, x_{U_\mathbf{B}}]$ for modes **A** and **B** are obtained.

UniDip thus recursively exploits the dip's ability to 1) make a binary decision (unimodal or not), and 2) identify the primary modal interval. Assuming the procedure is provided with a sorted sample, its worst-case time complexity is in $\mathcal{O}(n \cdot k)$, where k is the number of modes found ($k \leq n$, typically $k \ll n$). For interest, Figure 5 demonstrates UniDip on a somewhat busier sample ($k = 17$).

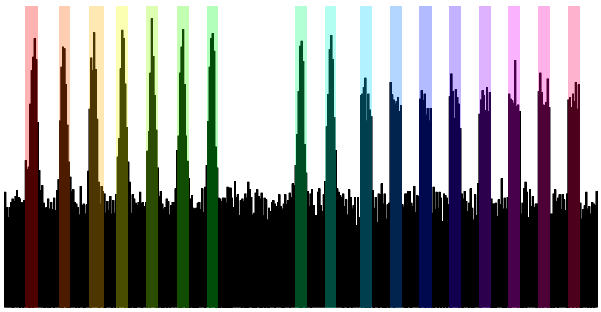

Figure 5: UniDip finds the 17 modes in this univariate sample (Gaussian and uniform clusters in a sea of 75% noise). Replication code in supplement.

input : Sorted univariate sample $x_1, \ldots, x_n$, significance level α, indicator $\delta = 1$ if $\vec{x}$ was identified as modal interval, else $\delta = 0$
output: Modal intervals $\mathcal{M} = \{[x_{L_1}, x_{U_1}], \ldots\}$
1 `/* Dip over the sample, obtaining modal interval estimate and p-value */`
2 $\{x_L, x_U, p\} \leftarrow$ `dip`$(\vec{x})$;
3 **if** $p > \alpha$ **then return** δ ? $\{[x_1, x_n]\} : \{[x_L, x_U]\}$;
4 `/* Recall using the modal interval to find the list` $\mathcal{M}_m$ `of modal intervals within */`
5 $\mathcal{M}_m \leftarrow$ `recall`$(\forall x_j : x_L \leq x_j \leq x_U,\ \alpha,\ \delta = 1)$;
6 `/* Check to the left and right of the modal interval, recursing to extract additional modes where appropriate */`
7 $u \leftarrow \min_{x_U \in \mathcal{M}_m} (x_U), l \leftarrow \max_{x_L \in \mathcal{M}_m} (x_L)$;
8 $p_l \leftarrow$ `dip`$(\forall x_j : x_j \leq u)$, $p_u \leftarrow$ `dip`$(\forall x_j : x_j \geq l)$;
9 $\mathcal{M}_l \leftarrow p_l \leq \alpha$? `recall`$(\forall x_j : x_j < x_L,\ \alpha,\ 0)$: $\emptyset$;
10 $\mathcal{M}_r \leftarrow p_u \leq \alpha$? `recall`$(\forall x_j : x_j > x_U,\ \alpha,\ 0)$: $\emptyset$;
11 **return** $\mathcal{M}_l \cup \mathcal{M}_m \cup \mathcal{M}_r$;

Algorithm 1: UniDip

4. THE MULTIVARIATE CASE

We wrap UniDip with another recursive heuristic, this time over the dimensions of the data space, in order to extract the modal hyperintervals from a continuous *multivariate* distribution. More specifically, we construct our hyperintervals one edge at a time, with each edge corresponding to a modal interval found from clustering the univariate projection of appropriately-filtered data. Again we explain using our running example (Figure 6 and Algorithm 2).

We begin by assuming for now that each cluster in the space forms a unimodal shape in all coordinate directions (like the rectangles and circles in our running example) – this assumption will be relaxed in Section 5. We consider the first direction in our data (line 4 of Algorithm 2, the horizontal-axis in our running example) onto which we project all data. Note that information may be lost in this step – considering Figures 1 and 4 we see that mode **D** (green) is formed from the projection of two separate clusters and much noise. We cluster this projection (lines 6 and 7) using our approach from Section 3, which yields the set of five modal intervals $\mathcal{M} = \{[x_{L_\mathbf{A}}, x_{U_\mathbf{A}}], \ldots, [x_{L_\mathbf{E}}, x_{U_\mathbf{E}}]\}$ for our example.

In Figure 6 we see the recursive step in action for the example modal interval $[x_{L_\mathbf{D}}, x_{U_\mathbf{D}}]$ (it is done analogously for the others). The values $x_{L_\mathbf{D}}$ and $x_{U_\mathbf{D}}$ represent the bounds of the hyperinterval in the first dimension. To find the bounds in the second dimension, we consider the subset of data objects whose coordinates in the first dimension lie within $[x_{L_\mathbf{D}}, x_{U_\mathbf{D}}]$ (line 10; that is, all objects within the green region in Figure 6). For these data points, we take a univariate projection onto the *second* direction (line 4, one recursive layer deeper, the vertical direction in our running example), the density of which is shown superimposed in Figure 6. We then cluster this projection, again using our approach from Section 3. This time we find two modal intervals (labeled D_1 and D_2 in Figure 6), from which we generate two modal hyperintervals (dark boxes in Figure 6), each with bounds of $[x_{L_\mathbf{D}}, x_{U_\mathbf{D}}]$ in the first direction. The procedure is repeated recursively for each of these hyper-

