Modified Kernel Principal Component Analysis and Autoencoder Approaches to Unsupervised Anomaly Detection

Nicholas S Merrill

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Mechanical Engineering

Azim Eskandarian, Chair
Saied Taheri
Alfred L. Wicks

April 30, 2020
Blacksburg, Virginia

Keywords: Anomaly Detection, Unsupervised, Outlier, Kernel Principal Component Analysis, Autoencoder, Machine Learning, Deep Learning

Modified Kernel Principal Component Analysis and Autoencoder Approaches to Unsupervised Anomaly Detection

Nicholas S Merrill

(ABSTRACT)

Unsupervised anomaly detection is the task of identifying examples that differ from the normal or expected pattern without the use of labeled training data. Our research addresses shortcomings in two existing anomaly detection algorithms, Kernel Principal Component Analysis (KPCA) and Autoencoders (AE), and proposes novel solutions to improve both of their performances in the unsupervised settings. Anomaly detection has several useful applications, such as intrusion detection, fault monitoring, and vision processing. More specifically, anomaly detection can be used in autonomous driving to identify obscured signage or to monitor intersections. Kernel techniques are desirable because of their ability to model highly non-linear patterns, but they are limited in the unsupervised setting due to their sensitivity of parameter choices and the absence of a validation step. Additionally, conventionally KPCA suffers from a quadratic time and memory complexity in the construction of the gram matrix and a cubic time complexity in its eigendecomposition. The problem of tuning the Gaussian kernel parameter, σ, is solved using the mini-batch stochastic gradient descent (SGD) optimization of a loss function that maximizes the dispersion of the kernel matrix entries. Secondly, the computational time is greatly reduced, while still maintaining high accuracy by using an ensemble of small, *skeleton* models and combining their scores. The performance of traditional machine learning approaches to anomaly detection plateaus as the volume and complexity of data increases. Deep anomaly detection (DAD) involves the applications of multilayer artificial neural networks to identify anomalous examples. AEs

are fundamental to most DAD approaches. Conventional AEs rely on the assumption that a trained network will learn to reconstruct normal examples better than anomalous ones. In practice however, given sufficient capacity and training time, an AE will generalize to reconstruct even very rare examples. Three methods are introduced to more reliably train AEs for unsupervised anomaly detection: Cumulative Error Scoring (CES) leverages the entire history of training errors to minimize the importance of early stopping and Percentile Loss (PL) training aims to prevent anomalous examples from contributing to parameter updates. Lastly, early stopping via Knee detection aims to limit the risk of over training. Ultimately, the two new modified proposed methods of this research, Unsupervised Ensemble KPCA (UE-KPCA) and the modified training and scoring AE (MTS-AE), demonstrates improved detection performance and reliability compared to many baseline algorithms across a number of benchmark datasets.

Modified Kernel Principal Component Analysis and Autoencoder Approaches to Unsupervised Anomaly Detection

Nicholas S Merrill

(GENERAL AUDIENCE ABSTRACT)

Anomaly detection is the task of identifying examples that differ from the normal or expected pattern. The challenge of unsupervised anomaly detection is distinguishing normal and anomalous data without the use of labeled examples to demonstrate their differences. This thesis addresses shortcomings in two anomaly detection algorithms, Kernel Principal Component Analysis (KPCA) and Autoencoders (AE) and proposes new solutions to apply them in the unsupervised setting. Ultimately, the two modified methods, Unsupervised Ensemble KPCA (UE-KPCA) and the Modified Training and Scoring AE (MTS-AE), demonstrates improved detection performance and reliability compared to many baseline algorithms across a number of benchmark datasets.

Dedication

To all the people in my life who were a constant source of support and encouragement over these last two year.

Acknowledgments

I wish to thank my committee members, Dr. Saied Taheri and Dr Alfred Wicks, who both demonstrate a commitment to education and a generosity with their time and experience. A special thanks to my advisor and chair, Dr. Azim Eskandarian, who guided and supported me throughout this process and has always made time for his students. I would also like to thank Dr. Colin Olson who I learned an enormous amount from during my time at the Naval Research Lab. Finally, I would like to thank the other members of the ASIM lab for their helpful suggestions and feedback.

Contents

List of Figures

List of Tables

List of Abbreviations

$\boldsymbol{\Phi}$	feature space transformation function
$\boldsymbol{\Sigma}$	Covariance Matrix
$\boldsymbol{X}$	dataset
$\boldsymbol{x}$	a single example point in the dataset
κ	kernel function
$\mathcal{F}$	Feature Space
$\mathcal{L}$	Loss
σ	Gaussian Kernel Bandwidth
B	knee-multiple stopping parameter
b	bias
D	number of features
d	L_2 Euclidean Distance
e_b	number of *burn-in* epochs
i_D	index of dispersion
J	number of epochs
K	kernel Gram matrix

k	number of nearest neighbors
m	sample mean
N	number of examples
N_b	subsampling batch size
N_m	number of models used in the ensemble
N_s	number of examples used to form a *skeleton* subsampled kernel matrix
q	number of retained Principal Components
s	sample standard deviation
z	latent representation of an example
AE	Autoencoder
AP	Average Precision
AUC	Area Under Curve (ROC)
CES	Cumulative Error Scoring
DAD	Deep Anomaly Detection
DL	Deep Learning
DNN	Deep Neural Network
FN	False Negative
FP	False Positivee
FPR	False Positive Rate

iForest Isolation Forest

KDE Kernel Density Estimator

KPCA Kernel Principal Component Analysis

KPCA Kernel Principal Component

LOF Local Outlier Factor

ML Machine Learning

MSE Mean Squared Error

MTS-AE Modified Training and Scoring Autoencoder

NN Neural Network

OC-SVM One Class Support Vector Machine

PCA Principal Component Analysis

PCA Principal Component

PL Percentile Loss

ROC Reciever Operating Characteristic (Curve)

SVDD Support Vector Data Description

TN True Negative

TP True Positive

TPR True Positive Rate

UE-KPCA Unsupervised Ensemble Kernel Principal Component Analysis

Chapter 1

Introduction

1.1 Background

The goal of an anomaly (outlier) detection methods is to detect anomalous points within a dataset dominated by the presence of ordinary background points. Machine learning (ML) methods are commonly employed to analyze datasets to uncover anomalies. ML has been most successfully applied to supervised tasks in which labeled data is available during training. In anomaly detection, labels indicate whether a training example is considered anomalous or normal. However, Anomalies are by definition rare and are often generated by different or unknown underlying processes [26, 32]. Consequently, the conventional paradigm of supervised learning is not well suited to anomaly detection because obtaining a sufficient number of labeled anomalous examples is often infeasible [13, 32, 74].

Semi-supervised approaches to anomaly detection still require training data but attempt to circumvent the need for labeled anomalous examples by only using more readily available normal examples. Semi-supervised methods first attempt to model the single class of normal examples. Then, examples that do not conform to the model are identified as anomalous. Unfortunately, semi-supervised techniques are susceptible to over-fitting, or under-fitting; the effect of which is poor precision or recall, respectively. [13, 32, 89].

The most flexible assumption is that of unsupervised anomaly detection where no labels are

available. The training data is not *clean* and may contain anomalies or challenging normal examples. Instead, the training process alone attempts to separate normal and anomalous examples based on the intrinsic properties of the data. Unsupervised techniques are necessary if labeled data cannot be reasonably obtained or when patterns distinguishing anomalous and normal behavior change irregularly over time [74]. Figure 1.1 provides a visual description of the differences between the supervised, semi-supervised, and unsupervised anomaly detection settings.

Even in domains where semi-supervised or supervised methods may be feasible, unsupervised methods can be deployed to more readily label normal or anomalous examples. Unsupervised techniques can also be used as a preprocessing or boosting step [101]. For conventional supervised classification problems, removing or weighting anomalous examples in a training set can produce significant improvements in accuracy [90]. Unsupervised methods can also be used to automatically identify *representation bias* in training data by highlighting patterns in the normal examples. For example, if most images of dogs in a training set have grass in the background, then a classifier might key in on the grass features and ignore the relevant target, causing a lack of generalization that would not be recognized from a holdout validation set [55]. An unsupervised anomaly detector can assign the more abundant grass-background examples of the dog class lower *anomaly scores*. A practitioner could then use this knowledge to address the bias before training the supervised classifier.

Many approaches have been proposed to address the problem of unsupervised anomaly detection. Traditional approaches can be thought of as belonging to the following categories: statistical, proximity-based, subspace-based, or separation-based methods [32, 95]. The taxonomy is loosely defined, often methods combine techniques from multiple categories. Traditional approaches to anomaly detection, as with other machine learning problems, tend to work well when the amount of data is small in number and dimensionality. However,

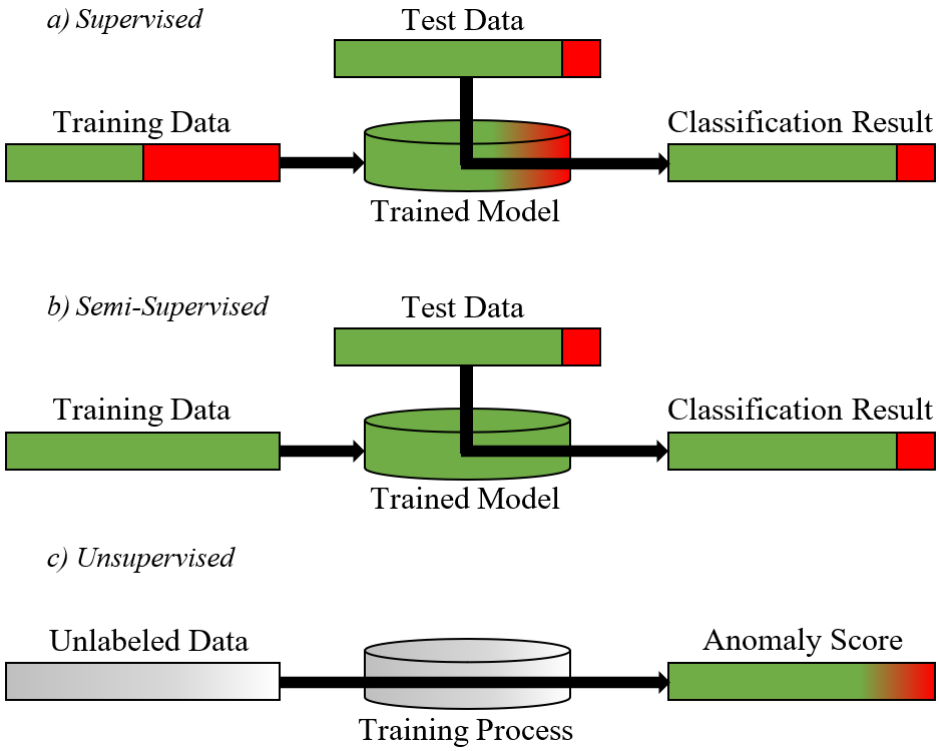


Figure 1.1: Comparing the different modes depending on the availability of training data: a) Supervised anomaly detectors uses a fully labeled dataset containing anomalous and normal examples during training and returns a classification b) Semi-supervised anomaly detectors use only normal examples during training c) Unsupervised anomaly detection takes unlabeled data and produces an anomaly score. Adapted from [31]

deep learning-based, approaches are often necessary in order to scale to larger datasets and higher dimensional inputs, such as images and sequential data. *Deep* refers to the multi-hidden-layer arrangements of artificial neural networks (NN). The success of Deep Anomaly Detection (DAD) techniques can be traced to their ability to automatically extract hierarchical features from the successive hidden layers. This has the additional benefit of removing the need for manual feature extraction by domain experts and allows models to be trained in an end-to-end manor from raw inputs.

