Using Polygonal Data Clusters to Investigate LIME

JESSE HE, The Ohio State University, USA
SUBHASISH MAZUMDAR, New Mexico Institute of Mining and Technology, USA

As machine learning models become more widely adopted, it is important to be able to create interpretable explanations for these models' predictions. LIME is a popular explanation framework which, despite successes in text and image data, still has some difficulties with tabular data. We present a method for generating synthetic tabular datasets drawn from classes that can be described geometrically using polygons, and use this method to try to investigate LIME's behavior on a classifier given classification tasks of various difficulties.

CCS Concepts: • **Software and its engineering** → **Software libraries and repositories**; • **Computing methodologies** → Supervised learning by classification.

Additional Key Words and Phrases: synthetic data, polygonal boundaries, machine learning explanations

ACM Reference Format:
Jesse He and Subhasish Mazumdar. 2021. Using Polygonal Data Clusters to Investigate LIME. 1, 1 (November 2021), 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With the increasing success of machine learning in analyzing data and the rising potential for its applications in diverse areas, the necessity for understandable explanations of machine learning predictions has become readily apparent. A key tool in evaluating such explanations, and in machine learning more generally, is the ability to generate synthetic data sets to use as easily visualizable examples of real machine learning problems. One class of problems with n-dimensional data involves identifying classes whose boundaries can be described geometrically using polygons, especially when the polygons are non-convex. Below, we will use the term "polygonal cluster" to refer to such a class of data points which is definable with a polygonal boundary, and the term "polygonal clustering" to refer to the task of classifying such data. We will also allow a polygonal cluster to have a certain fraction of outliers which lie outside the polygonal boundary. This presents an interesting class of classification problem that currently lacks a robust synthetic data tool.

There are a number of efforts in the realm of explainable AI, particularly regarding "opaque" or "black-box" models [2]. One popular approach is the use of Local Interpretable Model-agnostic Explanation (LIME) [8], which attempts to identify the contribution of individual features towards a classifier's prediction by observing the behavior of the classifier around perturbations of the original data point. An intuitive illustration from Ribeiro, Singh, and Guestrin [8] is given in Figure 1.

Although LIME has seen adoption in image and text analysis, LIME's application to tabular data has certain limitations. Different methods of perturbing or identifying a local neighborhood for a data point can create different explanations

Authors' addresses: Jesse He, The Ohio State University, Columbus, Ohio, USA; Subhasish Mazumdar, New Mexico Institute of Mining and Technology, Sorocco, New Mexico, USA.

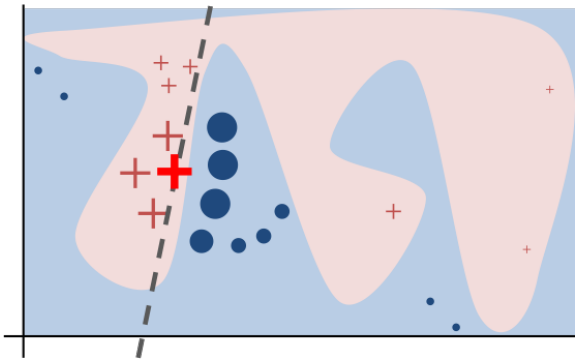

Fig. 1. Illustration to build intuition for LIME from Ribeiro, Singh, and Guestrin[8]. The lack-box model's prediction for the red cross is based on a complex decision boundary, illustrated by the blue/pink background. LIME samples nearby instances, gets the model's prediction, and attempts to create an explanation that is locally faithful.

for the same classifier on the same dataset [3], and Slack et al. [9] have described a method for "tricking" LIME with an adversarial biased classifier whose biases are undetected by LIME, which they argue is a weakness of similar post-hoc explainers. Because of the prevalence of tabular data in machine learning applications, it is important to investigate LIME's ability to explain tabular classification models.

One technique of interest is the use of a polygonal model, which attempts to create a suitable polygon to represent the shape of a data cluster, often employed with spatial data [1]. Of particular interest are clusters bound by polygons which are non-convex, since convex hulls can create large empty areas that do not tightly fit the structure of a cluster. Although sample datasets exist that can be used for such problems, there is no robust way to randomly generate and customize new datasets with polygonal clusters. In this paper, we describe a software tool which uses a simple but effective method of generating such classification problems, and we demonstrate its utility in investigating the behavior of LIME on a black box classifier.