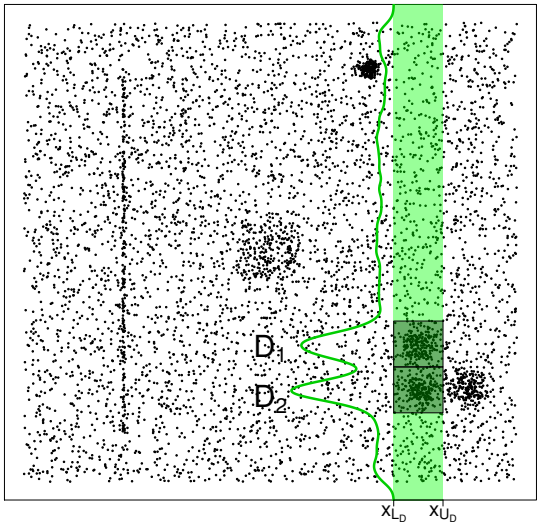

Figure 6: Dipping into the second dimension to find the modal hyperintervals.

input : Data $X \in \mathbb{R}^{n \times m}$, significance level α, bounds for current hyperinterval $\left(\vec{x}_L \in \mathbb{R}^d, \vec{x}_U \in \mathbb{R}^d\right)$ ($d = 0$ on first call)
output: Modal hyperintervals (clusters) $\{[\vec{x}_{L_1}, \vec{x}_{U_1}], [\vec{x}_{L_2}, \vec{x}_{U_2}], \ldots\}$

```
1  /* We are done if all dimensions checked   */
2  if d == m then return {[x_L, x_U]} ;
3  /* Otherwise, take next projection         */
4  x ← X_{·,(d+1)};
5  /* Sort/cluster this univariate projection
       (yields set of modal intervals M)       */
6  x ← sort(x) ;          /* (ascending sort).  */
7  M ←UniDip(x, α, δ = 0);
8  H ← ∅;
9  for [x_L, x_U] ∈ M do
10 |  Y ← [∀X_{i,·} : (x_L ≤ X_{i,(d+1)} ≤ x_U)];
11 |  H.add(recall(Y, α, x_L.push(x_L),
   |   x_U.push(x_U)));
12 end
13 return H;
```

Algorithm 2: SkinnyDip

intervals for the remaining dimensions in the space (in our running example we would be finished).

On line 6 we must sort the data before clustering. The overall worst-case run-time complexity of SkinnyDip hence depends on the run-time complexity to sort a vector of n real numbers. We note that the sort mappings can be computed for each feature vector in advance (an optimized implementation would then refactor line 6 to use global sort-mappings). We also observe that we never have more than m levels of recursion, and due to the strict partitioning the maximum count of data objects considered across SkinnyDip function invocations for any given recursion level is n. Assuming the use of a comparison-based sorting algorithm like merge- or heap-sort (worst-case run-time in $\mathcal{O}(n \cdot \log(n))$), we thus find SkinnyDip's **worst-case time complexity** as a full-space clustering technique to be in $\mathcal{O}(n \cdot \log(n) \cdot m + n \cdot m \cdot k)$, where m is the dimension of the data space and k is the maximum number of modes found from a call to UniDip. It is clear that SkinnyDip involves **no "convergence"-style processes** (e.g. meeting an accuracy threshold, or iterating until the local maximum of an objective function is obtained). Section 6.3 shows that *practical* run-time growth is *linear* in nm.

5. FINDING A SUITABLE BASIS

In the previous section we assumed that each cluster forms a unimodal shape in *all* coordinate directions. This is often unrealistic – many features of a data set may be irrelevant to a clustering. In this section we show how we can use the idea of *multimodal projection pursuit* to find an appropriate basis in which to cluster. Our dip-based search strategy complements our dip-based algorithm SkinnyDip. It is motivated by the insight that *the directions which depart the most from unimodality are promising when searching for a clustering.* To be clear, we are *not* searching for arbitrary subspaces *per cluster*, but rather a *single basis* in which clustering is to be performed (discussed further in Section 7).

We thus seek directions in our data that *maximize* the dip, thereby generating a basis in which the SkinnyDip algorithm can exploit these maximal-multimodal directions in order to find clusters. Fortunately, continuity properties of the dip enable us to exploit gradient ascent to help find the directions we seek. Specifically, for m-dimensional data, Krause [11] showed that the dip varies smoothly with respect to a changing projection vector $\vec{a} \in \mathbb{R}^m$. If we denote by d the dip of the projection of our data $D \in \mathbb{R}^{n \times m}$ with respect to $\vec{a}$, the partial derivatives[10] of d with respect to the elements of the projection vector are given by:

$$\frac{\partial d}{\partial a_i}(\vec{a}) = \begin{cases} -\frac{i_3 - i_1}{n} \cdot \frac{\vec{a} \cdot \left(\beta_i \vec{\gamma} - \gamma_i \vec{\beta}\right)}{(\vec{a} \cdot \vec{\gamma})^2} & \text{if } \eta > 0, \\ \frac{i_3 - i_1}{n} \cdot \frac{\vec{a} \cdot \left(\beta_i \vec{\gamma} - \gamma_i \vec{\beta}\right)}{(\vec{a} \cdot \vec{\gamma})^2} & \text{if } \eta \leq 0, \end{cases} \tag{1}$$