Both traditional and deep methods of unsupervised anomaly detection must be able to identify different types of anomalies. At the highest level of abstraction, an anomaly is an example that does not conform to normal behavior [26]. In practice, applying this idea is challenging. There are several cases where the separation between anomalous and ambiguous data is ambiguous. Figure 1.2 illustrates the challenge of classifying anomalies.

The points a_1 and a_2 can more easily be identified as global anomalies, as they strongly deviate from any clusters of normal examples. However, a_3 requires more careful consideration. Because it is not clearly distinct from the nearest cluster, it could be considered a normal example that is part of N_2, however, when observed only in context of its local neighborhood, a_3 appears *locally* anomalous [10, 16]. Another more subjective example is presented by N_3, which could be considered as a small cluster of normal data, or as three coincidentally group anomalies. Furthermore, the figure illustrates how anomalies can often only be identified by a collection of features. If each dimension is considered independently, there is nothing abnormal about a_1 or a_2. Each anomalous points share similar D_1 and D_2 values to the collection of points in N_2 and N_3. Only by considering both dimensions together, simultaneously, does the anomaly become obvious. For instance, a car driving on the highway in reverse is highly unusual, but on a driveway it is expected. Correct classification often requires the correct feature representation, such as the inclusion of time or location,

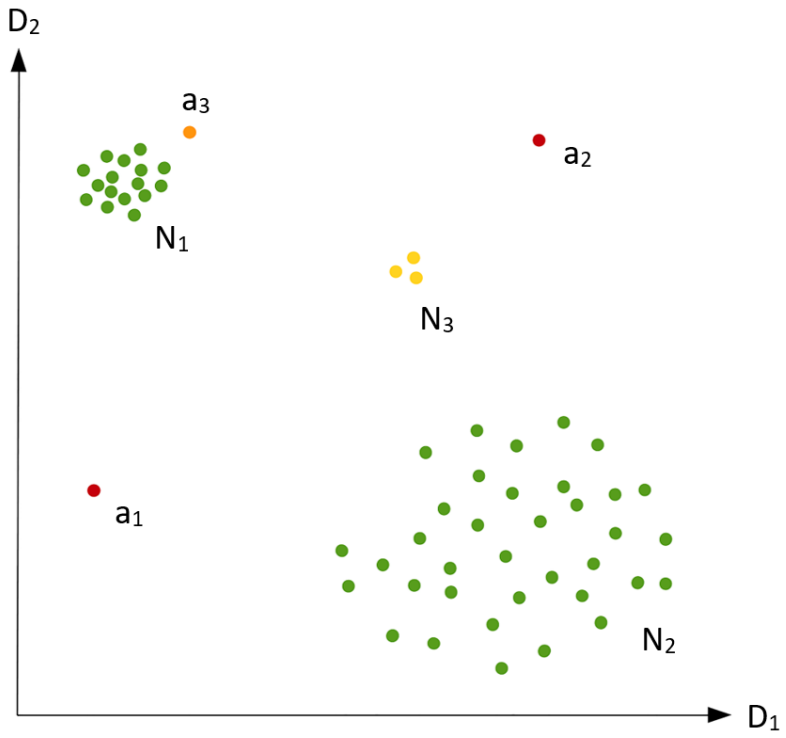


Figure 1.2: Different types of anomalies in an example 2-dimensional dataset. Adapted from [16] and [32].

to identify these types of *contextual* anomalies.

The subjectivity of these assignments further separates the task of anomaly detection from conventional classification tasks. A method that provides a meaningful, continuous anomaly score that describes the level of outlierness for each example is more desirable than a binary label output [32]. In Figure 1.2 for example, it is sensible to assign a_1 a higher anomaly score than a_3. The data may then be ranked according to the more flexible outlier score, and often a domain and method specific threshold (decision score) makes the final categorization if necessary.

Perhaps the greatest challenge of applying unsupervised techniques is that of parameter selection. In the unsupervised context there is no distinction between training and testing data as shown in Figure 1.1 [32]. This removes the normal practice of a validation step;

therefore, a conventional parameter search where labels are used *prior* to testing to determine the best parameters is not possible.

As a result, methods cannot be sensitive to parameter choices, or must implement some reliable means of making an appropriate selection. For example, traditional kernel-based methods are capable of capturing non-linear patterns to better identify local anomalies than simpler cluster-based approaches, but are highly sensitive to parameter settings [27]. In practice, domain specific heuristics are often applied, but may not generalize well. The problem of parameter selection is further amplified in deep approaches as the models become more complex. Though the parameters represented by the weights and biases of the network are trained by back-propagation, many hyperparameters must still be selected manually. Neural networks often require tuning the network architecture, learning rate, regularization, activation functions, and number of training steps for optimal results [13]. DAD methods specifically, are often prone to the *overgeneralization* to anomalous examples present in the training data [7, 33].

Despite the challenges, unsupervised anomaly detection is of critical practical importance and has been applied to areas of medicine, fraud detection, fault detection, cybersecurity, industrial inspection, surveillance and autonomous vehicle safety [13, 16, 32, 52, 95]. Specifically, our research focuses specifically on improving the performance of two anomaly detection algorithms, Kernel Principal Component Analysis (KPCA) and reconstruction-based autoencoders (AEs), in the unsupervised setting.

1.2 Objectives

The objective of this research is to improve upon the existing machine-learning-based anomaly detection methods of Kernel Principal Component Analysis (KPCA) and the Autoencoder

(AE) by better adapting them to the unsupervised setting.

1.3 Contributions and Scope

The specific novel contributions of our research are outlined below:

1. Previous methods, which have been applied, approximate the Gaussian kernel parameter, σ in the unsupervised setting, but are either domain specific [11, 62], or computationally inefficient on large datasets [27]. Few have been applied to Hoffman's formulation of KPCA for unsupervised anomaly detection [43]. Our research describes a modification of previous work, [27], to fit the framework of mini-batch stochastic gradient descent. In doing so, costly full constructions of the kernel matrix are avoided, allowing for an efficient means of selecting nearly optimal σ for KPCA when a search on labeled data is not possible.

2. Though KPCA has shown the ability to more tightly model background data than other methods such as the One-Class Support Vector Machine [43], KPCA's adoption as a practical tool has been limited by the cubic time complexity of the eigendecomposition step. Others have proposed approximation methods [67], but these suffer from reduced detection and increased variation as anomalies may disproportionately contaminate the sample. By combining an ensemble of very small *skeleton* models, the efficiency becomes effectively linear, while still maintaining or even increasing detection performance.

3. AEs are amongst the fundamental architectures for DAD, but AEs suffer from the problem of overgeneralizing anomalies when present in the training data. This unwanted property results in a diminished ability to separate anomalous and normal

examples based on reconstruction errors [7, 33]. Cumulative error scoring (CES) is introduced to leverage the training history of the AE to reduce the sensitivity to the stopping epoch.

4. Percentile loss (PL) is introduced to reduce the impact of anomalies on the parameter updates during training during training. PL ignores the most difficult examples during backpropagation to help prevent the AE from fitting anomalous examples present in the training data.

5. Early-stopping via knee detection, uses the smooth loss metric from CES to identify a stopping point for training. This again helps to prevent the AE from learning to reconstruct anomalous examples.

1.4 Outline

This thesis introduces the problem of unsupervised anomaly detection, briefly describes the current challenges, and highlights the contributions of this thesis in Chapter 1. Chapter 2 overviews a taxonomy of current traditional and deep approaches to unsupervised anomaly detection, while situating the contributions of our research in the larger framework. Chapter 3 details the baseline algorithms, KPCA and the AE, as well as the novel contributions that constitute the two new proposed method, Unsupervised Ensemble KPCA (UE-KPCA) and the Modified Training and Scoring AE (MTS-AE). The baseline methods, benchmark datasets, and metrics used in the evaluation of UE-KPCA and MTS-AE are described in Chapter 4. Next, Chapter 5 demonstrates the results across a number of ablation studies to identify the impact of the proposed modifications. Additionally, the proposed methods are compared to the popular baselines in broader evaluation. Finally, Chapter 6 concludes this thesis by summarizing the contributions and suggesting future areas of research.

Chapter 2

Review of Literature

This chapter reviews the most relevant research in the domain of unsupervised anomaly detection. Section 2.1 describes the different challenges in traditional machine learning and deep learning-based approaches and explains the need for progress in both. A brief review of traditional anomaly detection methods is provided in Section 2.2. Subsection 2.2.1 places Kernel Principal Component Analysis (KPCA) within the taxonomy of traditional methods as well as explaining its major limitations and noting recent attempts to address them. Section 2.3 highlights the major direction in deep anomaly detection (DAD) research, while Subsection 2.3.1 emphasizes the role of reconstruction-based approaches based on the autoencoder (AE).

2.1 Traditional Versus Deep Approaches

Progress in machine learning (ML) approaches to unsupervised anomaly detection has followed a different trajectory compared to other tasks such as classification or regression. In many areas of ML a subset of methods, known as deep learning (DL), have achieved state-of-the-art results that far exceed that of traditional approaches. The promise of DL methods lie in their ability to 1) scale to very large datasets 2) learn hierarchical features in an end-to-end fashion and 3) draw complex, non-linear boundaries between normal and anomalous data [95]. However, the deep learning methods suffer from two inherent disadvantages: 1) A

sensitivity to hyperparamter selection and 2) A reliance on large amounts of labeled data for learning [104]. Unsupervised anomaly detection methods do not have access to labels, therefore traditional means of training deep models are not applicable. As a result, traditional ML approaches remain the best performing in a number of domains [71, 89].