2 GENERATING CLUSTERS WITH POLYGONAL BOUNDARIES

In this section we describe briefly the method for generating polygonal clustering problems. More detailed pseudocode can be found in Appendix B.

The first task in generating a polygonal clustering problem is to be able to generate polygons in the plane. We can specify an n-gon p as an ordered list $p = (v_1, v_2, \ldots, v_n)$ of its vertices. In order to generate these randomly we use a star-like approach: given a specified or randomly generated "center" we draw a radius r uniformly at random from some range $[r_m, r_M)$, where $0 \leq r_m < r_M$, and an angle θ uniformly at random from $[0, 2\pi)$, giving us the polar coordinates for each vertex. We then connect these vertices counter-clockwise to produce the polygon.

If we want to ensure that the polygon contains the central point about which it is generated, we ensure that no consecutive angles θ_1, θ_2 have a difference of more than π radians counter-clockwise. For $n > 3$ we it suffices to generate the angle θ of the k-th vertex inside the interval $[2(k-1)\pi/n, 2k\pi/n)$ for $n > 3$. For triangles, however, this may fail, so instead we generate the vertices (r_1, θ_1), (r_2, θ_2), (r_3, θ_3), with θ_1 selected uniformly at random from $[0, 2\pi)$ as before, then taking θ_2 from $[\theta_1, \theta_1 + \pi)$, and θ_2 from $[\theta_1 + \pi, \theta_2 + \pi)$.

Then given a polygon p, we wish to generate points that lie inside p. The simplest way to do this is by rejection sampling: simply generate trial points and discard any that lie outside of p. Using this approach, we can draw samples

not just uniformly distributed from the polygon but also from a multivariate normal distribution centered inside p, as well as allowing some number of points which lie outside of p. Of course, p need not be generated randomly by the above process; the user may specify a polygon by enumerating its vertices.

Combining these techniques, we can easily generate random point sets with polygonal cluster boundaries. Our software uses Matplotlib's `path` library [6] to represent polygonal paths and test if a polygon contains a given point. This also allows for easy visualization in plots made using Matplotlib and for integration with other Python libraries, including efficient vectorized operations with NumPy [5]. Examples are shown in Figures 2 and 3.

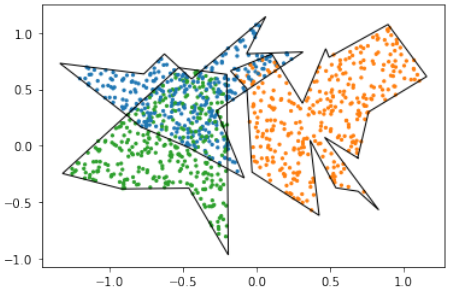

Fig. 2. Three uniform balanced polygonal clusters.

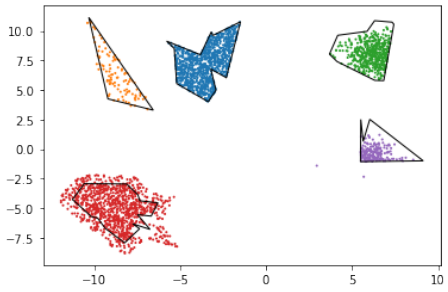

Fig. 3. Customized clusters.

2.1 Controlling Overlap

One additional task in generating new classification problems is to customize the difficulty of the problem. The primary way we achieve this is by moving polygonal clusters closer or farther away from each other, and in particular manipulating the *overlap* between clusters. Given a set $X \subseteq \mathbb{R}^2$ of points and regions P_1, P_2 bounded by polygons p_1 and p_2, respectively, we define the overlap between these clusters by

$$\text{overlap}(X, p_1, p_2) = \frac{|X \cap P_1 \cap P_2|}{|X|}.$$

For problems with more than two polygons, we extend this definition by counting the number of points of X that lie in the intersection of any two polygons:

$$\text{overlap}(X, p_1, \ldots, p_k) = \frac{\left|X \cap \left(\bigcup_{1 \le i < j \le k} (P_i \cap P_j)\right)\right|}{|X|}.$$

Then shifting clusters gives us control over the overlap of a classification problem, as seen in Figure 4. By manipulating the overlap of our polygonal point sets, we can effectively scale the difficulty of the classification problem: When there is no overlap, a simple linear model may suffice to accurately classify new test points. Introducing overlap between clusters can give us insight into how classifiers create their decision boundaries in the presence of ambiguous data.