where $\vec{\beta} = (\beta_1, \ldots, \beta_m) = D_{i_2 \cdot} - D_{i_1 \cdot}$, $\vec{\gamma} = (\gamma_1, \ldots, \gamma_m) = D_{i_3 \cdot} - D_{i_1 \cdot}$ and $\cdot$ for two vectors is their scalar product. The indices i_1, i_2, i_3 correspond to the location of the so-called *modal triangle*, the height h of which satisfies $h = 2d$. The value $\eta = i_2 - i_1 - (i_3 - i_1) \left(\vec{a} \cdot \vec{\beta}\right) / (\vec{a} \cdot \vec{\gamma})$ simply accounts for the case when the modal triangle points in the negative direction. See [11] for full details.

Of course, gradient ascent will not identify the global maximum-dip direction in the general case. Indeed, looking at Figure 8, which shows the dip variation for projection vector angles in the range $[0, \pi]$ for our running example data, we see valleys with numerous maxima. Although the surface becomes more smooth with increasing n, it is clear that gradient ascent would profit from a good initialization.

We therefore require a mechanism for choosing a *starting point* for gradient-ascent. Simple random sampling, that is, randomly generating a number w of unit vectors in the space and greedily taking the one giving a data projection with maximum dip, is clearly a straight forward option here that is trivial to support. However, it is undesirable to introduce an element of randomness into our work at this stage

[10] We note that a syntax error appears to be present in the original paper. Our version contains the suggested correction.

(SkinnyDip and UniDip are deterministic and involve no random-sampling). We hence sample from the nodes of a *sparse grid* [3, 13] – a structured and scalable way to span high-dimensional spaces. Sparse grids are based on a multiscale basis which is derived from a one-dimensional multiscale basis by a tensor product construction. Discretizing a function on a sparse grid requires only $\mathcal{O}(q \cdot (\log q)^{m-1})$ nodes, where q is the number of grid points in one coordinate direction at the boundary. Unlike simple random sampling, the nodes of a sparse grid are generated deterministically. Furthermore, although we cannot make any theoretical guarantees that the use of sparse grids will in *every* case produce more favorable candidate nodes for arbitrary dip surfaces (we may well "get lucky" using simple random sampling), reproducible results in our supplement[1] show that their use directly leads to better clustering results for *all ten* of the real-world data sets we present in Section 6.2. On a more general level, systematic experiments on numerous high-dimensional benchmark optimization surfaces[1] (such as the Rosenbrock and Michalewicz functions in Figure 7) additionally show the benefits of this sampling strategy.

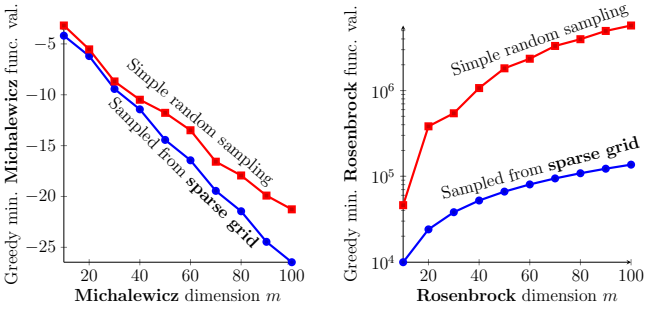

Figure 7: Simple random sampling versus sparse-grids sampling: a comparison of the greedy minimum value obtained over an equally-sized (large) sample by varying function dimension. Note that the global minimum value of the Michalewicz function decreases with increasing dimension.

With SparseDip (Algorithm 3) we hence sample from a sparse grid rather than randomly. As we only require unit vectors and can exploit symmetry, we can for an m-dimensional data space construct a sparse grid in $[0, \pi]^{m-1}$ "angle" space (line 6) in order to obtain sparsely-positioned unit vectors lying on a $(m-1)$-sphere in m-dimensional space (line 8, where SphToCart converts spherical coordinates to Cartesian). Figure 9 shows a concrete example for $m = 3$.

To maintain efficiency, our implementation of SparseDip uses a bound for the number of grid nodes to evaluate. When asked to generate a grid of dimension m, the method SgGen returns the regular sparse grid over $[0, \pi]^m$ of level l whose node-count is closest to w (i.e. $\mathcal{O}(w)$, line 6). In general, exploring higher-dimensional spaces would call for the use of a higher value for w, however for simplicity and efficiency we use a default of $w = 1000$ for all our work.

For each unit vector we project the data and compute the dip (requiring us to sort the sample), greedily taking the maximum as the starting point for gradient ascent (line 10). The worst-case run-time complexity for finding the starting point is easily shown to be in $\mathcal{O}(n \cdot \log(n) \cdot m \cdot w)$.