Yet, traditional ML approaches alone do not provide a comprehensive answer to the problem of unsupervised anomaly detection, especially in areas where the dimensionality of the data is high, such as images. A common problem, referred to as *Curse of Dimensionality*, indicates that traditional approaches which that rely on metrics of distance and density lose their ability to measure dissimilarity in higher dimensional spaces [3, 105]. Due to the issues of performance on raw, high dimensional, or noisy data, a conventional component of many traditional ML pipelines is *feature engineering* [103]. However, feature engineering requires domain expertise; moreover, where the absence of labels inhibits a validation step, the engineered features may not adequately differentiate anomalies from normal examples. Apart from dimensionality, high cardinality is also an issue for many traditional approaches that incur a quadratic or even cubic time complexity with the number of examples

Neither traditional ML nor DL methods fully address the challenges of unsupervised anomaly detection, which motivates continued work in both approaches. Our research makes contributions to both a traditional method, Kernel Principal Component Analysis (KPCA), and to a DAD method, the Autoencoder (AE). The following sections attempt to place these constituent methods within a larger framework. Figure 2.1 provides a taxonomy of unsupervised detection methods, showing the division and overlap of traditional and deep methods.

2.2 Traditional Anomaly Detection

Traditional ML methods of anomaly detection can be roughly categorized into the following:

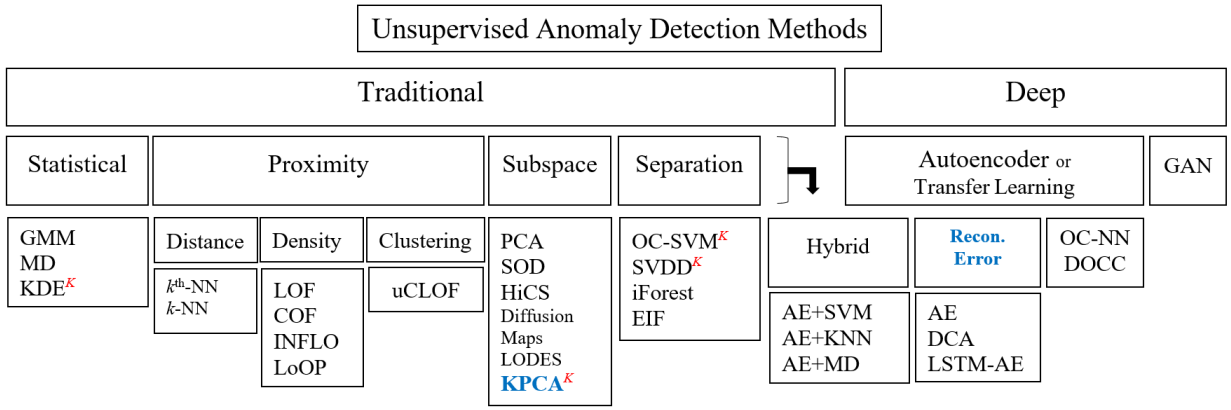


Figure 2.1: A taxonomy of unsupervised anomaly detection methods. A superscript K indicates a kernel-based method. The light blue indicates methods that are the focus of this thesis, KPCA and reconstruction-based AEs

- *Statistical approaches* assume that the data fits a known statistical model and identifies anomalies based on the distribution.

- *Proximity-based approaches* use measures of distance, density, or clustering assignment to identify anomalies.

- *Subspace-based approaches* assume that a reduced lower-dimensional representation or *manifold* better models the normal data, such that anomalies can be more easily identified in the appropriate subspaces.

- *Separation-based approaches* learn boundaries that separate normal and anomalous data using approaches similar to many traditional ML classification methods.

This categorization does not cover all existing anomaly detection methods. Moreover, methods often combine techniques in ways the blur the distinctions between categories. This review only attempts to provide a structure to the most commonly-used, traditional ML approaches for unsupervised anomaly detection [32, 95].

Before exploring specific methods, it is worth noting a meta-class of techniques, *Ensemble*

approaches assume that different methods trained on different subsets of the data better capture different types of anomalies. That is, a combination of the anomaly detection methods provide the most effective and robust solution [95]. This collaborative framework further motivates the broad investigation into new methods, as different approaches may strengthen a larger ensemble.

Statistical methods can be either parametric or non-parametric. The term parametric a misnomer in this context, as it does *not* refer to the presence or absence of specific parameters used in the algorithms. Instead, the major difference is that parametric methods assume an underlying distribution to the data and estimates the parameters of the model, whereas non-parametric requires no prior assumptions about the data. The Gaussian Mixture Model (GMM) is one of the most common parametric approaches.

The GMM attempts to fit several multi-variate Gaussian distributions to the data by using the global optimal expectation maximization algorithm [97]. GMMs have a high time complexity and demonstrate sensitivity to anomalies in the training data, limiting their use in the unsupervised setting. The Maholonobis distance (MD) simplifies the GMM by assuming the data can be described by a single multivariate Gaussian distribution [95]. The details of calculating the MD are provided in Section 4.1. Though the MD cannot capture non-linear relationships between features in the data, its fast run-time and simplicity make it a popular baseline for anomaly detection in many areas such as remote sensing [76]. The Histogram-Based Outlier Detection (HBOS) algorithm is a notable, non-parametric approach. HBOS uses static and dynamic bin width histograms to model the distribution of features [30]. HBOS is among the most computationally efficient methods, but the independent treatment of features prevents it from capturing correlations.

Distance-based anomaly detection methods are a class of direct, non-parametric approaches that identify anomalies by their greater distance from neighboring points. The most common

of these approaches, detailed in Section 4.1, uses the concept of a *nearest neighbor*. First, the pairwise Euclidean distances of all points are calculated. A parameter k, determines the number of points surrounding each example to be considered that example's neighbors. The anomaly score then becomes the distance to the furthest k^{th} nearest neighbor ($k^{th}NN$) [73] or the average distance to all k nearest neighbors (k-NN) [12]. A major limitation of these methods is the pair-wise comparison, which incurs a quadratic time and memory complexity with the number of examples.

Closely related to distance-based approaches, density-based approaches are driven by the assumption that anomalies are found in comparatively lower-density regions. Breuning *et al.* introduced arguably the first density-based approach, the Local Outlier Factor (LOF) method. LOF uses nearest-neighbor distances to compare the densities of local neighborhoods [10]. The details of LOF are found in Section 4.1. The normalization of the anomaly scores produced by measuring relative density by improves interpretability over nearest neighbor-based methods. LOF is one of the most cited baseline methods among anomaly detection algorithms [12, 32, 71].

There are several notable variations of the LOF. The Connective-base Outlier Factor (COF) uses a chaining distance rather than a Euclidean distance to calculate proximity. The chaining distance is the sum of the shortest path that connects all k neighbors to an example [92]. The COF significantly outperforms the LOF in instances where data exhibits a high linear correlation [32]. The Local Outlier Probabilities (LoOP) method attempts to increase the interpretability of the LOF by replacing the normalized density score with an probability of an anomaly by fitting half-Gaussian distributions to the nearest-neighbor distances [50]. Though the LoOP method attempts to address the problem of interpreting anomaly scores, it is unlikely to produce a better ranking for identifying anomalies [32]. The Influenced Outlierness (INFLO) algorithm uses an additional reverse nearest neighbor set to check if

a point is at the boundary of two nearby normal clusters of different densities, preventing erroneous high anomaly scores for those instances [44].

For the previously mentioned methods, the choice of the k parameter significantly impacts performance. The Local Correlation Integral (LOCI) method attempts to address the problem of parameter selection in the unsupervised setting [69]. LOCI defines a radius around each point that defines the points r-neighborhood. A maximization approach expands the r for each point to maximize its anomaly score. As with LoOP, LOCI assumes a half Gaussian distribution, but attempts to fit the aggregate number of examples, rather than the distances in a neighborhood, to measure density. Additionally, instead of using the local density's ratio, LOCI compares the radii of local neighborhoods. Because the maximization of r requires a search over all pairs for each example, the training incurs a cubic time complexity. The authors of LOCI offer an approximate version, aLOCI, which provides some reduction in complexity by using quad trees [69]. However, empirical performance of aLOCI shows a sensitivity to parameter settings [32].

Distance and density methods rely on pairwise comparisons, but do not attempt to explicitly identify patterns in data. Clustering-based approaches differ by directly identifying groups of similar data by means of clustering. The k-means clustering algorithm is the most common approach for membership assignment [61]. The algorithm begins by randomly locating a number of cluster centroids, then assigns each point membership to a cluster. The centroids are then recomputed, and the assignment step repeated. After a number of iterations, the centroids and membership assignments converge to a final state. Though typically more computationally efficient, clustering approaches are sensitive to noise, the specification of the initial number of clusters, and the initialization of cluster centroids; all of which can lead to poor performance [61]. Again, Section 4.1 details one of the generally best performing cluster-based methods, the unweighted Cluster-Based Local Outlier Factor (uCBLOF) [40].

The methods discussed so far attempt to identify anomalies by modeling the data in its full (ambient) dimensionality. However, anomalies often only clearly exhibit abnormal characteristics in a smaller subset of linear or nonlinear combination of one or more lower-dimensional subspaces [98, 105]. The idea that anomalies are more pronounced in these subspaces (embeddings, projections) by the removal and/or transformation of features characterizes subspace-based approaches [16].

Principal component analysis (PCA), detailed in section 4.1, is the most common dimensionality reduction technique and can also be applied to anomaly detection. PCA projects data onto orthogonal axes, or principal components (PC), representing the greatest variation. Anomalies tend to lie farther from these principal components than normal data. However, PCA assumes features are linearly correlated and is sensitive to both the presence of outliers during training and the choice of the retained principal components. Robust and local versions of PCA have been proposed, but require additional parameter selection [49, 51].

Other algorithms attempt a more explicit search of a subspaces for anomaly detection. The High Contrast Supspace (HiCS) assumes that rare patterns are statistically more common in subspaces that display less uniformity. Combinations of subspaces are sampled and then pruned based on statistical testing. Finally, the LOF is calculated across each set and the anomaly scores are aggregated [46]. Subspace outlier detection (SOD) uses a distance-based criterion for selecting local subspaces rather than a fixed set over the entirety of the data. For each data point, a collection of k nearest neighbors forms a reference set. The subspace is determined as the set of dimensions that minimizes the variance. The Euclidean distance to the neighborhood's mean, normalized by the new dimensionality, serves as the anomaly score [48].

The methods discussed so far use only a subset or linear transformation of the original features. However, high dimensional data is often distributed along non-linear lower dimen-

sional *manifolds* of arbitrary shape [16, 34]. Spectral methods are special class of subspace methods that are focused on modeling these patterns. Local Density Meets Spectral Outlier Detection (LODES), uses PC from a connective graph of nearest neighbors [84]. Similarly, A Diffusion Map decomposes a transition matrix from a random walk in a transformed feature space [19]. Non-linear methods produce better models of the data, but often at a greater computational cost and can be susceptible to problems of over-fitting and parameter selection [16].

