Journal of Artificial Intelligence Research 1 (2021) 1-15 Submitted 10/21; published 12/21

P-Sphere Hull: an Explicit Domain Model for Trusted AI

Benjamin Mathiesen BENJAMIN.MATHIESEN2@CAPGEMINI.COM
Hybrid Intelligence, Capgemini Engineering

Abstract

Machine learning has an extrapolation problem: standard pipelines for training, validating, and deploying a model do not recognize when a new input lies outside the domain of the training dataset. In principle, a new prediction can be trusted if the inputs lie inside a well sampled region of the feature space, but high-dimensional datasets can have complex topologies and low contrast between densely and sparsely sampled regions. To compensate for this intrinsic uncertainty, industry players are proposing common-sense standards for trusted AI, such as periodic performance evaluations and training users on the intended scope of the model. In contrast, this paper proposes a framework to explicitly model the boundary of a training domain in high-dimensional feature space, and test each new input for membership in the domain before making a prediction. In this way, we can build AI applications that are trustworthy by design, because they respond appropriately when presented with an input that does not look like its training data. The domain model, called a *P-Sphere Hull*, is a data description and validation framework that can be learned independently of the predictive model and used with any AI technique. It is implemented as a Python package, available on the author's public github.

1. Introduction

This paper attempts to answer a very basic question: "How can we know when a machine learning model is probably wrong?" It is a question with very real impact on the performance and value of projects in the data science industry, as well as on the acceptance of AI applications by society as responsible and transparent decision support systems (Cheng, Varshney, & Liu, 2021). We are starting to see the fallout from industrial applications that failed to answer this question, ranging from lawsuits against biased algorithms to adversarial attacks on safety-critical systems. In order for machine learning (ML) and artificial intelligence (AI) to gain widespread acceptance, data scientists in academics and industry must be ready to address the issue of trust.

A basic fact of numerical analysis is that interpolation error is bounded, while extrapolation error grows without limit. This principle applies to all models based on a sample of observations, from simple parametric fitting functions to deep learning networks. In research, we describe the sampling domain alongside any fitted model, so that people reusing the model know its range of validity. In the domains of ML and AI, however, domain modeling and the possibility of large extrapolation errors have not generated much discussion.

There are many reasons for this silence. First is the difficulty of visualizing high-dimensional datasets. Gaining an intuition for the topology of the training domain requires significant human effort, in the form of trying out various dimensionality reduction techniques, plotting different projections, and exploring subsets of the data. Second, it is convenient to assume that the training data are representative of future inputs. We may feel this assumption is justified if we have access to many years of historical data, or if the features represent a physical system whose constraints we understand very well.[1] Third, if a model makes accurate predictions 99% of the time, there may be no business value in understanding the 1% of errors.

I do not want to suggest that the academic and industrial communities are indifferent to questions of prediction quality and extrapolation error. Discussion on this issue is growing steadily, with many interesting results (see Section 5, Literature Review). However, when we speak of explainable, interpretable, and trusted AI, the conversation often focuses on understanding why the model is making a specific prediction, and approximating its local behavior with human-interpretable rules. Rarely do we ask ourselves whether the training examples are adequate for a prediction, and if we do, the answer is often to go get more training data. Major industry players such as Google (Google AI, 2021) and Microsoft (Roach, 2020) have published common-sense standards for deploying trustworthy AI applications, for example:

- Periodically test model performance and retrain as often as necessary.
- Enrich the data with counterexamples, for example to combat gender and racial bias.
- Train consumers of the model on its intended scope and limitations.
- Report performance statistics in intuitive ways.

These are excellent principles, but they imply that a client's level of trust should be based on their relationship with the AI providers, not on the model itself.

This work proposes a different approach: to deliver models that are aware of their own training domain, and can either refuse to make a prediction or alert the user to low-quality predictions when the received inputs are out of scope. I describe a simple, practical framework for modeling the domain of a high-dimensional training dataset. This model can be used to develop a filter that is independent of the ML or AI model making the prediction, giving it the ability to test new inputs for membership in the domain. The framework, called "P-Sphere Hull", is implemented as a python package available on github (https://github.com/bmathiesen/p-sphere-hull). The basic idea is as follows: first, use a clustering method to divide the training data into sub-domains, then model the domain as the union of envelopes containing each cluster. The resulting domain boundary (hull) is a collection of intersecting simple geometric forms, in practice hyper-cylinders (shapes that are spherical in one subset of dimensions, and cuboid in the others). The membership test for new inputs is a sequence of individual tests against the sub-domains with low computational cost. The hull can also be used to gain an intuition for the global topology of the dataset, because the subdomains have different local dimensionalities and sets of low-variance features.

Section 2 of this paper describes the justification for the domain model in more detail, and explores the various trade-offs that I considered in designing it. Section 3 explains the algorithm, the proposed workflow for building and deploying a domain-aware ML model, and how the different modules of the package support the process. Section 4 describes experiments using the P-Sphere Hull to detect extrapolation error in four public datasets.

1. Given that anomaly detection is an important application of ML in many industry sectors, however, clearly understanding the normal behavior of a system is not enough.

Section 5 provides a literature review of papers with a strong connection to the extrapolation problem. Section 6 concludes and suggests future development of the P-Sphere Hull approach.

2. Motivation

This section provides additional context for this research. First I discuss some of the common topological features of high-dimensional datasets that I have observed in various industrial sectors. Next, I discuss the urgent need for trusted AI in practical applications and how an explicit domain model can promote trust. Finally, I list the specific trade-offs and choices that I made in designing the P-Sphere Hull method.

2.1 Real datasets have complex topologies

A key motivation of this work has been to better understand industrial datasets, which often have moderately high dimensionality (tens to hundreds of features) and non-trivial geometry. These datasets can be imagined as point clouds in feature space. If we zoom in on any subset of the point cloud, we usually observe that it occupies a lower-dimensional manifold (for example, only 5 of 10 features display signficant variation). However, the scatter in the point cloud tends to blur any intrinsic geometry that might exist. By plotting different 3D projections of such data, we can gradually gain insight into its global topology. At this point in the data science workflow, we usually start trying to select or engineer a subset of independent, information-rich features to represent the dataset.

However, even after de-correlating, rescaling, and selecting the features, the following topological properties are typical:

1. Most data form a single irregular cloud, but there are some disconnected “islands”.
2. Some low-dimensional subdomains extend outward from the central mass.
3. There are several strong concentrations of data, but it is not obvious whether they are connected.
4. A feature has low variance within some clusters, but high variance within others.
5. The variance and the geometry of a substructure are similar in scale.

Figure 1 shows some examples of datasets with non-trivial topologies from the OpenML library (Vanschoren, van Rijn, Bischl, & Torgo, 2013).

In this context, we see that there are some vast regions of feature space where the model has no training data at all (extrapolation) and some regions where the data are poorly sampled but still span an important subvolume (interpolation). Outside the point cloud, we can expect meaningful but less accurate predictions if a new input is not too far from the sampled region. Otherwise, we can expect meaningful and accurate predictions if the function being modeled is “well-behaved” or varies slowly on scales comparable to the holes in the data.