Having greedily selected a starting point (the sparse-grid-based unit vector with largest dip), we proceed with gradient ascent (line 11) on the hypersphere to "fine tune" this vector, obtaining a unit vector with a locally-maximum dip. This vector becomes the first in the set $\mathcal{B}$ of orthonormal vectors defining the basis in which we plan to cluster. The basis is expanded one direction at a time by repeating this process on the data's projection to the orthogonal complement of this basis (line 4), in which we find the next direction (orthogonal to our existing directions) maximizing the dip. The process terminates when a maximum-dip direction is found for which H_0 cannot be rejected (i.e. we can't find an "interesting" direction in the remaining space, line 13), or if the basis obtains the cardinality m (the dimension of the original data). SkinnyDip (the full-space clustering technique from Section 4) can then be applied to the data's projection onto the found basis. In the case where there are as many "interesting" directions in the data as original features m, the overall worst-case run-time complexity for finding a basis using SparseDip is in $\mathcal{O}(n \cdot \log(n) \cdot \frac{m(m-1)}{2} \cdot w)$.

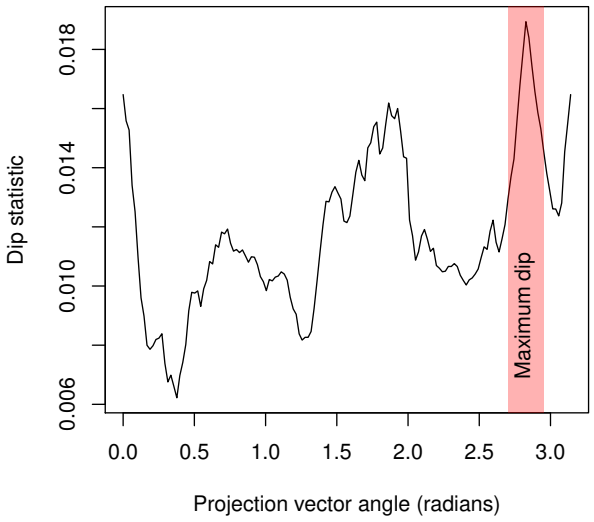

Figure 8: Dip variation against angle of the projection vector (running-example data).

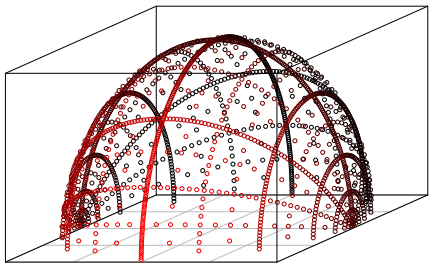

Figure 9: The two-dimensional regular sparse grid of level 8 can be used to define sparsely-distributed unit vectors lying on the 2-(hemi)sphere.

6. EXPERIMENTAL EVALUATION

We now turn to a practical evaluation of SkinnyDip, comparing it to state-of-the-art *automated* representatives from different families (in Section 7 we elaborate on the moti-

input : Data $D \in \mathbb{R}^{n \times m}$, significance level α, sparse grid point count w.
output: Orthonormal basis set $\mathcal{B}$ s.t. $|\mathcal{B}| \leq m$.

$\mathcal{B} \leftarrow \emptyset$;
while $|\mathcal{B}| < m$ **do**
  /* Project data to orthogonal complement of our existing basis */
  $D_c \leftarrow D \cdot \left(\mathcal{B}^{\perp}\right)$; /* ($D_c \in \mathbb{R}^{n \times (m-|\mathcal{B}|)}$) */
  /* Sparse grid points over angle space */
  $\left\{\vec{\theta}_1, \ldots, \vec{\theta}_{\mathcal{O}(w)}\right\} \leftarrow$ SgGen$(m - |\mathcal{B}| - 1, w)$;
  /* Cartesian candidate vectors (lengths $m - |\mathcal{B}|$) for gradient ascent */
  $\left\{\vec{v}_1, \ldots, \vec{v}_{\mathcal{O}(w)}\right\} \leftarrow$ SphToCart$(\left\{\vec{\theta}_1, \ldots, \vec{\theta}_{\mathcal{O}(w)}\right\})$;
  /* Gradient ascent starting from candidate with maximum dip */
  $\vec{v}_m \leftarrow \underset{\vec{v} \in \left\{\vec{v}_1, \ldots, \vec{v}_{\mathcal{O}(w)}\right\}}{\arg\max}$ (dip(sort$(D_c \cdot \vec{v})$));
  $\vec{v}_m \leftarrow$ GradAsc$(\vec{v}_m, D_c)$; /* See [11] */
  /* Done if result not significant */
  $p \leftarrow$ dip(sort$(D_c \cdot \vec{v}_m)$);
  if $p > \alpha$ **then return** $\mathcal{B}$;
  $\mathcal{B}$.push$(\vec{v}_m)$;
end
return $\mathcal{B}$;

Algorithm 3: SparseDip

vation for our selection). We begin with RIC [1], which fine-tunes an initial coarse clustering based on the Minimum Description Length principle (a model-selection strategy based on balancing accuracy with complexity through data-compression). PgMeans [8] and DipMeans [10] are wrappers for automating EM and k-means respectively. Self-tuning spectral clustering (STSC) [15] is the popular automated approach to spectral clustering from Zelnik-Manor and Perona. With DBSCAN [5] we have the popular member of the density-based family. Sync [2] is a more recent automated algorithm which combines density-based concepts with a novel approach inspired by phase-oscillators.