Many traditional classification techniques serve as the basis for popular separation-based anomaly detection approaches. Influenced by the random forests used for classification, the Isolation Forest (iForest) algorithm, detailed in Section 4.1, uses an ensemble of isolation trees to separate anomalies from normal data[56]. The method is arguably another form of subspace-learning as the branches in the trees are built from randomly selected features. Extended Isolation Forest (EIF), improves upon the original algorithm by allowing the branching hyperplanes to take on any slope as opposed to standard iForest which allows only orthogonal hyperplanes [38].

The One-class Support Vector Machine (OC-SVM) extends the popular support vector machine (SVM) algorithm for classification to anomaly detection. OC-SVMs attempt to explicitly separate the normal (single positive) class from anomalies by constructing a maximum-margin hyperplane through the data [86]. Obviously, a linear hyperplane in ambient space does little to isolate the normal class. Instead, the OC-SVM method is one of many *kernel* methods that rely on an initial non-linear transformation of the data.

Given N data points, $\mathbf{x}_i \in \mathbb{R}^{D\times N}$, comprising the dataset, $\mathbf{X} \in \mathbb{R}^{D\times N}$, kernel methods are motivated by the idea that a better model of the data may be formed in a feature space, $\mathcal{F}$, where a kernel function, $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$, allows the efficient computation of inner products between each points. $\boldsymbol{\Phi}(\boldsymbol{x}_i)$ in $\mathcal{F}$ without the need to explicitly calculate the mapping $\mathbf{x}_i \rightarrow \boldsymbol{\Phi}(\boldsymbol{x}_i)$

from the ambient space to the feature space. The choice of kernel function allows for different mappings [54].

Kernel-based methods are a meta-class of methods where familiar proximity, subspace, and separation algorithms can be applied the feature space. Other popular kernel-based anomaly detection methods include the Support Vector Data Description (SVDD) and the Kernel Density Estimator (KDE). The SVDD is similar to the OC-SVM, but imagines the boundary as a hypersphere enclosing the data, rather than a hyperplane [93]. The KDE, or Parzen–Rosenblatt window method, is a non-parametric, statistical method based properties of the Gaussian kernel [79]. Section 4.1 details OC-SVMs and the KDE and explains their close relationship to the OC-SVM.

2.2.1 Kernel Principal Component Analysis

Kernel Principal Component Analysis (KPCA) is a spectral technique that extends linear PCA to the non-linear feature space using kernel methods [87]. Hoffmann [43] first demonstrated that the independent treatment of points by the OC-SVM yields boundaries that do not tightly enough model the data. In turn, Hoffman argues that KPCA discovers a better model of the data by discovering the underlying manifold expressed by the principal components in $\mathcal{F}$. Next, an anomaly score can be defined as the reconstruction error between a given example and a subset of the learned principal components, creating a tighter decision boundary, as shown in Figure 2.2. The steps in KPCA are described in greater detail in Section 3.1.1. With the correct parameter settings, Hoffman showed that KPCA demonstrated better generalization, accuracy, and robustness over linear PCA, KDE, and OC-SVMs on a number of real-world and toy datasets.

Despite a demonstrated success in anomaly detection, KPCA has several limitations. KPCA

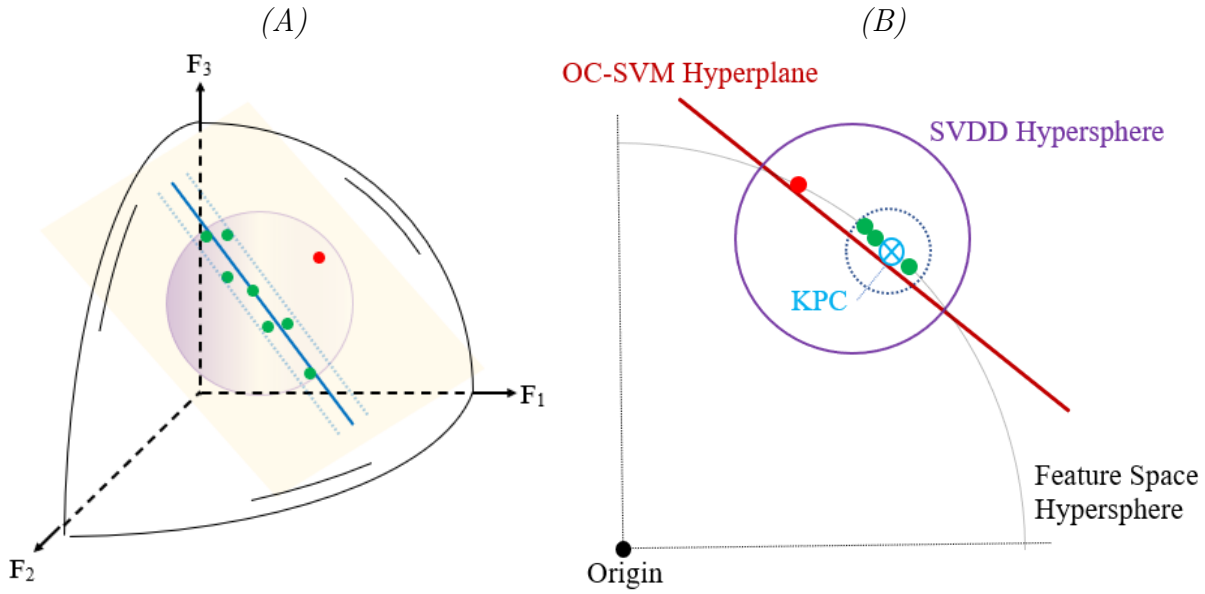


Figure 2.2: The decision boundaries of KPCA, OC-SVM, and SVDD in the feature space produced by a Gaussian Kernel. (A) on the left shows the boundaries in a three-dimensional representation. Both normal examples (green) and the single anomaly (red) lie on the surface of a hypersphere. The (blue) line represents a kernel principal component (KPC), where a small boundary captures all the normal examples. Both the SVDD hypersphere and OC-SVM must include the anomaly in order to enclose all the normal points. (B) on the right shows a cross section orthogonal to the principal component, representing the same situation in (A). Adapted from [43].

requires the eigenvalue decomposition of a $N \times N$ kernel (Gram) matrix. This incurs a cubic time complexity with increasing N which can be limiting for many applications. Techniques to discover reduced approximations of the kernel matrix have been developed, largely in response to the computational costs of the more popular kernel-based support vector machines.

Uniform subsampling of the data in order to reduce the cost of calculating the adjacency matrix has been proposed for both classification [6] and anomaly detection [66]. However, a *bad* sampling can be contaminated with a greater number of anomalies. An alternative approach to subsampling is the Nyström algorithm; an out-of-sample extension method associated with dimensionality reduction. However, the method still requires a quadratic

time full construction of the kernel matrix [8, 53].

Other methods attempt to batch the data to iteratively update principal components without full evaluations. The kernel Hebbian algorithm (KHA) adapts the generalized Hebbian algorithm commonly used in iterative linear PCA to the feature space [36, 47]. However, because updates are made on previous solutions, these iterative methods cannot be computed in parallel and may be slow to converge.

In addition to the challenges of computational efficiency, all kernel-based methods are strongly sensitive to parameter choice. Aware of this shortcoming, Hoffman presented KPCA in the semi-supervised setting with a a hold-out set of anomalies for parameter tuning. In the unsupervised case heuristics based on statistics of the adjacency matrix are often used for selecting parameters for kernel method [11], but do not always produce satisfactory results.

Ultimately, in the unsupervised setting there is no guaranteed means for verifying that a specific parameter selection is an optimal or even reasonable choice. However, theoretically motivated formulations to the problem of kernel parameter selection have shown promise in OC-SVMs. These methods are motivated by the idea of minimizing the sparsity in the high dimensional feature space, so that anomalies are more easily differentiated from normal points [11]. Evangelista *et al.* demonstrated that maximizing the index of dispersion of kernel entries produces generally good choices for the σ parameter of a Gaussian-kernel OC-SVM [27]. However, the optimization incurred a large time complexity via multiple evaluations of the full kernel matrix.

Our research makes contributions that improve KPCA to perform more reliably in the unsupervised setting and to run more efficiently on larger datasets. Section 3.1 reviews the details of the base KPCA method, introduces a mini-batch stochastic gradient descent method to efficiently determine the kernel parameter, and presents an ensemble method that avoids

avoids the construction and decomposition of the full kernel matrix.

2.3 Deep Anomaly Detection

The *deep* in Deep Learning and Deep Anomaly Detection (DAD) refers to the use of multi-layer artificial neural networks (NN) which are inspired by the structure of biological brains. Each computational unit (node or neuron) of a typical NN receives inputs, performs an element-wise multiplication by a set of individual weights, sums the results, adds a bias, then finally applies a non-linear activation function. The basic feed-forward network (multi-layer perceptron) organizes the computational units into layers, and then arranges the layers sequentially, so that the output of one layer serves as the input to the next, forming a Deep Neural Network (DNN) [34].

Training NNs first requires the formulation of a loss (cost) function, which measures the difference between the network's current output and some desired, target output. The gradient of loss is then *back-propagated* through the parameters (weights and biases) to reduce this difference. By iterative applications of this process, the network learns to better match inputs to target outputs. In addition to the parameters that constitute the network, training also involves a set of hyperparameters, such as learning rates, mini-batch sizes, training steps, weight decay, ect., that need to be specified by the user [34].

DNNs are able to learn hierarchical discriminative features and highly non-linear boundaries in an end-to-end manner. The state-of-the art performance of DNN classifiers has motivated the exploration of Deep Anomaly Detection (DAD) methods. However, because of the challenges of hyperparamter selection, choosing architectures, and over or under-fitting most DAD methods have been developed in the semi-supervised framework. Nevertheless, with the correct settings DAD methods have shown improvement over traditional ML approaches,

particularly on higher dimensional data [13, 60, 71].

Semi-supervised DAD methods often employ autoencoders (AE) as well as other generative models such as variational autoencoders (VAE) and generative adversarial networks (GANs) to model normal data [21, 60]. Similar to subspace-methods, *hybrid* approaches apply more traditional detection methods to the embedding spaces of AEs. VAEs can be used to enforce statistical properties in the embedding space. Some examples of this paradigm include the AE+OC-SVM [5], the AE+kNN [35], and AE+MD [24]. Alternatively, *transfer learning* uses the final hidden layers of pre-trained classifiers for feature extraction. Transfer learning usually produces better embeddings, but requires a large corpus of labeled data in a similar domain to train the original classifier [63]. Lastly, deep one-class classification approaches, such as the One-class Neural Networks (OC-NN) [15] and Deep One-Class Classifiers (DOCC) [80], involve modifying the training objective in deep architectures to extract features that differentiate anomalies and create a decision boundary [13].