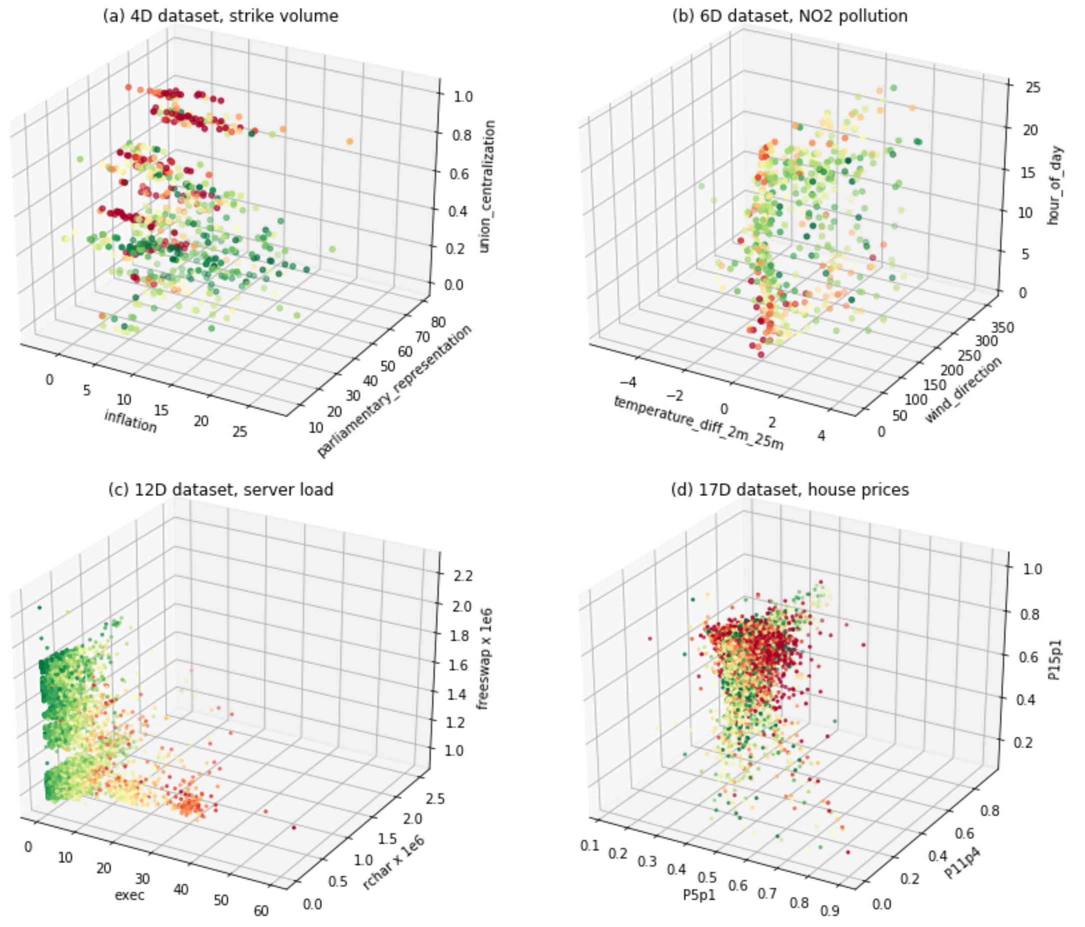

Figure 1: Four high-dimensional regression datasets selected from the OpenML repository (Vanschoren et al., 2013) with non-trivial topologies. Only a single 3D projection is shown for each dataset. The color scale represents the target variable.

More insidious are the gaps adjacent to multiple densely sampled regions, for example those between the "spokes" radiating from the center in Figure 1d or the gaps between clusters in Figure 1a.[2] Any AI model will interpolate into the empty region without complaint, and from a mathematical standpoint we can even estimate an error bound on its prediction. However, the different data clusters can have different business meanings. For example, in image classification with deep learning, a novel input that maps to a gap between clusters in the network's latent space may not represent any category of image that the network was trained to learn; nonetheless, a prediction will be made.

2. The "hub and spoke" topology of Fig. 1d is also typical of document corpora transformed into dense vectors with latent semantic analysis. In this context the spokes represent a set of documents with similar vocabulary distributions but different lengths.

Note that while speaking of interpolation and extrapolation errors, our intuition leans towards regression problems where we try to predict a continuous target variable. In this paper I focus on regression datasets because the meaning and magnitude of error is easier to interpret. The method I propose is also valid for classification problems, in particular for models such as SVM and logistic regression that provide interpretable probability estimates which we convert to a class prediction by thresholding. In this class of problems, extrapolation error generates uncertainty in the position of the decision boundary, so the impact on performance metrics depends on the quantity of data positioned close to the boundary. The problem is tractable but I prefer to relegate it to future research.

2.2 There is a need for trusted AI

Ideally, even when working in high dimensions, we would like to define an envelope or hull around the training data that defines the domain of the model's knowledge. The previous examples show some of the reasons why this is difficult. This subsection explores some of the consequences on trusted AI if we ignore the problem.

Commonly, data scientists perform cross-validation using one portion of the available data for training, and test performance on a validation set held in reserve. Good performance on the validation set is taken as a sign that the model has correctly generalized–at least for interpolation. We *hope* that the performance will also be strong on new inputs, but we also know that it is not guaranteed. Therefore, we encourage clients to implement continuous monitoring. In practice, the field performance of a model is often inferior to its training performance, and models can degrade over time. There are two reasons this can happen. First, the relationships in real-world data can be different from the training data. For example, a recommendation system can change user behavior after it goes live. Second, the new inputs can be drawn from a different region of feature space. In both cases, the solution is to retrain the model, but this happens only after the client or end user has experienced a disappointing (and hopefully not catastrophic) result.

Some applications attempt to describe the domain of applicability as a hypercube[3] delimited by the individual feature ranges, and raise a warning if a new input falls well outside this region of feature space. This does not add much reliability or trust to the model in practice, because it is rare for a new input to be so far from the training data in feature space. Further, as illustrated in Figure 1, the hypercube can contain very large regions that are empty of training data.

To raise the level of trust in AI applications, neither continuous monitoring nor a hypercube domain boundary suffices. We want to be able to catch potential errors in the model prediction *before* they reach end users, or we will degrade their trust when the model is wrong. This means identifying and modeling the gaps in the training data, and raising a warning to the end user if a new input falls inside one of these gaps.

Ignoring the problem can have severe consequences: adversarial examples can be framed as an exploitation of the training domain's unknown topology. In high dimensions, the training domain is difficult to formalize as a geometrical boundary, yet data in the gaps between densely sampled regions can be found with disturbing ease. For example, "white-box at-

3. This term has gained some traction in industrial data science, although it signifies a high-dimensional cuboid with unequal side lengths.

tacks" with access to the model architecture and weights can estimate the sensitivity of the prediction to perturbations, and discover new vectors that trigger a desired adversarial output while minimizing the distance from known inputs (Chakraborty et al., 2021). In other words, traveling a short distance in a direction normal to the training domain boundary is an efficient way to construct adversarial examples. Another way to characterize the relationship between domain topology and adversarial examples is through the detection of "non-robust features" that an AI model relies on for classification. A simple example of a non-robust feature would be a curve that transverses the training domain while crossing gaps in the point cloud. A series of decision surfaces perpendicular to the curve can be very useful for prediction, but the constructed feature lacks crucial information on the behavior of the target and is vulnerable to exploitation (Ilyas et al., 2019). A domain-aware model, on the other hand, would be able to provide high-confidence predictions for inputs in well-sampled regions, while refusing to make predictions in the adversarial domain as a matter of principle. Hence, domain modeling is an essential component of safety-critical AI applications.

Solving the problem will bring several other benefits. A domain-aware model would not only have a much lower error rate, it would do a better job of communicating uncertainty to the end user. Along with the prediction, it could provide qualifying information such as "this input comes from a densely/sparsely sampled region of feature space" or "this input falls outside the training domain, but not by much". (These metrics can be quantitative for the data scientists working on the application, but qualitative for end users.) It would also give more precise estimates of model performance during cross-validation, by allowing data scientists to distinguish between in-domain and out-of-domain performance metrics. Finally, it would permit generating simulated data by resampling in-domain regions of feature space, even when the dataset is too small to train a generative neural network.

2.3 Design considerations and trade-offs

The P-Sphere Hull approach to defining the training domain meets several criteria for integration into industrial applications:

- **Modular.** It provides a geometric model of the training domain as a collection of subdomains bounded by simple geometric envelopes. The subdomains can be defined by any clustering method upstream and the domain model can be consumed by any AI model downstream.

- **Computationally inexpensive.** The domain model should add almost no overhead to training and prediction.

- **Intuitive.** The domain model can provide information on the topology of the dataset in high-dimensional space. For example, it can reveal a series of overlapping subdomains with low intrinsic dimensionality and similar orientations extending away from the main body of the point cloud.

- **Compact.** The domain model hypervolume is small compared to the hypercuboid defined by individual feature ranges.

- **Insensitive to scatter.** I assume that industrial data are subject to both measurement errors and random sampling uncertainty. Hence, I view the training domain as a collection of point clouds rather than a continuous manifold.

- **Permissive.** Since the data are assumed to be noisy, the boundaries of the domain should not contain too many data points.

The above choices lead to some important trade-offs.

Topology vs. noise: I did not try to integrate manifold learning methods such as UMAP (McInnes, Healy, & Melville, 2020) and locally linear embedding (Roweis & Saul, 2000) into the domain model, because I consider noisy and discontinuous data to be more common in industry settings than data that occupy an continuous and differentiable manifold. However, such methods are still very useful to apply upstream of the domain model, when it is time to define the initial clustering solution (see Section 3).