2.2 Polytopes in Higher Dimensions

Real datasets frequently feature several features, of which some may be uninformative. In order to replicate this we can sample points from polytopes of dimension > 2, shown in Figure 5. Currently in the software these polytopes are specified by their orthogonal projections onto each standard coordinate plane, although this creates certain practical

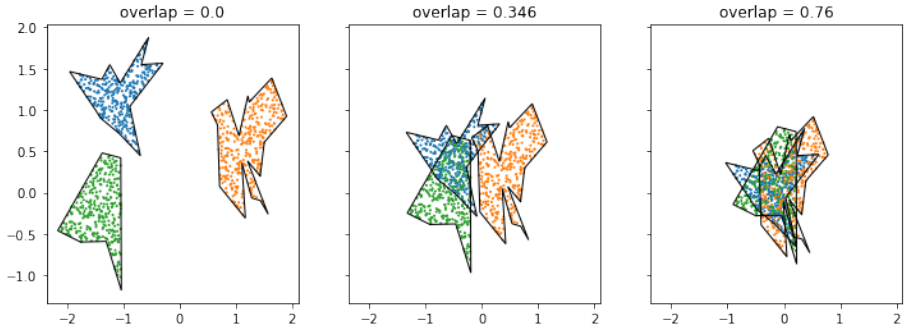

Fig. 4. Dataset from Fig. 2 with clusters shifted to create different overlaps. From left to right: dataset with no overlap, original dataset, dataset with high overlap

difficulties in implementation that are discussed further in Appendix A. We can also generate new redundant features by taking random linear combinations of existing features, and Gaussian noise can be added to increase the difficulty of identifying redundant features, illustrated in Figure 6.

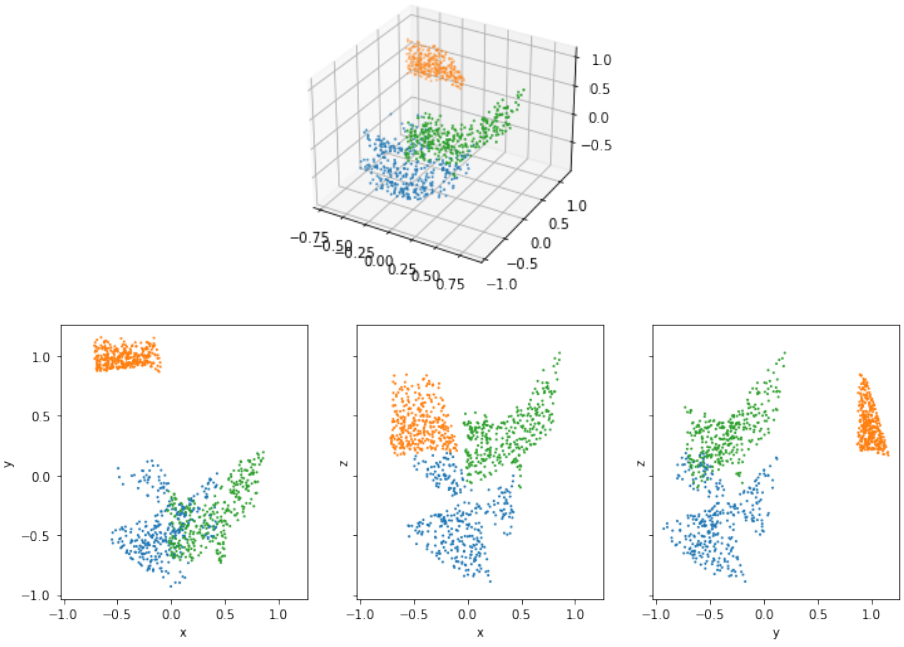

Fig. 5. A random clustering problem with polyhedral boundaries and its projections onto the xy, xz, and yz planes.