Methods designed for semi-supervised DAD can often be applied in an unsupervised setting as well. Yet difficulties in generalizing hyperparameter selection and sensitivity to anomalies in the training data often significantly degrade performance [106]. The simplicity and flexibility of the basic AE makes it a suitable starting point for addressing the challenges of unsupervised DAD.

2.3.1 Autoencoders

An AE attempts to learn the identity function with some constraint, to prevent a trivial mapping. Typically this constraint is in the form of contractive, *bottleneck* hidden layer, where the original dimensionality is reduced. This reduced representation of the original input, serves as end-to-end feature extraction step for the hybrid methods described in the

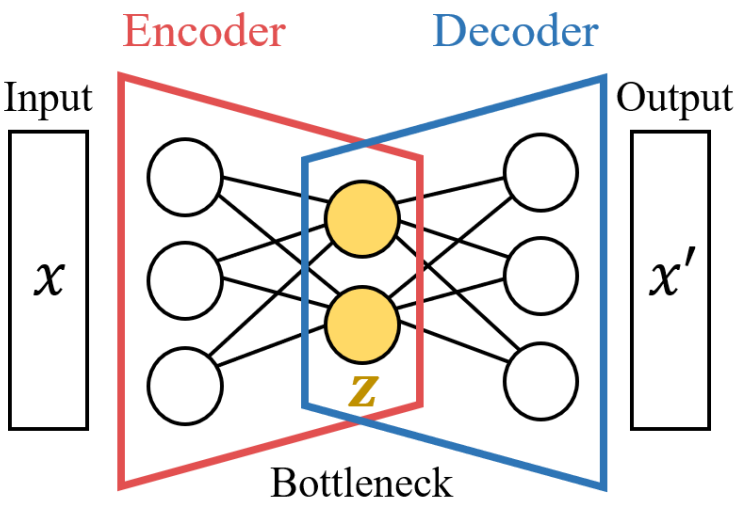


Figure 2.3: Architecture of a simple, contractive Autoencoder with one hidden layer.

previous section. Alternative to or in combination with a contractive layer, other constraints such as sparsity, regularization, or added noise can be enforced. Because the target output is simply the original data, AEs can be trained without labels [34]. Figure 2.3 illustrates the architecture of a basic, single-hidden-layer, contractive AE.

Reconstruction-based unsupervised DAD methods using AEs measure the magnitude of the reconstruction error (residual vector) to identify the anomalies. Anomaly detection in this approach relies on the assumption that AEs will learn to reconstruct normal examples, which are more prevalent in the training data, better than anomalous examples. Details of the AE are outlined in Section 3.2.

AEs have been used to model and detect anomalies in high dimensional multivariate point [64], image [20], temporal [81], and spatiotemporal data [100]. AE architectures have incorporated both 2D and 3D convolutional layers and recurrent modules such as RNNs, GRUs, and LSTMs [22, 39, 77]. The recent survey by Chalapathy *et. al* provides a more exhaustive list [13]. This paper focuses on the basic, fully-connected (dense) architecture, but the methods can be easily extended to other AE architectures or reconstruction-based, DAD

methods.

AEs used for anomaly detection suffer from the fact that the reconstruction error serves as *both* an objective function for training and an anomaly scoring metric. The fundamental problem is that in the unsupervised setting, the training acts to directly minimize the anomaly score of anomalous examples in the training data. Therefore, simply enforcing sparsity or other forms of regularization alone does not robustly prevent the problem of AEs overgeneralizing anomalies.

Though recent works have trained AEs for anomaly detection tasks, few have remarked on their significant limitations in the unsupervised setting. Beggel *et al.* proposed a hybrid approach that iteratively refined the training set by using a one-class SVM in the latent space of an adversarial autoencoder (AAE) to remove suspected anomalous examples [7]. Other boosting techniques and AE cascades have likewise shown resistance to generalization [83]. Gong *et al.* used a memory-augmented AE to memorize normality and limit the latent representations in order to prevent the model from learning anomalous examples [33]. Robust Convolutional Autoencoders (RCAEs) and related methods extend robust PCA to DAD by learning a an embedding space via an AE that captures most of the normal features, while providing a margin to account for anomalies during training [14]; however, this method involves selecting a noise absorption term.

The contributions of our research are distinct in the fact that they focus on the reconstruction in the ambient space and do not require any modifications to network architecture. Furthermore, the proposed techniques outlined in Section 3.2 can potentially be combined with other existing, reconstruction-based techniques, including those discussed, to further improve robustness and performance.

Chapter 3

Proposed Methods

3.1 Unsupervised Ensemble Kernel Principal Component Analysis

The goal of this section is to first detail Hoffman's original formulation of kernel principal component analysis (KPCA) for anomaly detection and then to demonstrate how issues of parameter selection and computational inefficiency can be effectively addressed.

3.1.1 The Kernel PCA Algorithm

The KPCA algorithm is designed to first calculate the non-linear mapping $\boldsymbol{x}_i \rightarrow \boldsymbol{\Phi}(\boldsymbol{x}_i)$ of a datum in $\boldsymbol{x} \in \boldsymbol{X} \subset \mathbb{R}^{D \times N}$ from the original D-dimensional (ambient) space into the potentially infinite-dimensional (for a Gaussian kernel) feature space $\mathcal{F}$. After the initial non-linear mapping, the data are centered in $\mathcal{F}$ via the transformation

$$\tilde{\boldsymbol{\Phi}}(\boldsymbol{x}_i) = \boldsymbol{\Phi}(\boldsymbol{x}_i) - \boldsymbol{\Phi}_0, \tag{3.1}$$

where

$$\boldsymbol{\Phi}_0 = \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{\Phi}(\boldsymbol{x_n}). \tag{3.2}$$

is the mean of the data distribution in $\mathcal{F}$. Following the transformation, the next step is to perform linear principal component analysis on the centered data to find the M-dimensional subspace $M \leq N$ associated with the M principal components representing the greatest variance of the data in $\mathcal{F}$.

The principal components of $\mathbf{X}$ in $\mathcal{F}$ are the eigenvectors corresponding to the largest eigenvalues of the covariance matrix formed in $\mathcal{F}$. The hope is that these eigenvectors will represent a non-linear model that describes underlying structure of the data. More directly, the objective is to find the eigenvectors $\mathbf{V} = [\boldsymbol{V}^1, \boldsymbol{V}^2, \ldots, \boldsymbol{V}^M]$ and corresponding eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_M$ of

$$\tilde{\boldsymbol{\Sigma}}_{\mathcal{F}} = \frac{1}{N} \sum_{i=1}^{N} \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}_i) \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}_i)^T. \tag{3.3}$$

Recalling the relationship to the eigenvectors and eigenvalues being

$$\tilde{\boldsymbol{\Sigma}}_{\mathcal{F}} \boldsymbol{V}^k = a_k \boldsymbol{V}^k. \tag{3.4}$$

The main challenge is that covariance matrix of the feature-space, centered data, $\tilde{\boldsymbol{\Sigma}}_{\mathcal{F}}$, and therefore the principal components, $\boldsymbol{V}$, cannot be explicitly computed, as the transformed data $\boldsymbol{\Phi}(\boldsymbol{x}_i)$ is never available. However, the so called *kernel trick* allows a kernel function, κ, to replace the inner product operations in $\mathcal{F}$, where

$$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{\Phi}(\boldsymbol{x}_i) \cdot \boldsymbol{\Phi}(\boldsymbol{x}_j). \tag{3.5}$$

For the choice of a Gaussian kernel,

$$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = exp(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}). \tag{3.6}$$

Rather than explicitly calculating the transformation in order to determine $\boldsymbol{V}$, the projections of data $\boldsymbol{\Phi}(\boldsymbol{x}_i)$ onto $\boldsymbol{V}$ are found instead. Because $\boldsymbol{V^k}$ is one eigenvector of $\tilde{\boldsymbol{\Sigma}}_{\mathcal{F}}$, by definition a kernel principal component can be expressed as a linear combination of points $\boldsymbol{\Phi}(\boldsymbol{x}_i)$,

$$\boldsymbol{V^k} = \sum_{i=1}^{N} \boldsymbol{\alpha}_i^k \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}_i), \tag{3.7}$$

where each element α_i^k is a component of a vector $\boldsymbol{\alpha}^k$, which is an eigenvector of the $N \times N$ kernel adjacency matrix $\tilde{K}_{ij} = \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}_i) \cdot \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}_j)$. Using the kernel trick, this matrix may in turn be expressed solely as a function of ambient data,

$$\tilde{K}_{ij} = K_{ij} - \frac{1}{N}\sum_{q=1}^{N} K_{iq} - \frac{1}{N}\sum_{p=1}^{N} K_{pj} + \frac{1}{N^2}\sum_{p,q=1}^{N} K_{pq}, \tag{3.8}$$

where $K_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The eigenvectors $\boldsymbol{\alpha}^k$ and corresponding eigenvalues v_k are then found by the eigendecomposition of $\tilde{K}_{ij}$ ultimately yielding N eigenvectors $\boldsymbol{\alpha} = [\boldsymbol{\alpha}^1, \boldsymbol{\alpha}^2, \ldots, \boldsymbol{\alpha}^N]$. A scaling of each $\boldsymbol{\alpha}^k$ is performed so that each $\boldsymbol{V^k}$ has unit length, $||\boldsymbol{\alpha}^k||^2 = 1/v_k$.

3.1.2 Anomaly Scoring

The anomaly score for a point $\boldsymbol{x}$ is found by determining the *reconstruction error* in $\mathcal{F}$ from some number of leading principal components that represent a normal model of the data [43]. The reconstruction error is computed by

$$d_E(\boldsymbol{x}) = \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}) \cdot \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}) - \boldsymbol{W}\tilde{\boldsymbol{\Phi}}(\boldsymbol{x}) \cdot \boldsymbol{W}\tilde{\boldsymbol{\Phi}}(\boldsymbol{x}), \tag{3.9}$$

where $\boldsymbol{W}$ contains M rows of principal components $\boldsymbol{V^k}$ corresponding to the M largest eigenvalues, $\boldsymbol{W} = \boldsymbol{V}_M^T = [\boldsymbol{V}^1, \ldots, \boldsymbol{V}^M]^T$.

The first term in (3.9) represents the spherical potential of $\boldsymbol{x}$ found by taking the scalar product

$$d_p(\boldsymbol{x}) = \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}) \cdot \tilde{\boldsymbol{\Phi}}(\boldsymbol{x}), \tag{3.10}$$

which is simply the squared distance of $\tilde{\boldsymbol{\Phi}}(\boldsymbol{x})$ from the data mean $\boldsymbol{\Phi}_0$ in $\mathcal{F}$. The second term in (3.9) represents the projection of the data onto a reduced number of M principal components.