Computationally inexpensive: p-spheres are a natural choice for bounding a cluster of data, since testing whether a new vector lies inside the p-sphere requires only a translation and a scalar product. In addition, the collection of p-spheres only needs to be generated once before deploying a model. I considered other shapes for bounding a cluster, such as a simplex or a bounding box rotated to the principal components of the subdomain. Implementing membership tests for these shapes is orders of magnitude more expensive, and the cost is hard to justify. For example, while using rotated bounding boxes would reduce the volume of the bounding envelope, it would also place more data on the hull boundary (see next paragraph) and in initial tests the directions of the principal components were subject to too much random error to represent well the local topology.

Compact vs. permissive: The bounding p-sphere of a cluster is typically larger than the bounding box by one or two orders of magnitude.[4] Therefore, the compactness criterion implies that a domain model should be composed of p-boxes. However, p-spheres have two advantages. First, with p-spheres we can estimate a radial density distribution of the cluster data, and use this distribution for prediction quality metrics and resampling. Second, p-spheres are a more permissive choice. Their outer shells may have low data density, but a new observation in the outer shell of a p-sphere is close enough to high-density regions that it can still be considered part of the domain. The p-box geometry does not lend itself to profiling an interior density distribution. As developed, the model compromises between compactness and permissiveness by using p-cylinders, which enclose the features with small variance using a hypercuboid geometry.

The distinction between p-spheres and p-cylinders, as well as the algorithm for deriving a p-cylinder envelope from a dataset, are described in detail in the next section. Hereafter, the term "p-sphere" is used generically to indicate a single subdomain envelope, whether its actual geometry is fully or just partially spherical.

4. This observation can be explained as follows: for a p-spherical envelope, the single point farthest from the cluster center is responsible for an outer shell that contains a large volume. Even in 3 dimensions, the outer spherical shell with thickness 0.1 accounts for about 25% of the unit sphere's volume. In p dimensions, the outer shell has an enormous impact on the p-sphere hypervolume. For p-boxes, on the other hand, the same point will usually increase the size of the box in only one or two faces, so the hypervolume increase due to a single outlier is much less significant.

3. Methods

This section describes how a data scientist can build domain-aware AI models, and the key algorithms and class methods of the new python package `PSphereHull`. All machine learning methods used by the package (specfically, K-Means, PCA, and Sparse PCA) are implemented using scikit-learn version 0.23.2 (Pedregosa et al., 2011).

3.1 Data science with a domain model

The following list describes an ideal data science workflow for delivering a predictive model with high-dimensional data. Steps 1-3 are common practice, if time and budget allow. Steps 5 and 6 describe the process of designing and delivering a model. Step 4, building an explicit domain model, is new.

1. **Data definition and feature engineering.** In collaboration with domain experts, learn which variables are most meaningful and create a preprocessed dataset X consisting of transformed, scaled, non-redundant features.

2. **Data exploration.** If X has too many features to easily visualize, look for substructures using dimensionality reduction techniques such as random projection, PCA, t-SNE, and UMAP.

3. **Clustering.** If step 2 reveals significant substructure, try to find an appropriate clustering method. The choice depends on the topology of the dataset: whether the clusters are overlapping or disjoint, whether they have similar or diverse sizes, whether the clusters are more or less convex, and so on. The goal is to enrich the dataset with cluster labels to support model segmentation.

4. **Domain model.** The new step, supported by `PSphereHull`, is to build a geometric model of the training domain, so that we can compare in-domain to out-of-domain (OOD) performance and package our machine learning algorithm with explicit awareness of the domain boundary.

5. **Modeling.** Test and optimize different machine learning algorithms on the training data, select the best one according to the client's value criteria (accuracy, recall or precision, trust, explainability, etc.) Cross-validation using the domain model can now distinguish in-domain metrics (interpolation) from OOD metrics (extrapolation).

6. **Industrialization.** Package the final model with its P-Sphere Hull model and deploy them to a server.

The next sections describe how step 4 can be implemented in more detail.

3.2 Building the domain model

The starting point to building an explicit domain model is to run a clustering algorithm on the dataset. The goal is not to find the most "natural" clusters in the data, but to subdivide the point cloud into a set of manageable subdomains. Ideally, the subdomains should be relatively small (tens to hundreds of samples) and roughly convex. The results of step 3 above might suggest a specific clustering method suited to the topology of the data. However, using K-means clustering or a Gaussian mixture model to find 30-50 subdomains provides a good starting point in most situations.

Running a clustering method on the training data yields a set of labels and cluster centers. The next step is to visualize the clusters and ensure that they all have reasonable sizes and densities. Often, at least one cluster will collect a set of widely distributed points far from the main concentration of data. The p-sphere surrounding this cluster will add a lot of unused space to the domain model. To mitigate this problem, one can subdivide the cluster or remove outlier data points (if they belong to a sparse subdomain, the client may agree that outliers are irrelevant to model training). By examining the clusters one by one, we can identify sparse clusters and adjust the clustering solution until the whole collection represents the data topology well.

The `PSphereHull` package contains a module called `refine_cluster_set` to assist in this process. It contains functions to calculate the size, intrinsic dimensionality, and density of each cluster (returning a dataframe of attributes indexed by cluster label), visualize the cluster set, split a cluster into two subclusters, fuse two clusters, and remove a single point from a cluster. Each function modifying the cluster set returns new versions of the labels and cluster centers, so that the workflow can be iterative and interactive. A jupyter notebook included with the package demonstrates this part of the workflow.

When the cluster set is a good representation of the training domain, without too much wasted hypervolume in the p-spheres that enclose individual clusters, the next step is to form the hull. The `PSphereHull` module defines two classes: `PSphere` represents the envelope around a single cluster, and `PSphereHull` represents a collection of `PSphere` objects. The classes include methods to detect whether a new point is inside the a single p-sphere or the whole hull, calculate the hypervolume of the hull, include or exclude p-sphere from the domain model, and detect redundant p-spheres.

By creating a `PSphereHull` object and packaging it with the model, any AI application can test new data for membership in the training data domain before making a prediction, then decide what to do with the new data based on the client's needs.

3.3 p-cylinder algorithm

By default, the `PSphere` class generates a p-cylinder, which is shaped like a p-sphere in one subset of dimensions with high variance, and shaped like a hypercuboid in the dimensions with low variance. The volume of a p-cylinder is the volume of the p-sphere times the volume of the hypercuboid. This choice reduces the hypervolume enclosed by the shape while conserving the desirable properties of p-spheres (permissivity, radial density profile) for the dimensions with high variance. Figure 2 shows an example of a point cloud shaped like a p-cylinder, with 3 high-variance dimensions and 3 low-variance ("compact") dimensions. Note that the p-cylinders used here are always aligned with the feature axes, in order to preserve the low computational cost of the method. Mathematically, it is also possible to characterize a dataset with complex topology as a collection of generalized hypercylinders (Ward & Guo, 2010).

Given a single cluster's dataset X with dimensions $n \times p$, the *local dimensionality* of the data is defined by principal component analysis (PCA) as the number of components that

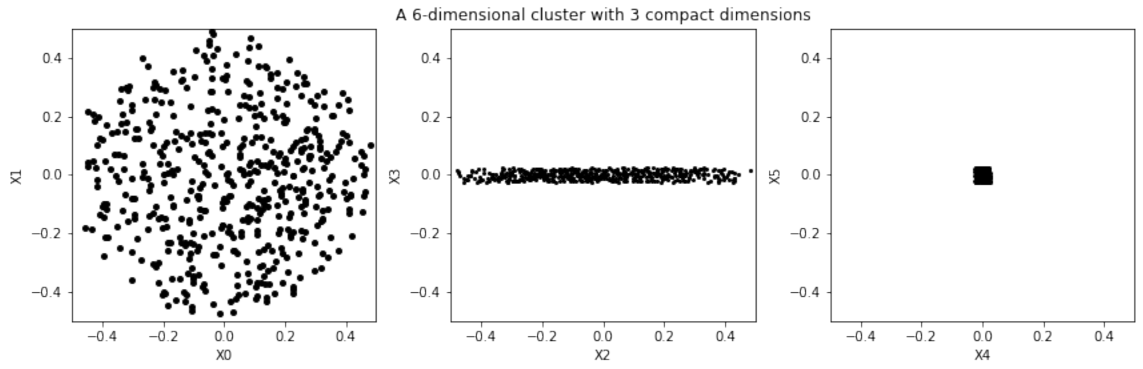

Figure 2: A six-dimensional dataset with p-cylinder geometry. In the first 3 dimensions (X0, X1, and X2) the boundary is a 3-sphere with radius 0.5. In the others (X3, X4, and X5), the boundary is a 3-cube with side length 0.05. The data were generated by uniform sampling within the 6-cube with side length 1, discarding samples outside the boundary. Note that the 6-cylinder volume is only 6.5e-5, so a machine learning model trained on this data cluster needs to extrapolate to reach most of the volume inside the 6-cube.

explain 90% of the total variance. This variance threshold is the main tunable parameter of the algorithm that defines the p-cylinders. After running PCA to determine the local dimensionality of X, the `PSphere` class runs Sparse PCA (Zou, Hastie, and Tibshirani, 2004; Jenatton, Obozinski, and Bach, 2010) to find the same number of components; i.e., if the local dimensionality of X is 2, then it calculates the first 2 sparse components. Any features not used by the Sparse PCA components are considered "compact".