We thank various peers[11] for sharing code implementations. In our supplement[1] we share our code as well as all data sets, generative models and results which follow.

6.1 Synthetic Data

We choose our generative model such that the data sets exhibit the challenging properties on which we focus. Specifically, we desire high levels of noise around clusters of varying shape that may only be separated by a small distance. That is, our generative model creates data with properties similar to our running example. By default, we choose six clusters of 200 objects each in three dimensions. Half of the clusters are Gaussian with a standard deviation of 0.005. The remaining half are rectangular (one thin side of width 0.005; the remaining sides of width 0.25). The centers of the three Gaussians are pairwise-separated by a distance of 6σ (similar to the bottom Gaussians in our running example), and are otherwise randomly placed in the space. The three rectangles are similarly separated by a distance of 0.03. By default, 80% of the objects in the data set are "noise", that is, objects which have been sampled from the uniform distribution over the unit hypercube. With our default parameters, we hence obtain data sets with $n = 6000$ and $m = 3$.

Relative to the default parameters, we systematically vary the number of clusters $k = \{3, 6, \ldots, 15\}$, the noise percentage $\eta = \{20, 25, \ldots, 90\}$ and the dimensionality $m = \{2, 3, \ldots, 8\}$. For each parameter combination we randomly generate 20 data sets (560 data sets in total).

SkinnyDip uses its default value of $\alpha = 0.05$ in all cases. We likewise use the default parameter values for the provided implementations of the comparison algorithms. To automate DBSCAN, we fix *minPts*= 10 and run DBSCAN for all $\epsilon = \{0.01, 0.02, \ldots, 0.2\}$, reporting the best AMI result from these parameter combinations in each case.

Figure 10 presents the results for SkinnyDip and the comparison techniques on these data. A point in a plot series is the mean AMI value from the 20 data sets corresponding to that parameter combination, and the error bars span one standard deviation in each direction (omitted on every second point to reduce clutter). In fairness to the techniques which have no concept of noise (e.g. centroid-based techniques), the AMI only considers the objects which truly belong to a cluster (non-noise points). The results are discussed in Section 7.

6.2 Real-World Data

In Table 1 we present results (AMI value) for **ten real-world data sets** of varying size from the UCI and Comprehensive R Archive Network (CRAN) repositories. These classification-style data are often used to quantitatively evaluate clustering techniques in real-world settings; however it should be noted that *every* point is assigned a semantic class label (none of the data include a noise label). For this reason we run the k-means iteration (based on Euclidean distance) on the final SkinnyDip result to assign any detected noise objects to a "true" cluster.

As some of the data exhibit high dimensionality, SkinnyDip makes use of the SparseDip basis-search (Section 5), noting of course that it can yield the full space when appropriate. Particularly for the high-dimensional data sets, our use of SparseDip is somewhat unfair on the methods which are not able to perform such a search, and for this reason we additionally evaluate the recent FOSSCLU [6] method here. FOSSCLU is a state-of-the-art technique which, like SkinnyDip, searches for a *single* basis in which to perform clustering in an automated manner. For DBSCAN we automate in the same way as for our synthetic experiments, reporting the best AMI value achieved from $\epsilon \in \{0.05, 0.10, \ldots, 5.00\}$ in each case[12]. RIC is always provided with a starting value of $k = 10$ (for all data sets we have $k \leq 10$), which it seeks to refine through cluster-merging. For all of the other automated techniques we use the default parameters as given by their provided implementations.

In a *quantitative* sense, SkinnyDip's results are promising compared to the competition. It achieves the strongest AMI value on seven of the ten data sets, and is ranked second on

[11] Warm thanks to Greg Hamerly, Christian Böhm and Junming Shao for PgMeans, RIC and Sync respectively. For each of DipMeans and STSC we used the implementation from the author's website.

[12] Here our range for ϵ needs to be larger because the data are in general of higher dimension than in the synthetic cases. *minPts* was again fixed at 10 (much smaller than any data set cluster).

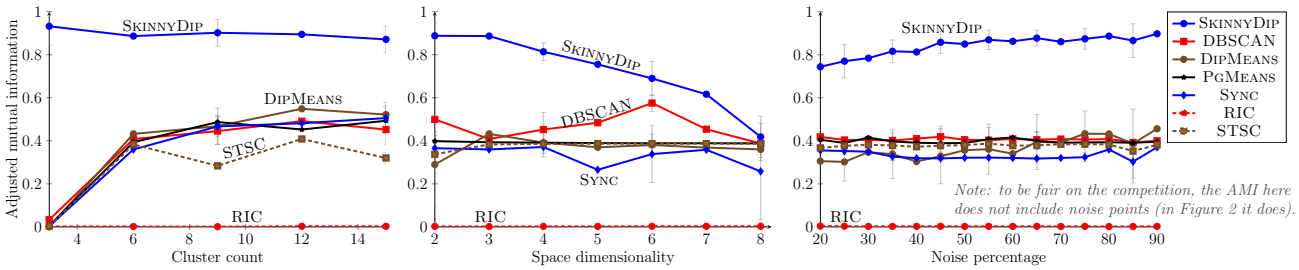

Figure 10: Synthetic experiment results.

two of the remaining sets. On the Pen Digits data set it ranks fourth.