Again, it is not possible to to obtain $\tilde{\boldsymbol{\Phi}}(\boldsymbol{x})$. Instead, the kernel trick (3.5) is used once again by substituting after substituting (3.2) into (3.10), resulting in

$$d_p(\boldsymbol{x}) = \kappa(\boldsymbol{x}, \boldsymbol{x}) - \frac{2}{N} \sum_{i=1}^{N} \kappa(\boldsymbol{x}, \boldsymbol{x}_i) + \frac{1}{N^2} \sum_{i,j=1}^{N} \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j). \tag{3.11}$$

Next, $f_k(\boldsymbol{x})$ is defined as the projection of $\boldsymbol{x}$ in $\mathcal{F}$ onto one of the principal components, as $f_k(\boldsymbol{x}) = \boldsymbol{V}^{k^T} \cdot \tilde{\boldsymbol{\Phi}}(\boldsymbol{x})$. This projection can be written as a function of only the ambient data by applying (3.7) and the kernel trick (3.5),

$$f_k(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i^k [\kappa(\boldsymbol{x}, \boldsymbol{x}_i) - \frac{1}{N} \sum_{q=1}^{N} \kappa(\boldsymbol{x}_i, \boldsymbol{x}_q) - \frac{1}{N} \sum_{q=1}^{N} \kappa(\boldsymbol{x}, \boldsymbol{x}_q) + \frac{1}{N^2} \sum_{p,q=1}^{N} \kappa(\boldsymbol{x}_p, \boldsymbol{x}_q)]. \tag{3.12}$$

Finally, the reconstruction error-based anomaly score can be directly computed by,

$$d_E(\boldsymbol{x}) = d_p(\boldsymbol{x}) - \sum_{k=1}^{M} f_k(\boldsymbol{x})^2, \tag{3.13}$$

which is the reconstruction error between $\tilde{\boldsymbol{\Phi}}(\boldsymbol{x})$, the centered projection of $\boldsymbol{x}$ into $\mathcal{F}$, and its representation in $\mathcal{F}$ as a projection onto the largest M principal components of the PCA

model learned from the data.

If $M = N$ then $d_E(\boldsymbol{x})$ will be zero for all $\boldsymbol{x}$ because the representation of $\tilde{\boldsymbol{\Phi}}(\boldsymbol{x})$ in PCA coordinates is identically $\tilde{\boldsymbol{\Phi}}(\boldsymbol{x})$. When $M < N$ then $d_E(\boldsymbol{x})$ will remain smaller for non-anomalous points because the learned PCA model better represents the background and the error associated with dropping low-eigenvalue eigenvectors will remain smaller as M decreases.

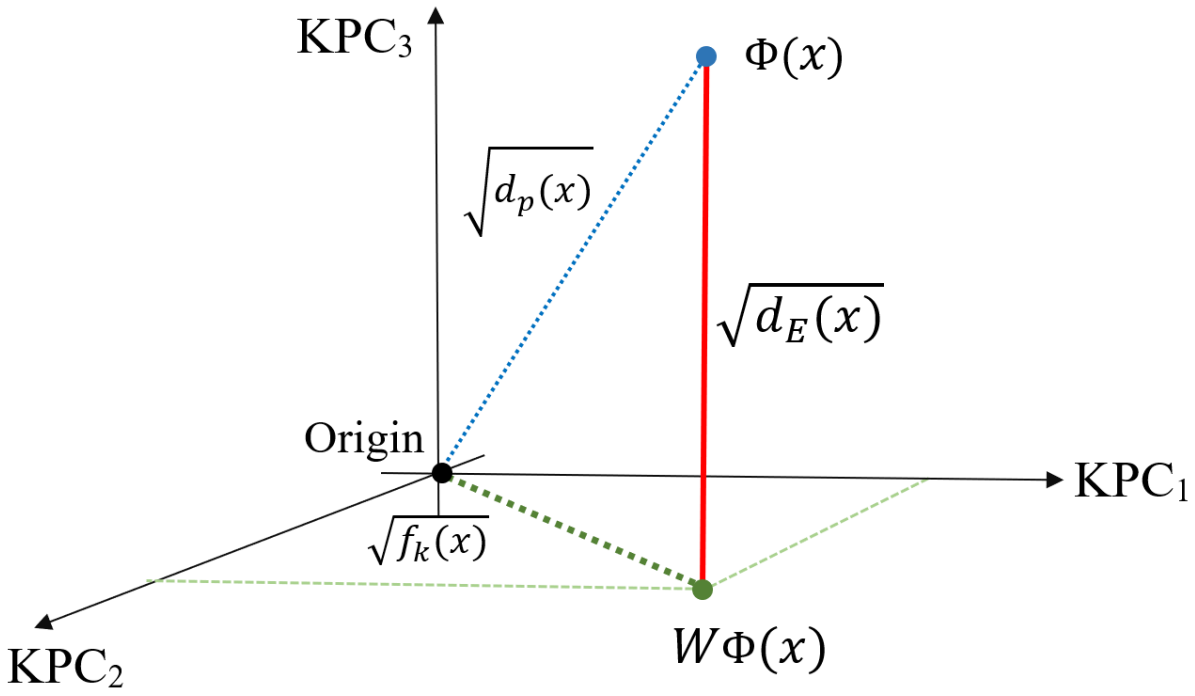


Figure 3.1: The anomaly scores is the squared distance represented by the red line indicating the separation between the point in $\mathcal{F}$ (blue) and its projection onto a subset of kernel principal components (green).

Figure 3.1 gives a visualization of the anomaly score. In this simplified version there are only three principal components shown (black). A point $\boldsymbol{x}$ is transformed into $\mathcal{F}$, $\tilde{\boldsymbol{\Phi}}(\boldsymbol{x})$. The reconstruction error in $\mathcal{F}$, which serves as the anomaly score, is then the squared distance between the point in centered feature space and its projection to a subset of the principal components. The reasoning is that a large $\tilde{\boldsymbol{\Phi}}(\boldsymbol{x})$ alone may not be indicative of an anomaly

if it lies close to a principal axis describing a larger pattern in the data. However, a large reconstruction error should be indicative of an anomaly as it is a measure of how far a point lies from the overall model of the data as represented by the principal components.

3.1.3 Challenges

In practice, it is difficult to implement KPCA for unsupervised anomaly detection for two main reasons: 1) The sensitivity to the parameter settings of the Gaussian kernel 2) the cubic time complexity of the eigendecomposition of the kernel matrix, $\tilde{K}$ (3.8). This section details these issues, while 3.1.4 and 3.1.5 describe the proposed solutions.

As with all kernel based methods, the ability for kPCA to detect anomalies is directly tied to parameter selection. For a Gaussian kernel, σ is the critical parameter choice [96]. It is worth first exploring the limits of the parameter choice and general properties of the kernel matrix. A Gaussian kernel matrix will always have a diagonal containing all ones, as the diagonal represents the self-distance term, e.i. $||\boldsymbol{x}_i - \boldsymbol{x}_i|| = 0$.

As σ approaches an arbitrarily large value, the argument of the kernel for any value of $\boldsymbol{x}$‘ approaches 0 as the argument of the exponent in (3.5) approaches negative infinity. Explicitly,

$$\lim_{\sigma \to 0} \kappa(\boldsymbol{x}, \boldsymbol{x}') = 0 \tag{3.14}$$

In this case $\tilde{K}$ approaches the identity matrix. This indicates that all data vectors in feature space become orthogonal to one another and the principal components become meaningless. Alternatively, as the width of σ increases the off diagonal tend toward 1,

$$\lim_{\sigma \to \text{inf}} \kappa(\boldsymbol{x}, \boldsymbol{x}') = 1 \tag{3.15}$$

Evidently σ values that are too small lead to an over separation of points in $\mathcal{F}$, a form of over-fitting. For σ values that are too large, all points begin to be mapped to similar locations in $\mathcal{F}$, a type of under-fitting [88]. Otherwise stated, in the former case all points appear to be anomalous $\mathcal{F}$, in the latter all points appear normal. In the unsupervised setting it is not possible to perform a parameter search because there is no conventional sense of a hold out validation set without the availability of labels. Section 3.1.4 describes a proposed solution to determining a nearly optimal kernel choice in the unsupervised setting.

Computational efficiency, both in terms of time and space complexity, is an issue at several steps in the conventional deployment of KPCA, and a major limiting factor in its applicability. Despite the demonstrated ability to fit non-linear patterns in data, kernel methods require the calculation of a distance (adjacency) matrix comprised of all pairwise similarity measures between each of the N data point in $\boldsymbol{X}$. Specifically the calculation of the adjacency matrix needed to form K, has $O(DN^2)$ time complexity and $O(N^2)$ space complexity. Of even greater concern is the $O(N^3)$ time complexity to decompose $\tilde{K}$. This is prohibitively expensive for large datasets. Section 3.1.5 outlines a process to greatly reduce the computational costs, without sacrificing detection accuracy.

3.1.4 Learning the Kernel

Parameter selection in the supervised setting is often challenging for KPCA, as the complexity makes an extensive grid search of parameters very expensive. However, for unsupervised tasks, a conventional search of any kind is not possible. Instead, a heuristic based on the nearest neighbor distance, or some other distance adjacency metric is often used to select σ [11, 37, 67]. These heuristics are usually sub-optimal and tied to the dispersion of the data. Other methods such as [27, 94] iterative full evaluations of the kernel matrix or estimations

of the error rate.

The proposed method extends [27] to KPCA and significantly reduces the time and space complexity. Evangelista and Embrechts employ a powerful, general heuristic for selecting a near optimal value of σ without the need for labeled data. The method is based on maximizing the coefficient of variance of the off diagonal entries in the kernel matrix.

In the full $N \times N$ kernel matrix there are $N^2 - N$ off diagonal entries. Because of symmetry, half are duplicates, so there are only l *unique* off diagonal entries, $l = N^2 - N$. Evangelista suggests the following *fundamental premise of pattern recognition*, that suggests a good model should follow,

$$\kappa(i,j)|(y_i = y_j) > \kappa(i,j)|(y_i \neq y_j), \tag{3.16}$$

Which simply indicates that points that are closer in the ambient space will produce larger kernel values than distant points. For the Gaussian kernel, this is a consequence of

$$\lim_{\|\boldsymbol{x},\boldsymbol{x}'\|\to 0} \kappa(\boldsymbol{x},\boldsymbol{x}') = 1. \tag{3.17}$$

For anomaly detection, most pair-wise comparisons are of normal to normal data, *i.e.* $y_i = y_j$.

On first approach it is natural to assume that simply tuning the matrix to take on high values will preserve the idea of adjacency in F. This is misguided, this leads to under-fitting where anomalies are not pronounced. Instead, the important metric is the dispersion of the data. Decomposing the disperse kernel matrix, results eigenvectors that are most representative of neighboring points, as they have proportionally higher values, while minimizing the impact of distant anomalies. This results in a good, non-linear model of the data.