The geometry of the p-cylinder envelope containing X is defined by the following properties. The p-cylinder envelope of a local dataset is initialized using Algorithm 1.

- p, dimensionality of the dataset (number of features)
- $localdim$, local dimensionality of the dataset (according to PCA)
- q, number of non-compact dimensions
- r, radius of the q-sphere surrounding the data projected into the non-compact dimensions
- $cdims$, boolean array with dimension p, indicating which features are compact
- $cranges$, float array with dimension $2 \times p$, indicating the minimum and maximum values of the compact features for the local dataset.

A `PSphere` object can test whether a new point x is inside its envelope using its `contains()` method, which returns True if two conditions are met: 1) each component

```
Input: local dataset $X$, cluster center $x_c$, dimensions $p$
Output: $localdim$, $q$, $r$, $cdims$, $cranges$
Initialize $cdims$ array to False, $cranges$ array to zero
Initialize $minwidth = 0.01$, EPS = 1.0e-7
# Recenter the local data on the cluster center
for $i$ in range($n$):
    $X'[i,:] = X[i,:] - x_c$
Run PCA on $X'$
$localdim$ = Number of PCA components that explain 90% of variance
Run Sparse PCA on $X'$ to retrieve $localdim$ components $\{spca\}$
# Identify compact dimensions and ranges
for $j$ in range($p$):
    if all($spca[j]$ == 0):
        $cdims[j]$ = True
        $cranges[0,j] = \min(X'[:,j])$
        $cranges[1,j] = \max(X'[:,j])$
# Ensure that no compact dimension has zero width
for $j$ in range($p$):
    if ($cdims[j]$ == True) and ($cranges[0,j] == cranges[1,j]$):
        $cranges[0,j] = cranges[0,j] - minwidth/2$
        $cranges[1,j] = cranges[1,j] + minwidth/2$
$cranges[0,:] = cranges[0,:]-$ EPS
$cranges[1,:] = cranges[1,:]+$ EPS
# Compute radius of q-sphere in non-compact dimensions
$q$ = Number of False components in $compact\_dims$
$S$ = submatrix of $X'$, the $q$ column vectors where $compact\_dims$ is False
$r = \max(\|S[i,:]\|)+$ EPS
```

Algorithm 1: Initialize the p-cylinder geometry for a local dataset X. The matrices X, X', and S are collections of n row vectors, indexed by i. The small constant factor EPS ensures that data points on the boundary of the envelope do not end up outside the p-cylinder due to numerical error in distance calculations.

of $x - x_c$ among the compact dimensions is within the interval defined by *cranges*, and 2) the projection of $x - x_c$ onto the non-compact dimensions has a magnitude less than r.

A p-cylinder can be generated with other algorithm settings by explicitly calling three methods in the following order after initializing the `PSphere` object `ps` on a dataset:

1. `ps.make_localdim(pct_threshold=0.95)` for a PCA variance threshold of 95%
2. `ps.make_sparse_basis(n_components=1)` to ask Sparse PCA to find just 1 component instead of *localdim* components.
3. `ps.make_pcylinder(min_compact_width=0.02)` to set a different minimum width when a compact dimension has zero range.

The result of applying Algorithm 1 to each cluster is a collection of overlapping p-cylinders that surround the point cloud without wasted volume in feature dimensions with low variability in their local manifolds. If the number of clusters is large, then the p-cylinders can trace the global topology of the point cloud. Some human supervision is necessary to select and adjust the clusters if a close fit to the topology is desired.

Figure 3 shows the collection of p-cylinders formed by applying the algorithm to a 4-dimensional dataset that has been subdivided into 20 clusters with K-Means. The clusters mostly align with the feature named "union centralization", and this variable is correctly identified as compact by the algorithm. Most of the p-cylinder geometries are either a 3-sphere extended by a small width into the union centralization dimension, or a 2-sphere extended by a small width into two compact dimensions including union centralization. In this particular dataset, it can be argued that the gaps in "union centralization" are due to low-precision measurements of a continuous variable. However, the data still serve as an example of the complex local geometries that we need to capture for trusted AI.

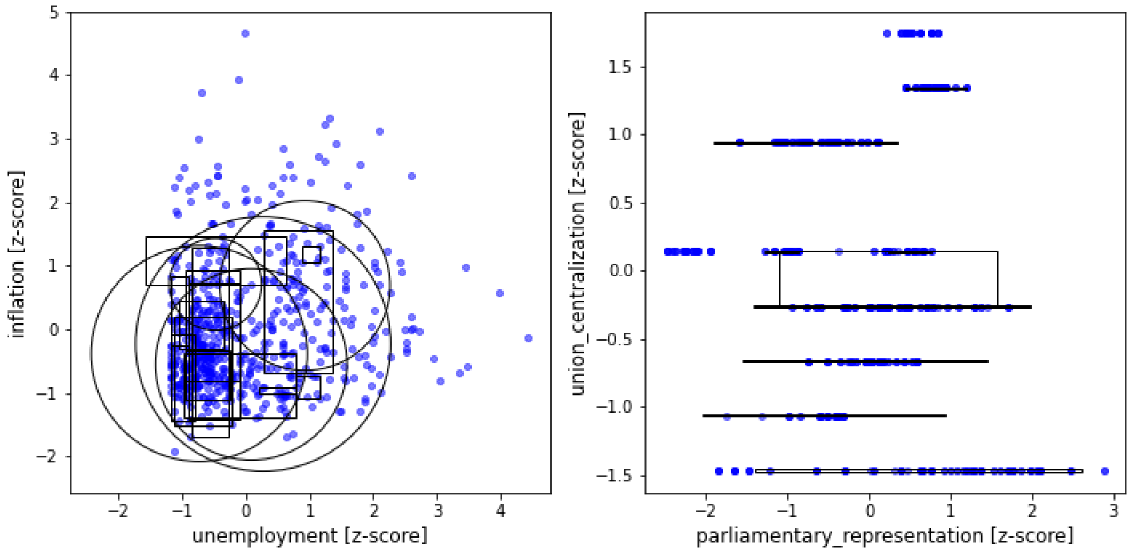

Figure 3: A four-dimensional dataset is divided into 20 clusters with K-Means, then a p-cylinder is defined for each cluster using Algorithm 1. Most of the clusters found by K-Means have a constant value of "union_centralization", so the algorithm recognizes this feature as a compact dimension. The projections of the 4-cylinders onto 2 axes are circles if neither feature is compact within the cluster, and rectangles of at least one feature is compact. The clustering algorithm was run on a subset of the data within a hypercube with side length 3.2; points outside this range are not in any cluster. The analysis of this dataset is described in Section 4.

3.4 Python modules

The package consists of two user-facing modules: `refine_cluster_set` and `PSphereHull`. The methods and attributes of these classes are summarized in Table 4 and Table 5 respectively.

The module `refine_cluster_set` implements functions to calculate the distribution of cluster volumes and change the clustering solution by hand after visualizing the distribution of data. For example, it allows you to subdivide a cluster into two subclusters and remove a single outlier from a cluster.

The module `PSphereHull` implements two classes: `PSphere` and `PSphereHull`. The user is expected to interact mainly with the `PSphereHull` class, which is a collection of p-spheres. This class includes methods to easily flag individual spheres as used or unused in the hull, and test new data points to see whether they fall inside any of the individual p-spheres.