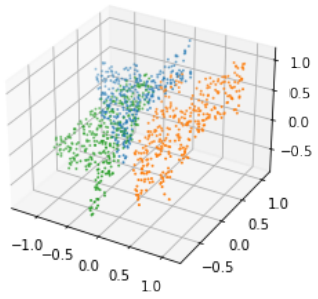

Fig. 6. The dataset from Fig. 2 with an artificial redundant feature.

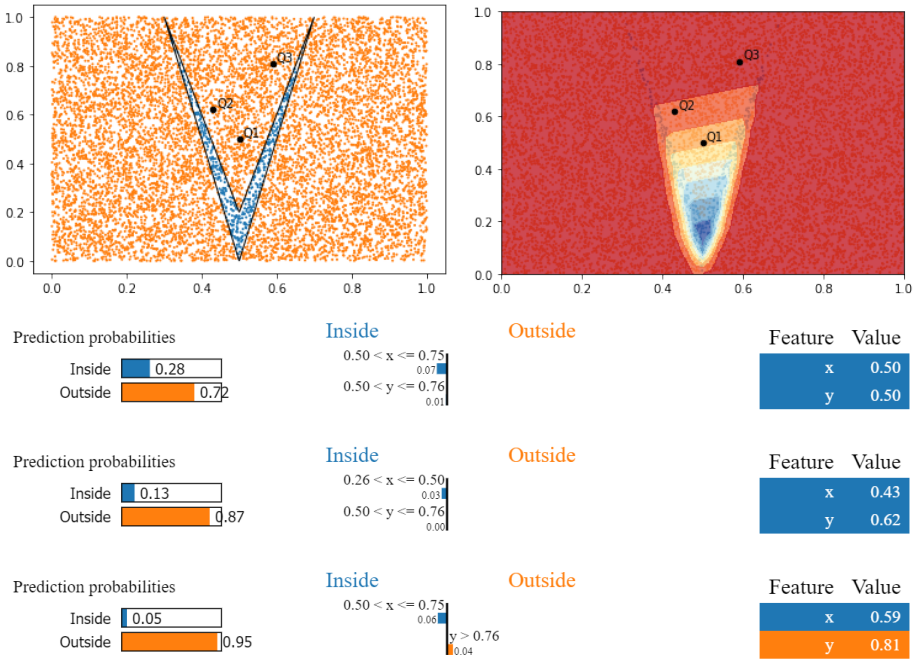

Fig. 7. A classification problem featuring a "V"-like polygon with sample points $Q_1 = (0.5, 0.5), Q_2 = (.43, .62), Q_3 = (.59, .81)$ (top left), the classifier's decision function (top right), and LIME's explanations at Q_1, Q_2 , and Q_3 (bottom).

3 EXPLAINING CLASSIFIER BEHAVIOR ON POLYGONAL CLUSTERS

We first demonstrate the use of polygonal boundaries to specify challenging classification problems by creating a polygon $p = ((.5, 0), (.7, 1), (.5, .2), (.3, 1), (.5, 0))$ and training a multilayer perceptron (MLP) classifier to classify points as inside or outside it. We then select predictions in challenging areas and ask LIME to explain these predictions.

As we can see in Figure 7, the inner triangle which is in the convex hull of p but not within its boundaries is challenging for LIME to explain. Because LIME looks at individual features, its attempted explanations for the specified

points contradicts the classifier's prediction: the classifier correctly identifies that each point lies outside p, but LIME's explanations suggest that each point *should* be classified as inside the polygon based on their x values.

Now, using our method of generating clusters we can demonstrate LIME's ability to identify the relevant feature in an easy clustering problem. We create two polyhedral clusters whose xy projections are nearly indistinguishable (an overlap of 0.633) but which are separated by a plane through $z = 0$. As Figure 8 shows, LIME can successfully identify that the classifier only needs to use the z coordinate to make its prediction.