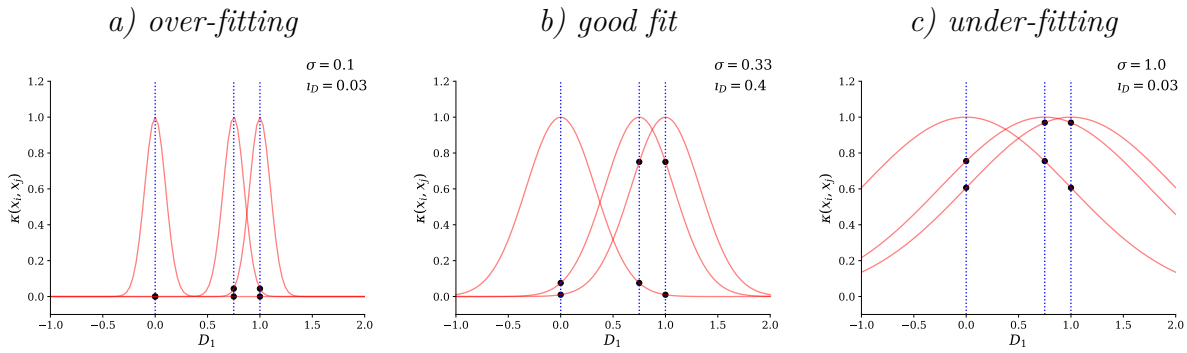


Figure 3.2: Gaussian kernels are corresponding to different choices of σ are fitted to one-dimensional data. The black dots indicate the corresponding off-diagonal kernel entries. The anomaly (at 0) is best separated when the index of dispersion is maximized.

The index of dispersion, i_D, (coefficient of dispersion, relative variance, or variance-to-mean ratio) provides a normalize measure of the dispersion of a distribution of values,

$$i_D = \frac{s^2}{m} \tag{3.18}$$

where s^2 is the variance and m is the mean. By applying this measure, it is possible to quantify the sparsity of the off-diagonal kernel entries, l. Figure 3.2 demonstrates the problem of over-fitting and under-fitting for a simple, one-dimensional example with three points. Furthermore, i_D of the l kernel entries exhibits a global maximum, making it an ideal target of an objective function to determine σ [27]

.

The main contribution of our research is to modify the objective to avoid the $O(iN^2)$ computational complexity associate with i iterative evaluations of the full kernel matrix. Instead of deploying the simple hill climbing optimization used in [27], the objective is cast as a loss function to fit the framework of mini-batch stochastic gradient descent. Equation 3.18 is inverted such that the objective becomes,

$$\mathcal{L}(\boldsymbol{X}_b, \sigma) = \frac{m_b}{{s_b}^2 + \epsilon}, \tag{3.19}$$

where s_b^2 is the variance of the off diagonal entries of K_b and m_b is the mean. K_b is formed by drawing a batch,, $\boldsymbol{X}_b$, of examples from $\boldsymbol{X}$, and then forming the adjacency matrix and applying the kernel.

The proposed sampling method is beneficial because the full kernel matrix never needs to be computed or stored in memory. Forming K_b in this way is equivalent to randomly drawing a set of rows from K, applying the same indexing to the columns, and saving the entries of intersecting rows and columns. This process relies on the assumption that the index of dispersion of the samples, i_{D_b}, approximates the i_D of all l entries, so that using iterative draws of i_{D_b} as a metric for tuning yields the same, near optimal, result for σ.

Specifically, n_b is the batch size and $l_b = {n_b}^2 - n_b$, so that

$$m_b = \frac{\sum_{i=1}^{n_b} \sum_{j=1+1}^{n_b} \kappa(i,j)}{l_b}, \tag{3.20}$$

and

$${s_b}^2 = \frac{\sum_{i=1}^{n_b} \sum_{j=1+1}^{n_b} (\kappa(i,j) - m_b)^2}{l_b}. \tag{3.21}$$

To prevent negative values for σ, optimization is instead performed on a bias, b, which is passed through an activation function,

$$\sigma = \log(1 + \exp(b)). \tag{3.22}$$

An initial b_0 is set to correspond to $\sigma_0 = 1$. Early stopping is performed by tracking the lowest value of the loss. The number of training steps (batches) since the record lowest loss is tracked, and if the number exceeds a set patience, p, training is halted, and the average σ over that period is returned. The Algorithm 1 outlines the steps for tuning σ.

The proposed extensions of [27] to the gradient descent framework allows for unsupervised, near-optimal kernel tuning on very large datasets efficiently. The space complexity is reduced to $O({N_b}^2)$ space and to $O(i_b {N_b}^2)$ time complexity, where i_b is the number of batches drawn before convergences is declared. Results in Section 5.2 show that batches as small as $N_b = 100$ are generally sufficient and convergence that typically occurs in under 2000 steps.

Algorithm 1: Gradient Descent σ optimization

input : $\boldsymbol{X}$ -globally min-max normalized data
Given: n_b -batch sampling size, p -patience, σ_0 -initial σ
initialize b_0;
initialize $\mathcal{L}_{\min}$;
initialize $t = 0$ -batches since last $\mathcal{L}_{\min}$ update;
repeat
  randomly draw n_b samples from $\boldsymbol{X}$ to form a subsample $\boldsymbol{X_b}$;
  calculate K_b from $\boldsymbol{X_b}$;
  extract l_b off diagonal unique entries from K_b;
  calculate $\mathcal{L}$ (3.19) ;
  apply gradient descent to update b;
  if $\mathcal{L} < \mathcal{L}_{\min}$ **then**
    $\mathcal{L}_{\min} = \mathcal{L}$;
    $t = 0$;
  else
    $t = t + 1$;
  end
until $t > p$;
output: $\bar{\sigma}$ during p

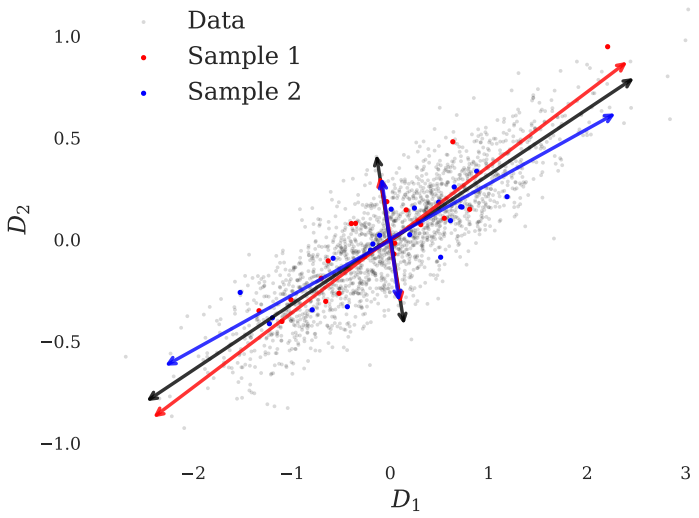


Figure 3.3: A two dimensional example of linear PCA. Random subsampling produces *skeleton* approximations of the PC.

3.1.5 Skeleton Ensembles

Even with the efficient means of computing an appropriate σ parameter, the cubic time complexity of the eigenvalue decomposition of the $\tilde{K}$ makes performing KPCA on larger datasets infeasible. An ensemble alternative technique is described that avoids the full construction or decomposition of K. The key insight is that a small sampling of a collection of points produces approximately the same principal components as the full dataset. Figure 3.3 shows a two-dimensional example using linear PCA to illustrate the point. PCA is performed on 2000 randomly generated points, the PC are indicated by the black arrows. Two separate random samplings of 20 points are taken to perform PCA again. Projections onto the sampled PC approximate that of the full model. The insight is that the knowing the true PCs is not important. Instead, the anomaly scores are averaged over many models that approximate the PCs.

The idea expands the concept of an out-of-sample extension [68], that is, a datum that

was not originally used in the eigendecomposition of K can be still be projected onto the set of principal components. Or more plainly, points that were not used to construct the approximate KPC can still be projected on to them. Most techniques focus on finding a single good approximation of the kernel[68, 99]. In the unsupervised setting this can be problematic, an unlucky random sampling may be strongly influenced by outliers. The proposed method differs in the fact that an ensemble accounts for the errors produced by decomposing a lower-rank K.

A randomly drawn number of examples n_s is drawn from $\boldsymbol{X}$, these are used to produce an approximate low-rank K_s, from which the *skeleton* eigenvectors, α_s, that form the approximate model of the data can be calculated. Then the reconstruction error (3.9) for all points in $\boldsymbol{X}$ are found. This process is repeated for to form an ensemble of, n_m, approximate low-rank models. The reconstruction errors across all ensembles are averaged for each example in $\boldsymbol{X}$ to form a final anomaly score. Algorithm 2 outlines the procedure.

The procedure of model averaging, also known as bootstrap aggregation or *bagging*, is an ensemble method that has been primarily been primarily developed for decision tree methods, such as Isolation Forest [2, 56]. Notably, this type of sampling does not work as well for some other methods such as OC-SVMs or distance-based approaches. As opposed to KPCA, where the Principal Components are comparable (Figure 3.3), the margins generated by OC-SVMs and nearest neighbor rankings vary significantly because the distance between the distances between points are much larger in the sample, especially in higher dimensional data.. Similar methods have applied KPCA ensembles to applications such as image denoising in the approximations of pre-images [82], however, this evaluation is the first to apply an ensemble version as a general approach to the problem of unsupervised anomaly detection.

The sampling process greatly reduces the computational complexity of KPCA, even when

Algorithm 2: Ensemble KPCA

input : $\boldsymbol{X}$ -globally min-max normalized data, σ -Gaussian kernel parameter
Given: N_s -skeleton sampling size, n_m -number of models in the ensemble
for *model* in N_m **do**
  randomly draw N_b samples from $\boldsymbol{X}$ to form a subsample $\boldsymbol{X_s}$;
  form K_s from $\boldsymbol{X_s}$;
  form $\tilde{K_s}$ from K_s (3.8);
  decompose $\tilde{K_s}$ to extract $\boldsymbol{\alpha}_s$;
  unit-norm $\boldsymbol{\alpha}_s$;
  calculate $d_E(\boldsymbol{x})$ for all $\boldsymbol{x} \in \boldsymbol{X}$ (3.9);
end
output: $\bar{d_E}(\boldsymbol{x})$ -average anomaly score for each example across all N_m models

accounting for the multiple evaluations necessary in the ensemble. The computational complexity of the eigendecomposition step is reduced from $\mathcal{O}(N^3D)$ to $\mathcal{O}(N_m{N_s}^3D)$. However, the scoring across all models requires a $\mathcal{O}(N_mN)$ time, but because only modest values of N_s and N_m are necessary to approach the accuracy of the full rank evaluation, the ensemble method quickly becomes the preferred approach as the cardinality increases. Another desirable features follows the fact that each model's score can be calculated independently, meaning the procedure parallelizable.