Note that the tables and description in this paper are not meant to be complete documentation of the code. The PSPhereHull package, along with jupyter notebooks demonstrating its use, is available for download on the author's github:

https://github.com/bmathiesen/p-sphere-hull

4. Examples with Open Datasets

To demonstrate the value of the P-Sphere Hull method, I should be able to show that it effectively separates OOD data in the extrapolation regime from OOD data that are "sufficiently close" to the training data. In high dimensional point clouds with non-trivial topology, the distinction between inside and outside, or interpolation and extrapolation, is unclear. Multiple mathematical definitions could be proposed with different results. However, we can guess that if we are inside, or interpolating, the training data then the error distribution should be similar to the error distribution of the test set held out during training. On the other hand, if we are outside, or extrapolating the training data then the error distribution should be wider and more prone to extreme values.

To test this hypothesis, I looked for datasets in the OpenML repository (Vanschoren et al., 2013) with the following characteristics:

- The target variable is continuous, hence suitable for regression modeling. The main reason for focusing on regression is to be able to measure error distributions directly and unambiguously, but the package can also be used to define a domain model for classification problems.
- The explanatory features are mostly continuous.
- When the continuous features are plotted as a point cloud, its shape is irregular or its topology is non-trivial (see Section 2.1 for examples).
- The data are not artificially generated.
- The data are high-dimensional (more than 3 features).

Among the Regression30 collection, I found 4 datasets that met these criteria, with the number of features ranging from 4 to 16. I designed a standard pipeline to preprocess the data, cluster it, generate the P-Sphere Hull, and measure extrapolation errors, which is described below. Table 1 summarizes the characteristics of the four OpenML datasets and any special treatment required.

Dataset	No. Rows	No. Cols	p	Target and Notes
strikes	625	7	4	strike_volume [days] (1)
no2	500	8	7	no2_concentration [log n] (2)
cpu_small	7898	13	12	usr [users] (3)
house_16H	22732	17	16	log10 price [USD] (4)

Table 1: OpenML datasets used to test the PSphereHull package. The number of rows and columns refer to the raw dataset. The number p is the dimensionality of the dataset used for regression. Additional notes: (1) I dropped the categorical features country_code and year. (2) The target variable in the raw data is already the logarithm of the observed NO2 particle count. (3) I dropped rows with usr ≤ 10, because this subset is perfectly predictable from the freeswap variable. (4) I removed data with price = 0, because this subset is small and disjoint from the main distribution. Since the distribution of housing prices appeared to be exponential, I transformed the target from price to log10(price).

4.1 Data Processing

The data processing pipeline is mostly automatic, but still requires a few human choices. Each raw dataset downloaded from OpenML goes through the following steps:

1. The target variable is identified and separated.
2. Categorical features are identified and removed.
3. If the prediction is trivial or impossible for a subset of the data, that subset is removed.
4. Normalize each column to its mean and standard deviation (sklearn: StandardScaler)
5. Test each feature pair for strong correlations (Pearson's $\rho \geq 0.75$). I replace a correlated pair of features with its mean and difference. No strongly correlated triplets or larger groups of features were found.
6. Run K-Means and Gaussian mixtures on the dataset. Choose the clustering method that produces the smaller number of high-volume clusters (upper limit of the distribution), estimated by calculating their p-cylinder envelopes.
7. Split the dataset into "Core" and "OOD" subsets, using a hypercube of side s. (For example, $s = 2$ works well for the larger datasets, but see the next subsection for more details on this choice.)
8. Run the chosen clustering method on the Core dataset.

9. Create the P-Sphere Hull using the resulting data labels and cluster centers.
10. Run `find_redundant_spheres()` to remove sparse p-spheres with no unique data.
11. Save the prepared dataset, the core and OOD datasets, and the `PSphereHull` object.

Table 2 summarizes the properties of the Core and OOD samples for each dataset.

Dataset	Core N	Core Cut	OOD N	Hull Volume	Clustering
strikes	417	1.6	194	$5.6 \pm 0.7\%$	GM, $k = 20$
no2	238	1.6	262	$8.7 \pm 0.9\%$	GM, $k = 20$
cpu_small	5298	2.0	2600	$1.3 \pm 0.4\%$	GM, $k = 100$
house_16H	16045	2.0	6687	$8.4 \pm 0.9\%$	KM, $k = 50$

Table 2: Core dataset, OOD dataset, and P-Sphere Hull properties. Core Cut is the side length of the hypercube used to separated the Core sample from the OOD sample. Since all features are centered and normalized, the units of length are standard deviations. Hull Volume is estimated by randomly generating 1000 points within the Core dataset's hypercube and counting the number that fall inside the hull. The clustering method is either K-Means (KM) or Gaussian mixtures (GM).

4.2 Extrapolation Errors

To measure the importance of extrapolation error and the effectiveness of the P-Sphere Hull in detecting the extrapolation regime, I train a regression model (Random Forest) on 80% of the core dataset and then test the model on the remaining 20%, called the "Core Test" set. I also apply the model to the OOD dataset. The points in the OOD dataset are further subdivided into "Interpolation" and "Extrapolation" subsets based on whether the `contains()` method of the P-Sphere Hull flags them as inside or outside respectively. Figure 4 shows an example of the p-sphere hull created for the Core subset of the server data, in six of the twelve dimensions.

It is desirable to have a comparable amount of data in each category. However, depending on the topology of the Core dataset and the P-Sphere Hull, it might be more or less common to find points from the OOD dataset that fall inside the hull. If the setting $s = 2$ in step 7 of the pipeline resulted in too few points in the Interpolation or Extrapolation sets, then a new value was chosen by trial and error to improve the balance between the three sets. This is why for the two smaller datasets the pipeline uses $s = 1.6$ and not $s = 2$.

The models were first tested with a range of forest sizes (the number of decision trees, sklearn parameter n_estimators) to find a reasonable number that avoids overfitting. All other parameters of the model are left at their sklearn default values. Note that the purpose of this analysis is to find a simple benchmark model for measuring the error distribution, not to reduce the errors as much as possible.

Once the model is trained, we directly measure the error in all held-out datapoints. Table 3 reports the mean absolute error (MAE) of the three subsets. In one of the datasets (strike) the target variable can be zero, so I do not report relative errors; however, the conclusions of this experiment remain valid under this alternative metric.

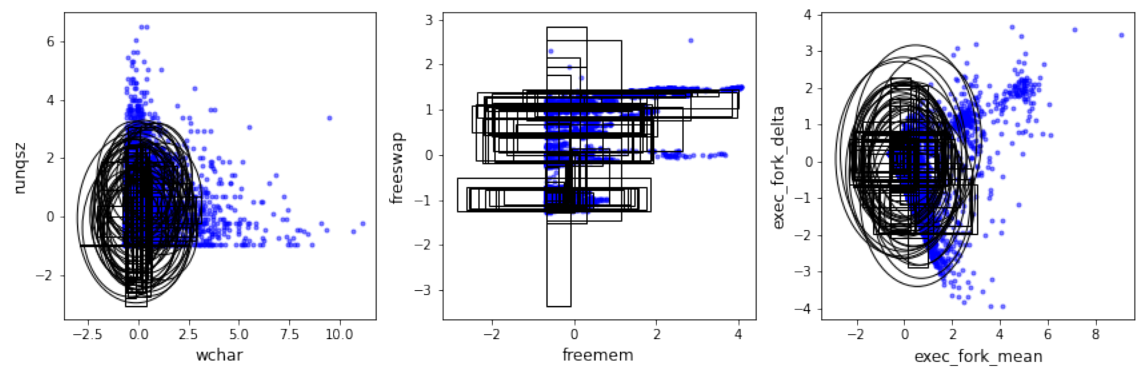

Figure 4: A "core" dataset is derived from the 12-dimensional server dataset by selecting points inside a two-sigma hypercube, then clustered into 100 subdomains with K-Means. A p-cylinder is created for each cluster using the method `PSphereHull.make_pcylinders()` (Algorithm 1). Redundant p-cylinders (those containing no unique points) are then identified and masked iteratively using the method `PSphereHull.flag_redundant_pspheres()`. 77 p-cylinders remain after this step. Most of the p-cylinders have 6 or 7 non-compact dimensions (out of 12). The figure shows three 2-D projections of the dataset. Points outside the two-sigma hypercube are also shown in the plot. Any point that does not fall inside at least one p-cylinder is identified as an out-of-domain extrapolation case.