As a *qualitative* supplement, we investigate two of the results in detail. Figure 11 depicts the case of the Whiteside data. Recorded at a house in England, the horizontal and vertical axes represent weekly average external temperature and gas consumption respectively. The class labels record whether cavity-wall insulation had been installed (blue) or not (red). Intuitively we expect a negative association between gas consumption and temperature (heating costs are lower when it is warmer), and also see that the insulation factor introduces a negative vertical offset (the use of insulation is associated with a reduced overall gas consumption). From a clustering perspective we can view this data as two elongated and neighboring clusters, which SkinnyDip is able to automatically detect (with an optimal AMI value of 1) by finding the direction of maximum dip and clustering in the corresponding projection.

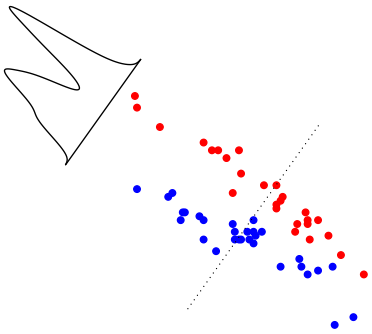

Figure 11: The projection vector and corresponding univariate distribution found by SkinnyDip on the Whiteside data. Subsequent SkinnyDip clustering yields the optimal result.

SkinnyDip's subspace-identification for the Image Segmentation data is also noteworthy. It finds a three-dimensional space for clustering, the projections for two directions of which are displayed in Figure 12. The first direction has a high loading for the feature *value-mean* and a large *negative* loading for *short-line-density-2*. It cleanly separates the *sky* images, which is sensible because images of sky and clouds typically contain few lines. The second direction has large positive and negative loadings for the features *hue-mean* and *exblue-mean* respectively. It cleanly separates the *grass* images, which we intuitively expect to have a large hue and significantly lack the color blue.

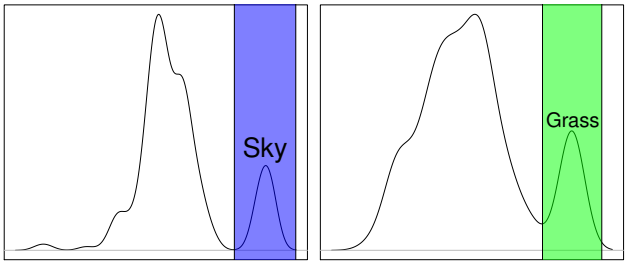

Figure 12: Densities along two of the basis directions found for the image-segmentation data.

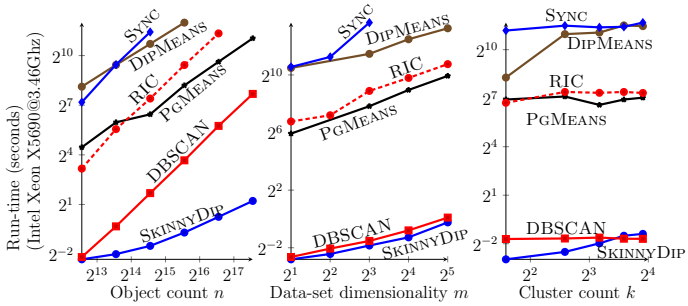

Figure 13: Run-time for varying n, m and k.

6.3 Run-Time Experiments

We support our stated theoretical run-time bounds with measurements and comparisons on synthetic data with scaled n, m and k. SkinnyDip is implemented in R, and we use the R implementation[13] of DBSCAN. RIC and Sync were provided in Java, and the remaining algorithms were provided in Matlab. Given the heterogeneous implementations (interpreted vs compiled languages, and varied levels of optimization), we focus only on the asymptotic trends.

We use the same generative model as that described in Section 6.1. By scaling the number of objects per cluster we are able to scale n (the noise percentage is fixed at 80%).

We see linear run-time behavior in m for all algorithms (except perhaps Sync). Run-time growth for increasing k is between zero (e.g. DBSCAN) and linear. SkinnyDip's run-time grows sub-linearly with k (linear growth is only expected for pathological cases).

Interestingly, the SkinnyDip results show a practically *linear* growth in n. Our theoretical bound has an additional

[13]Package *dbscan* on CRAN.

Data set (n, m)	Whites. 56,2	Derm. 358,34	Im-seg. 210,19	Maps 429,3	Seeds 210,7	Coalit. 340,7	Motor 94,3	Soy. Lg. 266,62	Prest. 98,5	Pen dig. 7494,16
SkinnyDip	**1.000**	**0.638**	**0.552**	**0.804**	0.534	**0.579**	**1.000**	0.554	**0.540**	0.647
STSC	0.272	0.296	0.211	0.434	0.436	0.297	**1.000**	0.238	0.311	**0.769**
PgMeans	0.000	0.571	*0.000	0.801	0.043	0.317	**1.000**	*0.000	0.365	0.619
DipMeans	0.000	0.296	0.210	0.066	0.000	0.313	**1.000**	0.319	0.448	0.605
Sync	0.122	0.608	*0.000	0.649	0.483	0.340	0.727	0.005	0.517	0.537
DBSCAN	0.254	0.620	0.399	0.754	0.453	0.385	**1.000**	0.294	0.479	0.692
RIC	0.000	0.053	0.444	0.001	0.000	0.575	0.536	**0.611**	0.000	0.002
FOSSCLU	0.493	0.324	0.000	0.536	**0.566**	0.271	0.727	0.000	0.410	0.694

Table 1: Real-world results (AMI value). *Three failed due to non-trivial bugs in provided implementations.

logarithmic factor which we expected to materlialize from the need to sort. This curious result was explained by isolated tests of the optimized R sorting routine used in our SkinnyDip implementation: it appears to manage ("near")-linear run-time growth in many "average" practical cases.