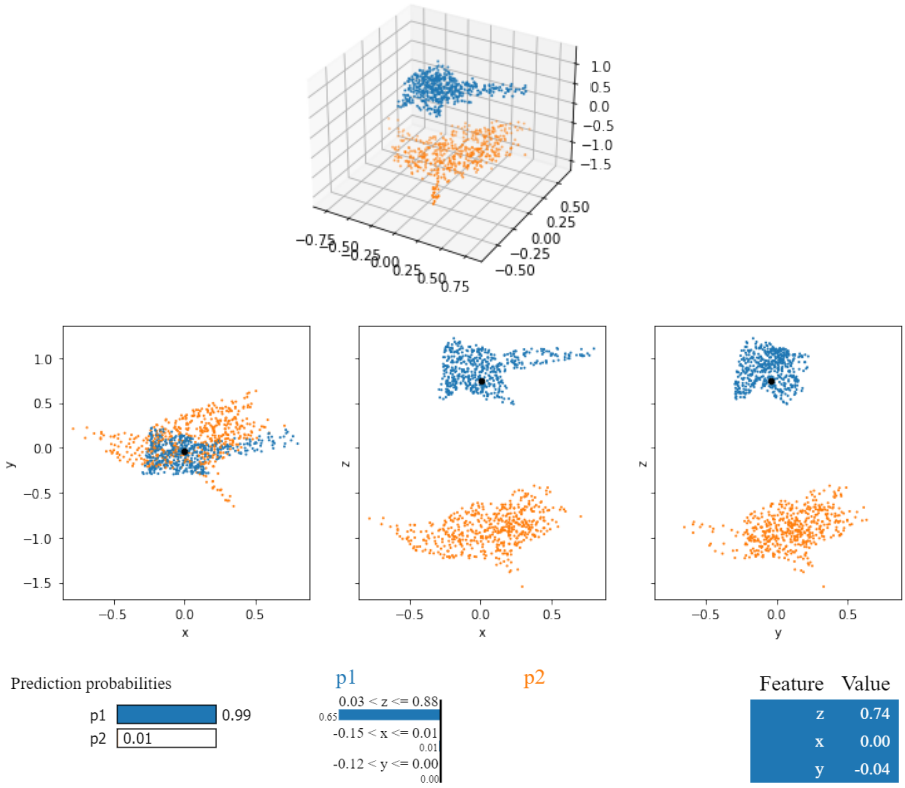

Fig. 8. An easy 3d classification problem featuring two clusters that are linearly separable with a plot (top) and its xy, xz, and yz projections (middle), along with LIME's explanation (bottom).

To investigate the efficacy of LIME in evaluating tabular data in 2 dimensions with the overlap we have defined, we create a binary classification problem with two polygonal clusters $p1$ and $p2$ whose overlap is between 0.1 and 0.2, and we train an MLP on a subset of the data. We then probe the classifier's prediction of a particular point using LIME and compare it to the classifier's actual decision function in Figure 9.

All three points were chosen to be difficult for the classifier. Each lies within the boundaries of both $p1$ and $p2$ but is very close to the boundary of $p2$, and this difficulty is reflected when examining the classifier's decision function. LIME's explanations at Q_1 and Q_2 reflect the classifier's uncertainty at those points, but its behavior at Q_3 is unintuitive: the MLP classifies Q_3 in $p2$ with a probability of 0.73, but LIME's explanation for this prediction asserts that the x and y