Dataset	Test MAE (N)	Interp. MAE (N)	Extrap. MAE (N)
strikes	188 (87)	230 (27)	350 (167)
no2	0.397 (58)	0.386 (30)	0.472 (181)
cpu_small	1.88 (1060)	1.77 (868)	4.69 (1732)
house_16H	0.0102 (3209)	0.0130 (3242)	0.0179 (3445)

Table 3: OOD interpolation and extrapolation errors in the OpenML datasets

In all four examples, the MAE of the Extrapolation dataset is significantly higher than the other two datasets. Furthermore, the MAEs of the Core Test and Interpolated datasets are much closer together. For the two larger datasets, where the number of points in the Interpolation dataset is large, I quantify the difference in their cumulative error distributions with K-S tests (scipy method ks_2samp). For the server dataset, the p-values comparing the Extrapolation errors to the Core Test and Interpolation errors are 8.5e-87 and 1.4e-15 respectively, while the p-value comparing the Core Test errors to the Interpolation errors is only 8.8e-7. For the house_16H dataset, the p-values comparing the Extrapolation errors to the Core Test and Interpolation errors are 8.5e-120 and 1.4e-46 respectively, while the p-value comparing the Core Test errors to the Interpolation errors is only 8.8e-15. Hence while all three distributions are clearly distinguishable from one another, the Extrapolation error distribution clearly has the most unique shape.

4.3 Results

Based on these results, I make the following observations.

- The model errors associated with new data outside the p-sphere hull form a wider distribution with a larger mean value. This demonstrates the unsuprising fact that extrapolation errors are larger than interpolation errors overall, and the danger of industrializing AI applications where the conditions of extrapolation have not been studied.

- The error distributions associated with the Interpolation and Core Test datasets are similar. I do not claim they are identical, but the closeness of their mean MAE values is striking and the K-S tests show that the p-value comparing Core Test to Interpolation is orders of magnitude higher than the p-values comparing both to Extrapolation.

- The p-sphere hull appears to fulfil its promised function of providing a useful model of the geometrical boundary between the interpolation and extrapolation domains in a high-dimensional feature space.

In practice, after testing the p-sphere on the four datasets listed above as well as several industrial (hence private) datasets, I see two distinct use cases for this method. The first occurs when the training dataset has a relatively simple global topology, such as an irregular point cloud without many gaps or low-dimensional substructures. In this case the P-Sphere Hull is a practical alternative to the convex hull of the dataset. The P-Sphere Hull has some major computational advantages over the convex hull: it easy to create and visualize, and it is cheap to test for membership and distance. The second use case applies when the dataset has a complex topology. The p-cylinders are capable of efficiently surrounding low-dimensional manifolds and gaps in the data, as shown in Figures 3 and 4. However, building a hull that closely follows the domain topology is still a handicraft process. It requires visualizing the clusters individually, then modifying some clusters and masking others to better approximate the shape of the point cloud.

I also looked for correlations between the errors and two metrics describing the point's relation to the P-Sphere Hull: distance from the hull (i.e., the smallest distance to any p-sphere boundary) and the density of the closest p-sphere. I found no significant correlations with either metric in any of the four datasets studied. The reason, I believe, is the high degree of overlap between the p-spheres composing the hull. In building the Core dataset, I place most of the interesting topological features in the OOD dataset. Within the Core dataset, even though each p-sphere is associated with a unique set of points, their surfaces are large enough to include many points from nearby spheres. Therefore, for a given OOD point, many p-sphere surfaces have nearly the same distance, so any correlations with distance from the hull and p-sphere density are obscured. My work in this area continues and I am hopeful to find that extrapolation error, and hence business risk, can be correlated with a metric that puts the OOD datapoint in relation with the P-Sphere Hull.

5. Literature Review

5.1 Explainable AI vs trusted AI

Academic interest in explainable AI (XAI) has grown remarkably in recent years, alongside the development of deep learning models with powerful capabilites but millions or billions of free parameters. The general idea of XAI is to understand how and why AI models make their decisions, often in terms of feature importance (global explanations) or interpretable, few-parameter models that approximate the behavior of the network around a single test sample (local explanations). This line of research includes a constellation of desirable AI properties such as fairness, robustness, reliability, interpretability, and trust. I will not attempt a comprehensive review of the XAI literature in this paper; several excellent works already exist such as a recent survey article by (Burkart & Huber, 2021) and Molnar's e-book (Molnar, 2021). However, to place the present work in context, I will repeat three key concepts:

- An *explainable* model is one whose mathematical operations can be analyzed and simplified so that humans can at least partially understand the decision. This area includes a wide variety of techniques, many of which focus on post hoc, local explanations for a specific AI decision. Industry actors often deploy XAI tools to support model development and testing, for example IBM's FreaAI (Ackerman et al. (2021)), which can identify subdomains of the data with a high concentration of inaccurate predictions.
- An *interpretable* model is one that provides human-readable decisions by nature. For example, expert systems, decision trees, fuzzy logic, and generalized linear models could all be considered interpretable.
- A *trusted* model is one that provides enough context and explanation for humans to have confidence in its decisions. An explainable model is not necessarily trustworthy, or vice versa; the key components of trusted AI more often lie in dataset design, feature engineering, and ethical considerations than in the model itself.

The focus of this work is on building trusted AI models. I consider that a crucial component of trusted AI, indeed the *essential minimum*, is that the model knows the domain of the dataset used for training. An AI application that has this information can then warn the user when it attempts to predict a value for a point outside this domain. The challenge is doing this for high-dimensional datasets with non-trivial topologies. In the literature, this extrapolation problem is also referred to as out-of-domain (OOD) error, and it is a perennial problem in industrialized models where the distribution of the data can evolve over time. So far, the practical solution has been to retrain the model periodically, but no good solutions exist for modeling domains with complex topology.

5.2 The extrapolation problem in machine learning

In contrast with XAI, there is sparse literature in the AI domain on the underlying problem addressed by this paper: characterizing extrapolation in high-dimensional datasets. However, two recent papers have shared some interesting and complementary results.

Barbiero, Squillero, and Tonda (2020) study the relationship between statistical and topological characteristics of public datasets and the ability of classifier-type models to generalize outside their training data. They characterize extrapolation as the case where a test sample lies outside the convex hull of the training dataset, and show that the performance of the classifiers is consistently worse when they must extrapolate. They also show that using linear programming methods, it is possible to detect whether a new point lies outside the convex hull even in a very high-dimensional dataset, without explicitly modeling the shape of the hull itself. Hence, this is a promising, low-cost approach to identifying OOD test samples, but extrapolation can also occur inside the convex hull for topologically complex domains.

Madras, Atwood, and D'Amour (2019) describe a local, post-hoc method for detecting high-uncertainty predictions using the Hessian matrix of the model. They argue that model predictions will be under-determined in a feature subspace defined by the largest eigenvectors of the Hessian, and demonstrate how to generate a local ensemble of models with equivalent performance on the local training data to estimate the prediction uncertainty on a new data point. They designed and tested their "local ensemble" method for models such as deep neural networks, which are often under-determined locally because the number of parameters is extremely high. In contrast, the P-Sphere Hull assumes that a correctly generalized model will sometimes be asked to make predictions in a region of feature space where no training data are available. Therefore, the two approaches are complementary.

Finally, several machine learning methods support anomaly detection, such as one-class SVM and Isolation Forests. There is a close relationship between anomaly detection and OOD detection, and one could easily interpret the probability or anomaly score of such methods as a proxy for extrapolation risk. However, there are two issues with using such methods as an explicit domain model. First, since they are trained on the initial dataset, some useful training data will be assigned anomaly status. Sparse subdomains can still drive accurate predictions if the target variable is changing slowly, so local density is not a sufficent criterion for rejection. Practically, this means that a domain model based on anomaly detection methods will be too restrictive, probably limited to the densest parts of the point cloud. Second, these methods have several hyperparameters that should be tuned and optimized, compared to the P-Sphere Hull approach which has only one tunable parameter (the explained variance threshold for identifying compact dimensions with PCA and Sparse PCA).

5.3 The applicability domain in QSAR models

One area of research that has given serious attention to the extrapolation problem is QSAR (quantitative structure–activity relationship) models, which are statistical and machine learning models that predict whether a given molecule will fulfil a specific physical function based on its atomic structure. QSAR models are high-dimensional by nature, since the features represent physical and topological characteristics of a complex 3D structure. The pharmaceutical industry has a vested interest in ensuring that laboratory research is not wasted on molecules whose predictions are OOD and hence untrustworthy (i.e., the input parameters are outliers with respect to the training dataset). In this specialty, the region where reliable predictions can be made is called the *applicability domain* (AD).