We see *quadratic* growth in n for many of the comparison techniques (e.g. DBSCAN and Sync). We note that the use of appropriate data structures (e.g. *kd*-tree) can reduce the growth of DBSCAN to sub-quadratic in n. The "slowest" of the competition based on our measurements is STSC: the computation of its locally-scaled affinity matrix is very expensive. The run-time growth of the provided implementation is cubic in n, rendering it intractable for us to collect enough measurements on the scale used.

Overall, the practical run-time growth of SkinnyDip is within our predicted bounds. These low bounds, coupled with its freedom from convergence-based processes, renders it a strong candidate for work on the large-scale.

7. DISCUSSION AND RELATED WORK

We now turn to a discussion of SkinnyDip and its results in the context of related work. Many papers have been published on the broad subject of clustering, so we must filter known techniques considering the properties of our proposed technique and the kinds of data on which it performs well.

Perhaps the most practically-relevant criteria by which to filter algorithms is the nature and count of the required parameters. SkinnyDip is practically parameter-free, so we select prominent representatives that require neither obscure parameters nor additional processing (e.g. hierarchical methods require some additional mechanism for transforming a dendrogram to a flat clustering). It is in this light that we find PgMeans [8], a wrapper for automating EM clustering. It uses the Kolmogorov-Smirnov statistical test for selecting the most appropriate model, and was shown to outperform the earlier techniques G-Means [7] and X-Means [12]. We also find the recent DipMeans [10] technique – a wrapper for automating techniques of the k-means family. DipMeans is one of the few data-mining contributions that has explored the use of the dip test, albeit in a different manner. In DipMeans, data points are *viewers* and the dip is used to test whether or not the distribution of distance measurements (e.g. Euclidean) from a viewer is multimodal. Information about the cluster structure can be extracted by these means, however the technique suffers from the *viewer-selection* problem which requires enumeration of all data set objects and the computation of multivariate distances for each (time complexity unfortunately quadratic in n). It is also clear that such techniques do not have a clear concept of noise, but even if we ignore noise in our quality measurements (as in our synthetic experiments) we still see these techniques fail because of model- and distance-assumptions.

We also find Sync [2] and RIC [1] as two further approaches to automated clustering. Sync regards each object as a kind of "phase oscillator" which can change its position as clusters (the count of which is determined automatically through the MDL principle) converge toward so-called *local perfect synchronization* over time. Sync uses ideas from density-based clustering (like the ϵ-neighborhood) and exhibits some the same weaknesses (reliance on distance calculations, standard time complexity in $\mathcal{O}(n^2)$). RIC is designed to be a wrapper for arbitrary clustering techniques, starting with a preliminary clustering and refining it based on information-theoretical concepts. Unfortunately it meets difficulties when the amount of noise in the data set is non-trivial: for almost all our experiments the number of clusters detected was one (giving an AMI value of zero).

In the area of spectral clustering, we compare to the prominent *self-tuning* method (STSC) of Zelnik-Manor and Perona [15], which is automatically able to select the appropriate scale of analysis and the number of clusters using a *local* scale approach with affinities. STSC obtained a good result on the Pen-Digits data set, but was otherwise unable to differentiate itself from the other techniques (particularly on the challenging, highly-noisy synthetic cases).

The density-based technique DBSCAN is not strictly automatic (arguably unfair on the other techniques), but is nonetheless a useful comparison because is has a clear concept of noise. DBSCAN requires the parameters ϵ and *minPts*. Our running example showed that the well-known limitations of DBSCAN (finding clusters of varying density) are magnified when the data set also contains large amounts of noise. Run-time can additionally be a limiting factor for implementations without optimized index structures.

We are aware of two techniques which specifically focus on high levels of noise. In [4] the authors consider the problem of detecting minefields and seismic faults from "cluttered" data. Despite it being restricted to the two-dimensional case ($m = 2$), we provide the results for a core part of their algorithm on our running-example data in the supplement for reference (its performance is similar to EM). The authors of the second work [14] likewise only provide demonstrations on problems with low-dimensionality (up to $m = 5$), but more critically require execution "with a sequence of hundreds of different values" for unwieldy algorithm parameters.

Our results on synthetic data showed that SkinnyDip has the ability to outperform the comparison techniques by a large margin. Here we focused on data of the kind where

SkinnyDip shines, that is, data which 1) contain clusters each taking a unimodal form in each coordinate direction (otherwise are arbitrarily shaped), 2) contain clusters potentially "close" to others, and 3) contain a very high percentage (e.g. 80%) of noise. The large margin by which SkinnyDip outperforms the competition on this data is more or less maintained as the cluster count and level of noise are varied. Large differences *between* the comparison techniques are not evident. One main reason for their failure can be seen with an investigation into the $k = 3$ case: the clusters here are three long rectangles (similar to that in our running example), which the comparison methods tend to group together as one (giving an AMI of zero). In the noise plot the performance of all methods stays relatively constant as noise is varied. The reason for this is that the AMI in this case is only considering those objects truly belonging to a cluster (Figure 2, in comparison, provides results for the case in which noise points are also considered).