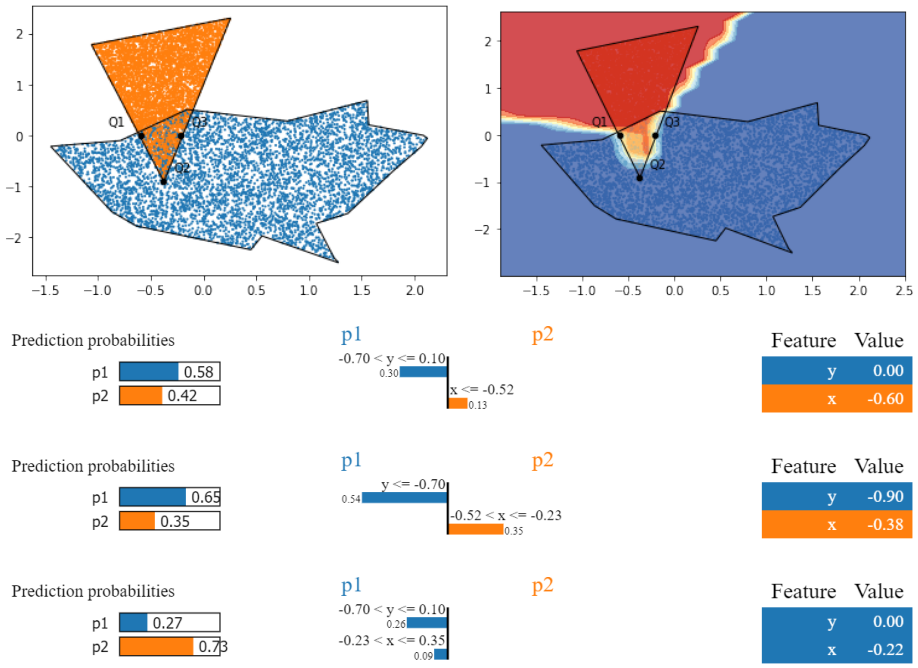

Fig. 9. A binary polygonal classification problem. A plot of the dataset and polygons (top left) next to the classifier's decision boundary (top right). In both plots the points $Q_1 = (-0.6, 0)$, $Q_2 = (-0.38, -0.9)$, and $Q_3 = (-0.22, 0)$ are each marked by a red "X." Below are LIME's explanations for the classifier's predictions at these points.

values are both in ranges correlated with $p1$. This explanation contradicts the actual prediction of the classifier at Q_3, appearing to violate LIME's local faithfulness.

4 CONCLUSION AND FUTURE WORK

This method of generating and manipulating polygonal tabular data is effective at generating simple example datasets for clustering problems. The generated datasets are easily visualized and can be customized to fit different geometric and statistical structures. Using polygons to specify the geometric structure allows for customization of cluster shape, and using overlap allows for customization of the difficulty of a generated classification problem.

In particular, our simple definition and manipulation of overlap for polygonal clustering problems allows us to demonstrate surprising behavior with LIME. Although overlap is currently only defined for polygonal clusters in two dimensions, the definition generalizes easily to other polytopes and a more refined computational approach may allow us to further investigate the behavior of LIME.

Future work could continue to develop this method of generating and manipulating tabular data with more features, adding more robust support for higher-dimensional polytopes. In addition, these methods could be used to investigate the behavior of other post-hoc black-box explainers like SHAP [2], as well as other classification methods including polygonal modelling [1] or dimensionality reduction.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation under grant no. CNS-1757945.

REFERENCES

[1] Fatih Akdag, Christoph F. Eick, and Guoning Chen. 2014. Creating Polygon Models for Spatial Clusters. In *Foundations of Intelligent Systems*, Troels Andreasen, Henning Christiansen, Juan-Carlos Cubero, and Zbigniew W. Raś (Eds.). Springer International Publishing, Cham, 493–499.

[2] Vaishak Belle and Ioannis Papantonis. 2021. Principles and Practice of Explainable Machine Learning. *Frontiers in Big Data* 4 (2021), 39. https://doi.org/10.3389/fdata.2021.688969

[3] Przemyslaw Biecek and Tomasz Burzykowski. 2021. *Explanatory Model Analysis.* Chapman and Hall/CRC, New York. https://pbiecek.github.io/ema/

[4] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. 2013. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 108–122.

[5] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature* 585, 7825 (Sept. 2020), 357–362. https://doi.org/10.1038/s41586-020-2649-2

[6] J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. https://doi.org/10.1109/MCSE.2007.55

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

[8] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California, USA) *(KDD '16)*. Association for Computing Machinery, New York, NY, USA, 1135–1144. https://doi.org/10.1145/2939672.2939778

[9] Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. 2020. Fooling LIME and SHAP: Adversarial Attacks on Post Hoc Explanation Methods. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society* (New York, NY, USA) *(AIES '20)*. Association for Computing Machinery, New York, NY, USA, 180–186. https://doi.org/10.1145/3375627.3375830

A LIMITATIONS IN HIGHER DIMENSIONS

The current approach to specifying a polytope in dimensions > 2 is based on its orthogonal projections: for example, we specify a polyhedron by taking a polygon p in the xy-plane and considering this in the xz- and yz-planes, then taking the intersection of the cylinders created by extending these projections. In other words, a point (x, y, z) is in the polytope if p encloses (x, y), (x, z), and (y, z) in two dimensions. This requires the polygon be roughly centered on the line $y = x$, since otherwise there may be no points which satisfy this condition for a given polygon, and even then generating points may be impossible. This also means that extending the overlap features to higher dimensions requires either checking each $\binom{n}{2}$ projections of an n-dimensional polytope, or would require a retooling of how polytopes are specified.