In fact, the *OECD Guidelines for (Q)SAR validation* recommend that all QSAR models in production include a formal definition of their AD (Organisation for Economic Cooperation and Development, 2007; Tetko et al., 2008). The main methods proposed by the OECD guidelines are:

- Defining an enclosing surface, such as a covariance ellipse, a convex hull, or a hypercuboid. The guidelines note that working with a convex hull can be difficult in high dimensions, and warn practitioners that any enclosing surface can still contain sparsely populated subvolumes where prediction is more difficult.

- Calculating a distance metric for each point with respect to the "center" of the training dataset, such as the Mahalanobis distance. Again, the OECD guidelines warn that regardless of the distance metric chosen, it cannot be taken as a sole indicator of whether a new data point is "inside" or "outside" the AD.

- Using a probability density estimator to map the density contours of the training dataset and calculate the probability that a new datapoint is "inside" the domain. The OECD guidelines remark that the method is computationally intensive, and again not guaranteed to work because prediction accuracy is not directly related to local sampling density.

The QSAR community benefits from mature statistical software packages to implement these guidelines. However, in order to guarantee reliable predictions, the AD of a QSAR model is often reduced to a convex, high-density sub-volume of the feature space. In other words, valid training data are often excluded from the AD simply because it is harder to guarantee the accuracy of nearby predictions.

5.4 OOD detection in deep learning

In deep learning (DL) the extrapolation problem is usually framed as a generalization problem: DL models have trouble maintaining accuracy and providing reliable uncertainty estimates for out-of-domain (OOD) samples. DL models are also under-determined, which implies that their behavior outside the training domain is unconstrained and varies widely from one training instance to the next. Researchers have proposed novel architectures and training methods to help DL models detect OOD cases, a few of which are listed below.

Uncertainty estimation: A recent tutorial by the Google Brain team at NeurIPS (2020) reviews several uncertainty estimation methods for DL. Two well-established approaches are Bayesian NNs and ensembles of NNs, both of which can be used to estimate the probability distribution of a prediction, thereby overcoming the tendency of a single DL network to be overconfident outside the training domain (Liu, Lin, Padhy, Tran, Weiss, & Lakshminarayanan, 2020). Another approach is to use local density estimators such as deep k-NN (Papernot & Mcdaniel, 2018).

Integrated clustering: Two recent papers describe NN architectures that model the geometry of the training domain as a set of clusters that correspond to the supervised classification labels. Van Amersfoort et al. (2020) propose a new method of training a radial basis function (RBF) NN, so that the distances from class centroids can provide a realistic uncertainty estimate for test samples far from the training domain. Their innovation permitted the RBF NN to learn a stable class distribution, which has previously been difficult for this type of architecture. Tan et al. (2019) describe an application of prototypical NNs for few-shot domain modeling and zero-shot OOD detection in short text documents. The idea behind the prototypical NN is that an embedding space can be found where the known classes are modeled by compact clusters, so that the distances from their centroids are meaningful. These two approaches create an implicit domain model that is similar in spirit to the P-Sphere Hull, but with two crucial differences. In the NN models, the clusters representing the training domain correspond to the training classes, whereas the purpose of P-Sphere Hull is to create a set of clusters that approximates the underlying geometry of the domain. Another important difference is that in feature space the point clouds of training classes can have complex shapes; there is no guarantee that in the embedding space the corresponding clusters are spherical or even compact.

Surprise detection: Another line of DL research uses introspection to detect unusual behaviors of the network itself. The Surprise Adequacy for Deep Learning metric (Kim, Feldt, & Yoo, 2019) uses the activation vector of a deep layer to detect anomalies. The hypothesis that unusual activation vectors result from unusual inputs is plausible, and upheld by experiments. However, the converse is not necessarily true: an atypical input will not necessarily map to an atypical activation vector, since the embedding space is lower dimensional and necessarily discards information. Therefore, while surprise metrics certainly provide a useful check on overconfidence, they are an incomplete model for the training domain. Another metric in this category is the Fisher information matrix, which summarizes the activation behavior of an entire DL network with respect to a classification problem (Martin & Elster, 2020).

All of these contributions attempt to learn the domain model and fit the training samples simultaneously. This approach increases the training cost and makes it difficult to transfer the domain model to other prediction problems on the same dataset. It also introduces a dependence on the prediction variable (e.g. the training class labels) that is unnecessary for modeling the geometry of the domain. Another limitation of the DL research is that it tends to focus on classification problems, which may be less sensitive to extrapolation error than regression problems if the test data lie far from the decision boundary. Regression problems are very common in industrial settings; there is a very real need to minimize the extrapolation error of machine learning when predicting energy efficiency, click-through rates, and clinical study costs, to name a few examples.

6. Conclusions and Future Work

The P-Sphere Hull provides an inexpensive framework for modeling the training domain that can be used with both classical machine learning and deep learning methods. Its goal is to provide an explicit method for OOD detection along with complete coverage of the training domain, without relying on the target variable distribution or the predictive model in any way. In practice, the training domain model, in the form of a p-sphere hull or some variation thereof, can be deployed in parallel with an AI application to screen data before making a prediction, thereby delivering "zero-trust" AI applications that do their own error checking in real time.

As currently implemented, the method still has some key limitations. For one, the geometry of the cylinders is always aligned with the feature axes. De-correlating can help by orienting the dataset to new feature axes without information loss; however, most datasets still contain low-dimensional subdomains oriented at arbitrary angles. A p-cylinder drawn around such a sub-domain can contain a large hypervolume that is empty of data. The best alternative to p-cylinders in this case would be a PCA-rotated hypercuboid. It is not difficult to implement such a "P-Box Hull" based on the same principles as the P-Sphere Hull, but care must be taken not to make the boxes too restrictive. A p-box has $2p$ faces, so we can expect $2p$ points of each subdomain to lie on the boundary of the box. The boxes can be enlarged, but by how much? The appropriate amount should depend on the local density of points and the gradient of the target variable. Another limitation is that creating a suitable p-sphere hull for complex topologies still requires a fair amount of human supervision. The computational cost of testing for membership in a rotated box is also higher, but perhaps not prohibitive.

Another limitation of the current work is that I have focused on datasets with only continuous features. If only a small number of features are binary or categorical, the p-sphere hull method can recognize them as compact dimensions and create envelopes that avoid the gaps. However, if most of the features are categorical then the problem becomes a binary optimization problem to which this method is ill suited.

The method is designed to be independent of the clustering solution used to create the subdomains. In practice I have found that standard clustering methods nearly always find some high-volume, sparse clusters that overlap and obscure the smaller, more compact p-cylinders that trace the local geometry. These can be removed or reduced by hand; however, I would like to find an upstream clustering approach that does a better job of avoiding gaps in the data. For example, building a two-stage pipeline with UMAP and Gaussian Mixtures might work better, if the scatter in the dataset does not obscure too much the local geometry of the manifold. Another possible approach is to implement agglomerative clustering with a strong penalty to adding points that cross a gap in any one-dimensional feature distribution. This line of work would also have to overcome the obstacle of low-precision continuous features that have meaningless gaps in their distributions.

Finally, it is important to test the p-sphere hull method on a much larger set of open datasets, to increase confidence in the method for both of the use cases mentioned in Section 4 (convex hull approximation and modeling complex topology). During this work I also intend to compare the P-Sphere Hull method to anomaly detection methods and other training domain envelopes, measuring performance in terms of computational cost and their ability to separate the interpolation regime from the extrapolation regime on test data. For industrial applications, I am also working on examples to demonstrate the business risk associated with extrapolation errors identified through the p-sphere hull.

In conclusion, I see this paper as a successful demonstration of a key principle: that AI models in production must have a way of modeling the geometry of their training data in feature space, so as not to expose the end users to unacceptable risk. We can achieve this goal even in high-dimensional datasets by applying a "divide and conquer" approach: first break the dataset into a collection of manageable subdomains, then model the geometry as a collection of geometrically simple envelopes. The P-Sphere Hull package is one possible implementation of this strategy. I invite the community to explore others.

Acknowledgements

This work was completed with the support of Hybrid Intelligence (formerly Tessella), an organization within Capgemini Engineering whose mission is to revolutionize industrial R&D by fusing human creativity, science, technology and data.

References

Ackerman, S., Dube, P., Farchi, E., Raz, O., & Zalmanovici, M. (2021). Machine learning model drift detection via weak data slices. *CoRR*, *abs/2108.05319*.

Barbiero, P., Squillero, G., & Tonda, A. P. (2020). Modeling generalization in machine learning: A methodological and computational study. *CoRR*, *abs/2006.15680*.