With increasing dimension m, however, the limitations of SkinnyDip's heuristic begin to surface. Practically though, we argue that "real" clusterings seldom require representation in such a high-dimensional space. The results of our real-world experiments showed, for example, that the use of SparseDip on high-dimensional data achieved dimensionality reduction to levels at which SkinnyDip's heuristics function well.

These SkinnyDip results on the real-world data were generally pleasing. Here we also compared to the recent FOSSCLU (*Finding the optimal subspace for clustering)* technique [6] which likewise tames high-dimensional data sets by searching for an appropriate clustering basis. Here it is necessary to draw a line between *generalized subspace clustering* techniques (which we do not address) and techniques like SkinnyDip and FOSSCLU. Subspace clustering techniques seek a separate subspace for each cluster in the general case. SkinnyDip and FOSSCLU, in contrast, argue that it is more valuable to find a *single* basis for clustering as *both* the intra- and inter-cluster relationships can be interpreted [6].

Like all clustering techniques, SkinnyDip has **limitations**. Firstly, it is clear that the dip is designed for univariate data. At a fundamental level there will thus exist cases of information-loss through our univariate projections. With some work we can construct pathological examples where SkinnyDip fails (consider $D \in [0,1]^{n\times 2}$ with one uniform cluster filling $[0, \frac{1}{2}]^2$ and another filling $[\frac{1}{2}, 1]^2$). The use of these projections, however, clearly also has its advantages: through them SkinnyDip is faster and is liberated from the need to perform multivariate distance computations.

It is also clear that SkinnyDip starts as a "global" technique but becomes increasingly "local" as the recursion progresses. That is, as the depth of recursion increases, the fraction of data points that are actually considered by the dip decreases. Practically this is seldom a problem, but again we can construct pathological examples which exploit this weakness (consider a univariate sample with successive modes that are each 50% the height of their predecessor, for example). Other limitations of SkinnyDip are that 1) a permutation of the features in the data may affect the outcome, 2) its modal hyper-intervals cannot precisely model "rounded" patterns, and 3) it is generally restricted to continuous data.

8. CONCLUSION

As a novel approach to clustering, SkinnyDip has a unique set of properties. It is deterministic, makes no strict model assumptions, and performs no distance calculations. Practically it is automatic (requiring only a statistical significance threshold) and *fast*. Its secret is the multi-level exploitation of the statistical *dip test* of unimodality on univariate projections of the data, enabling it to fundamentally distinguish itself from centroid-, distribution-, density- and connectivity-based approaches. This same test can be used to find an intuitive and interpretation-friendly basis for clustering through the concept of multimodal projection pursuit.

SkinnyDip outperforms competing automated techniques particularly well when clusters have heterogeneous unimodal shapes and the environment is extremely noisy. Future work shall explore possibilities for mitigating some of the undesirable effects of using univariate projections.

9. REFERENCES

[1] C. Böhm, C. Faloutsos, J.-Y. Pan, and C. Plant. Robust information-theoretic clustering. In *KDD*, pages 65–75, 2006.

[2] C. Böhm, C. Plant, J. Shao, and Q. Yang. Clustering by synchronization. In *KDD*, pages 583–592, 2010.

[3] H.-J. Bungartz and M. Griebel. Sparse grids. *Acta numerica*, 13:147–269, 2004.

[4] A. Dasgupta and A. E. Raftery. Detecting features in spatial point processes with clutter via model-based clustering. *Journal of the American Statistical Association*, 93(441):294–302, 1998.

[5] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, 1996.

[6] S. Goebl, X. He, C. Plant, and C. Bohm. Finding the optimal subspace for clustering. In *ICDM*, 2014.

[7] G. Hamerly and C. Elkan. Learning the k in k-means. *Advances in neural information processing systems*, 16:281, 2004.

[8] Y. F. G. Hamerly. Pg-means: learning the number of clusters in data. *Advances in neural information processing systems*, 19:393–400, 2007.

[9] J. A. Hartigan and P. Hartigan. The dip test of unimodality. *The Annals of Statistics*, 1985.

[10] A. Kalogeratos and A. Likas. Dip-means: an incremental clustering method for estimating the number of clusters. In *Advances in neural information processing systems*, pages 2393–2401, 2012.

[11] A. Krause and V. Liebscher. Multimodal projection pursuit using the dip statistic. *Preprint-Reihe Mathematik*, 13, 2005.

[12] D. Pelleg, A. W. Moore, et al. X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML*, pages 727–734, 2000.

[13] D. Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. Verlag Dr. Hut, München, Aug. 2010.

[14] W.-K. Wong and A. Moore. Efficient algorithms for non-parametric clustering with clutter. In *Proceedings of the 34th Interface Symposium*, 2002.

[15] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *Advances in neural information processing systems*, pages 1601–1608, 2004.