B METHODS AND ALGORITHMS

The data generation methods described in this paper have been implemented at github.com/he-jesse/polydata. Interface design was based in part on the datasets module of the sci-kit learn project [4] and the implementation uses a number of well-known Python libraries [5–7]. This appendix contains pseudocode for simple versions of the two-dimensional algorithms. Note that the current implementation supports additional features not described here.

Algorithm 1: RandomPoly

Data: A number n of vertices, a center (x_0, y_0), a range (r_m, r_M) of distances from the center to a vertex
Result: A polygon $p = (v_1, v_2, \ldots, v_n)$
begin
  $p \longleftarrow ()$
  for $i \in \{1, \ldots, n\}$ **do**
    $(r_i, \theta_i) \longleftarrow (\text{random}(r_m, r_M), \text{random}(0, 2\pi))$
  end
  $V \longleftarrow \text{sort}(r_i, \theta_i)$ by θ
  for $(r_i, \theta_i) \in V$ **do**
    p.append$((r_i \cos(\theta_i) + x_0, r_i \sin(\theta_i) + y_0))$
  end
  return p
end

Algorithm 2: SamplePoly

Data: A number n of samples, a polygon p
Result: A set X of points with $|X| = n$ and $p.contains(X)$
begin
  $X \longleftarrow \emptyset$
  while $|X| < n$ **do**
    $x \longleftarrow$ random point in the bounding box of p
    if $p.contains(x)$ **then**
      $X \longleftarrow X \cup \{x\}$
    end
  end
  return X
end

Algorithm 3: MakePoly

Data: numSamples n, numPolygons m of polygons, range (r_m, r_M) to pass to Alg. 1, a bounding box B
Result: A set X of points, a set P of polygons, and a map $f : X \to P$ so that $f(x).contains(x)$ for $x \in X$
begin
  $X, P \longleftarrow \emptyset$
  for $i \in \{1, \ldots, m\}$ **do**
    $p \longleftarrow \text{MakePoly}(\text{random}(B))$
    $X_p \longleftarrow \text{SamplePoly}(p)$
    $X \longleftarrow X \cup X_p$
    $P \longleftarrow P \cup \{p\}$
    for $x \in X_p$ **do**
      $f \longleftarrow f \cup \{(x, p)\}$
    end
  end
  return X, P, f
end

Algorithm 4: ComputeOverlap

Data: A point set X, a polygon set P
Result: The proportion c of overlap
begin
  $Y \longleftarrow \emptyset$
  for $\{p_1, p_2\} \in P^{(2)}$ **do**
    for $x \in X$ **do**
      if p_1.*contains*(x) *and* p_2.*contains*(x) **then**
        $Y \longleftarrow Y \cup \{x\}$
      end
    end
  end
  return $|Y|/|X|$
end

Algorithm 5: MakeOverlap

Data: A point set X, a polygon set P, a function $f : X \to P$, a range (c_m, c_M) of overlap values
Result: A point set X' and polygon set P' such that $c_m <$ overlap $< c_M$
begin
  $X' \longleftarrow X$ $P' \longleftarrow P$ $s_m \longleftarrow -1$
  $s_M \longleftarrow 1$
  $s \longleftarrow (s_m + s_M)/2$
  while $c_m >$*overlap*(X', P') *or overlap*$(X', P')> c_M$ **do**
    if *overlap*$(X', P') < c_m$ **then**
      $s_M \longleftarrow s$
    end
    else if *overlap*$(X', P') > c_M$ **then**
      if $s_M - s_m < \varepsilon$ `/* ε is some small threshold */`
      then
        $s_M \longleftarrow s_M + 1$
      end
      $s_m = s$
    end
    $s \longleftarrow (s_m + s_M)/2$
    for $x \in X'$ **do**
      $x \longleftarrow x + s \cdot f(x)$.centroid())
    end
    for $p \in P'$ **do**
      $p \longleftarrow p + s \cdot p$.centroid()
    end
  end
  return *X', P'*
end