Burkart, N., & Huber, M. (2021). A survey on the explainability of supervised machine learning. *JAIR*, *70*.

Chakraborty, A., Alam, M., Dey, V., Chattopadhyay, A., & Mukhopadhyay, D. (2021). A survey on adversarial attacks and defences. *CAAI Transactions on Intelligence Technology*, *6*(1), 25–45.

Cheng, L., Varshney, K. R., & Liu, H. (2021). Socially responsible ai algorithms: Issues, purposes, and challenges. *JAIR*, *71*.

Google AI (2021). Responsible AI Practices..

Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., & Madry, A. (2019). Adversarial examples are not bugs, they are features. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E., & Garnett, R. (Eds.), *Advances in Neural Information Processing Systems (NeurIPS 2019*, Vol. 32. Curran Associates, Inc.

Jenatton, R., Obozinski, G., & Bach, F. (2010). Structured sparse principal component analysis. In Teh, Y. W., & Titterington, M. (Eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Vol. 9 of *Proceedings of Machine Learning Research*, pp. 366–373, Chia Laguna Resort, Sardinia, Italy. JMLR Workshop and Conference Proceedings.

Kim, J., Feldt, R., & Yoo, S. (2019). Guiding deep learning system testing using surprise adequacy. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, p. 1039–1049. IEEE Press.

Liu, J., Lin, Z., Padhy, S., Tran, D., Weiss, T. B., & Lakshminarayanan, B. (2020). Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. In *Advances in Neural Information Processing Systems Proceedings (NeurIPS 2020)*, 33, Virtual. NIPS.

Madras, D., Atwood, J., & D'Amour, A. (2019). Detecting extrapolation with local ensembles. *CoRR*, *abs/1910.09573*.

Martin, J., & Elster, C. (2020). Inspecting adversarial examples using the fisher information. *Neurocomputing*, *382*, 80 – 86.

McInnes, L., Healy, J., & Melville, J. (2020). Umap: Uniform manifold approximation and projection for dimension reduction..

Molnar, C. (2021). *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable.*

Organisation for Economic Cooperation and Development (2007). *Guidance Document on the Validation of (Q)SAR Models.*

Papernot, N., & Mcdaniel, P. (2018). Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *ArXiv*, *abs/1803.04765*.

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., & Édouard Duchesnay (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12*(85), 2825–2830.

Roach, J. (2020). Microsoft responsible machine learning capabilities build trust in ai systems, developers say..

Roweis, S. T., & Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *Science*, *290*(5500), 2323–2326.

Tan, M., Yu, Y., Wang, H., Wang, D., Potdar, S., Chang, S., & Yu, M. (2019). Out-of-domain detection for low-resource text classification tasks.. pp. 3557–3563.

Tetko, I. V., Sushko, I., Pandey, A. K., Zhu, H., Tropsha, A., Papa, E., Öberg, T., Todeschini, R., Fourches, D., & Varnek, A. (2008). Critical assessment of qsar models of environmental toxicity against tetrahymena pyriformis: Focusing on applicability domain and overfitting by variable selection. *Journal of Chemical Information and Modeling*, *48*(9), 1733–1746. PMID: 18729318.

Van Amersfoort, J., Smith, L., Teh, Y. W., & Gal, Y. (2020). Uncertainty estimation using a single deep deterministic neural network. In III, H. D., & Singh, A. (Eds.), *Proceedings of the 37th International Conference on Machine Learning*, Vol. 119 of *Proceedings of Machine Learning Research*, pp. 9690–9700, Virtual. PMLR.

Vanschoren, J., van Rijn, J. N., Bischl, B., & Torgo, L. (2013). Openml: Networked science in machine learning. *SIGKDD Explorations*, *15*(2), 49–60.

Ward, M. O., & Guo, Z. (2010). Generalized Hyper-cylinders: a Mechanism for Modeling and Visualizing N-D Objects. In Hagen, H. (Ed.), *Scientific Visualization: Advanced Concepts*, Vol. 1 of *Dagstuhl Follow-Ups*, pp. 1–10. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany.

Zou, H., Hastie, T., & Tibshirani, R. (2004). Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, *15*.

Table 4: User-facing functions of the module `refine_cluster_set`

Function	**Purpose**
`flag_outlier(data, labels, centers, cid)`	Finds the data point in cluster cid farthest from the cluster center, and replaces its label with −1.
`fuse_clusters(data, labels, centers, cid1, cid2)`	Merges clusters cid1 and cid2 by relabeling all points in cid2 with cid1, then recalculating the cluster center of cluster cid1.
`split_cluster(data, labels, centers, cid)`	Runs KMeans(2) on the data in cluster cid1, relabels the points in one subcluster with a new integer id, and updates the list of cluster centers to include the new cluster.
`split_linear_cluster()`	Helper function for `refine_linear_clusters()`
`split_all_linear_clusters()`	Helper function for `refine_linear_clusters()`
`refine_linear_clusters(data, labels, centers, min_size, pct_threshold)`	Calculates the local dimensionality of every cluster using PCA, then runs split_cluster on any cluster whose data are approximately linear. The function recursively splits new clusters with at least min_size data.
`describe_cluster_pspheres(data, labels, centers, use_pcylinders)`	Generates a pandas dataframe with volume and density statistics on the p-spheres and hypercuboids that bound the individual clusters.
`plot_cluster_pspheres(df, pdim, arg1, arg2, filename)`	Generates a matplotlib log-log plot of two columns (arg1 and arg2) named in the dataframe df generated by `describe_cluster_pspheres`.
`pcylinder_patch (df, pdim, arg1, arg2, filename)`	Generates a matplotlib artist patch representing a p-cylinder in a two-dimensional projection of the feature space.

Table 5: Attributes and methods of the `PSPhereHull` class

Attribute or Method	**Definition**
`PSphereHull(data, labels, centers)`	The constructor generates a PSphere object for each cluster (defined by the arrays `labels` and `centers`) in the dataset (`data`).
`pdim`	number of dimensions in the dataset
`n_spheres`	number of p-spheres in the hull
`spheres`	[PSphere] a list of p-spheres
`hc_min`	[float] lower bounds of the individual dataset features
`hc_max`	[float] upper bounds of the individual dataset features
`hc_volume`	hypercuboid volume of the full dataset
`localdims`	[int] local dimensionalities of the p-spheres found by PCA
`pcylinders`	[bool] indicates that all PSphere objects have calculated p-cylinders
`naive_vratio`	sum of p-sphere volumes divided by hc_volume
`sampled_vratio`	fraction of randomly sampled points inside any p-sphere
`sampled_vratio_sigma`	sqrt(N) uncertainty on sampled_vratio
`naive_pcylinder_vratio`	sum of p-cylinder volumes divided by hc_volume
`sampled_pcylinder_vratio`	fraction of randomly sampled points inside any p-cylinder
`sampled_pcylinder_vratio_sigma`	$\sqrt{N}$ uncertainty on sampled_pcylinder_vratio
`make_local_dimensions()`	recalculates the localdim attribute for all individual p-spheres and the localdims attribute for the PSphereHull.
`contains(new_data)`	tests a collection of row vectors for membership in the PSphereHull, returns a list of booleans.
`find_containing_spheres(vector)`	tests a row vector for membership in every PSphere, returns the labels of containing p-spheres as a tuple.
`make_volume_ratios(n_samples)`	calculates all attributes with the suffix vratio. If n_samples=0, it skips the sampled attributes.
`psphere_distances(new_data)`	tests a collection of row vectors for membership in the PSphereHull, returns a $n \times k$ array of distances of each row vector from the k PSpheres in the hull.
`make_pcylinders()`	calculates the p-cylinder dimensions for each p-sphere if it hasn't already been done.
`mask_spheres(cids)`	sets the used attribute to False for the spheres indicated in cids, a list of ints
`use_spheres(cids)`	sets the used attribute to True for the spheres indicated in cids, a list of ints
`sphere_is_redundant(cid:int)`	for each point in the cluster of PSphere cid, find the containing spheres. Returns True if the PSphere contains no unique points, and a list of p-spheres sharing points with cid.
`flag_largest_redundant_sphere()`	helper function for `flag_redundant_spheres()`.
`flag_redundant_spheres()`	test each p-sphere to see whether it has any unique data points, from least to most dense. If a sphere has no unique data points, flag it with used=False.
`apply_standard_pipeline()`	functions to run automatically when the P-Sphere Hull is initialized. Set use_pcylinders=True and compute_all=True to calculate volumes, define the p-cylinders, and mask redundant p-cylinders.