3.2 Modified Training and Scoring Autoencoder

As described in Section 2.3.1, an autoencoder (AE) is an artificial neural network (ANN) that is trained to reconstruct inputs. AEs are restricted by designed to prevent the learning of a perfect identity mapping; this is normally achieved by a *bottle neck* where some information is lost in compression.

Compared to other dimension reduction techniques, such as Principal Component Analysis (PCA), AEs are able to perform non-linear transformations of the data via their non-linear

activation function and hidden layers. This is a useful property for detection, as the division between normal and anomalous examples is often non-linear. In addition, the mini-batch gradient descent techniques used to train AEs scale well to large datasets and higher dimensional features.

An AE can be though of as two networks. An encoder network, $\mathcal{E}$, maps data from an input example $\boldsymbol{x} \in \boldsymbol{X} \subset \mathbb{R}^D$ in ambient space to a reduced latent space $\boldsymbol{z} \in \boldsymbol{Z} \subset \mathbb{R}^K$. Then, a decoder network, $\mathcal{D}$ maps the latent space representation, back to the ambient space, $\boldsymbol{x}' \in \boldsymbol{X}' \subset \mathbb{R}^D$.

The encoder and decoder networks are jointly trained by a loss (objective) function. The loss function aims to minimize the reconstruction error between the set of inputs $\boldsymbol{X}$ and the reconstructions $\boldsymbol{X}'$. Errors are back-propagated through the network parameters. Measures of reconstruction error are used to both train the AE and provide a measure of normality for each example. The most common of which is the $l2$-based mean squared error (MSE),

$$MSE(\boldsymbol{x}) = \mathcal{L}(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - (\mathcal{D} \circ \mathcal{E})\boldsymbol{x})\|^2 . \tag{3.23}$$

The AE is trained through the iterative backpropogation of the average error across a mini-batch of examples, as with other feed forward ANN architectures.

This conventional formulation of an AE as an anomaly detector assumes that normal examples will be reconstructed better than anomalous examples based on greater frequency during training. However, there is also an implicit assumption that anomalies *cannot* be reconstructed accurately. In practice, this premise cannot be relied upon; AEs can often generalize well enough to accurately reconstruct anomalous inputs [7, 33].

The over-generalization caused by the contamination of training data with anomalies cannot

be corrected by ad hoc increase in regularization, restriction network capacity, or reduction in training time. This is because any attempt to limit the generalization of anomalies also jeopardizes the reconstruction of normal examples, leading to a high number of false positives. Due to the unsupervised setting, the optimal hyperparameter choices cannot be determined from a conventional, extensive search. New approaches are necessary.

To address the problem of AEs over-generalizing to fit anomalous data, we propose several modifications to the training and anomaly scoring of AEs used for unsupervised deep anomaly detection (DAD). This section describes these modifications and their motivations in detail.

Rather than assuming an AE *cannot* learn to generalize anomalies, anomalies are assumed to require greater time to learn. That is, anomalous examples during training will have higher reconstruction scores over more training steps. The shift in perspective allows anomalous examples to be captured by leveraging the unsupervised AE training process instead of an arbitrary single state of the network.

3.2.1 Cumulative Error Scoring

Figure 3.4 shows reconstruction error of an anomalous example and normal example from one of the experiments on the *stop-sign* dataset detailed in 4.3. The converging errors demonstrate the problem of generalization in unsupervised AE training. Based on the raw error there is not a clearly best number of training steps. The pattern is different for other normal and anomalous examples, and also varies with different random initializations of the network parameters. As a result, simply setting an arbitrary number of training steps halt training is unreliable. The diverging bold lines represent the proposed solution.

Early in training, reconstruction error of anomalous examples are well separated; yet over time, the AE learns to reconstruct the anomalous example equally well as the normal one. In

the extreme example, where the loss is zero, the AE has no ability to distinguish anomalies. Over-training is obviously problematic, but arbitrarily choosing a stopping epoch may prevent the AE from fully assimilating the normal examples. In order to allow for greater laxity in the number of training steps and to fully leverage the history of the training process, we introduce Cumulative Error Scoring (CES). CES sums the errors of each example across all training epochs, normalizing by the number of epochs J, a monotonic function that does not affect ranking. The CES for each example $\boldsymbol{x}$ is then,

$$CES(\mathbf{x}) = \sum_{j=e_b}^{J} MSE(\mathbf{x})_j, \tag{3.24}$$

where $MSE(\mathbf{x})_j$ is the reconstruction error at the end of epoch j and e_b is the number of *burn-in* epochs.

The CES metric can be understood in several ways. Ensemble techniques (model averaging) are often used to add robustness to ANN training [34]. The summation of the reconstruction errors can be thought of as an ensemble of earlier states of the model during training. CES can be alternatively viewed as an approximation of the integrated error, as shown in Figure 3.4.

The bold lines in Figure 3.4 represent the cumulative errors of both anomalous and normal examples. CES more reliably separates the anomaly over the course of training. Additionally, the summation places less significance on later epochs where the reconstruction errors are smaller and the network may be overgeneralizing anomalies. Specifying a small number of *burn-in* epochs, e_b, acts to ignore the some of the initial period of training where the reconstruction errors are not reliably indicative of normality.

However, CES does not yield a final trained model that can be applied to unseen new data. Instead, it can only be applied to identifying anomalies as part of a full dataset during the

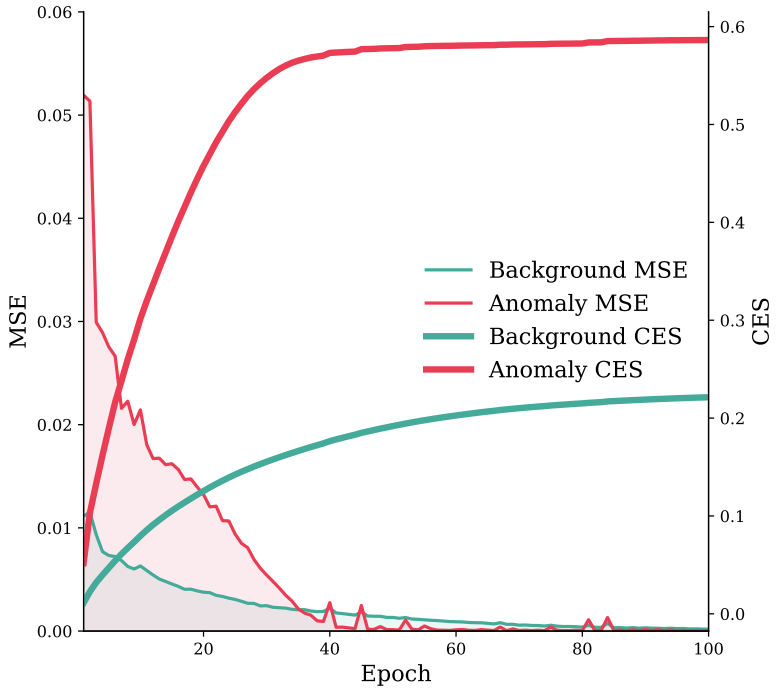


Figure 3.4: Reconstruction error over 100 epochs for a normal (green) and anomalous (red). Bold lines show cumulative error

act of training. One possible, but costly solution is to retain the historical states of the model during training and evaluating new examples with each of these states. In our research we only consider the unsupervised case where there is no distinct testing phase, again referring to Figure 1.1.

3.2.2 Percentile Loss

CES improves accuracy but does not directly address the contamination of training data by anomalies. This is evident by the decreasing trend in detection accuracy as measured by AUC, in figure 5.6. After the AE is able to generalize the anomalies, continued summation of error degrades performance over time as anomalous reconstruction errors are no longer reliably higher than those of background examples.

Even if the training data contains a relatively low percentage of anomalies (θ), there is still a significant probability that an anomaly will be present in any given mini-batch, as described by the hypergeometric distribution. The probability of contamination, P_c, that a randomly drawn mini-batch of size N_b contains at least one anomaly from a dataset that contains N is given by,

$$\mathsf{P_c} = 1 - \frac{\binom{\theta N}{0}\binom{N-\theta N}{N_b}}{\binom{N}{N_b}}. \tag{3.25}$$

A example calculation can provide some insight. If $N = 1000$, $\theta = 1\%$, and $N_b = 100$, then $\mathsf{P_c} = 65.3\%$. In this example, most mini-batches will contain at least one anomaly, regardless of the detector's current ability to distinguish anomalous examples.

When anomalies are present in a mini-batch, they contribute to the parameter updates of the network. Worse yet, because anomalies are intended to have the highest reconstruction

errors they contribute disproportionately. This problem stems from the conventional duel use of reconstruction error as both the training target and the basis of an anomaly score (3.23). The training directly acts to reduce the anomaly scores of anomalous examples.

The goal of modifying the training objective is to adapt AEs specifically for anomaly detection. The proposed method, Percentile Loss (PL), undermines an AE's ability to learn anomalies while still encouraging the generalization of normal examples. PL leverages the assumption that early in training, the anomalous examples will more often generate the highest errors in a given mini-batch. Or otherwise stated, the AE will be able to separate many of the anomalies by their higher reconstruction scores.

Rather than updating parameters based on the errors of all the examples in a mini-batch, we define an upper percentile q (*e.g.* $q = 95\%$) and a reconstruction error in each mini-batch corresponding to that percentile, P_q. PL then only performs parameter updates based on the reconstruction errors *less than* P_q; allowing the AE to *ignore* most anomalous examples during training.

Yet, even if the AE-based detector perfectly rank all anomalous examples above normal examples, it is still possible that a randomly drawn mini-batch contains enough anomalies that some number exists below the threshold. Nonetheless, this probability is much less than before. The probability of contamination by at least one anomaly *less than* q in a perfect detector is given by

$$\mathsf{P_c}* = 1 - \frac{\sum_{i=0}^{N_b-L} \binom{\theta N}{i} \binom{N-\theta N}{N_b-i}}{\binom{N}{N_b}}, \tag{3.26}$$

where we define L to be the position of q, $L = \lfloor N_b(q/100) \rfloor$. Using the same example values as before, there is a massive reduction in the probability, $\mathsf{P_c}* = 0.15\%$. The difference decreases for worse (non-perfect) detectors and higher anomaly percentages.

PL helps to prevent anomalies from contributing to parameter updates when they are present in the training data. One potential issue is that PL can cause normal examples above the percentile threshold to be ignored. However, PL relies on the assumption that by sufficiently training on other normal examples the AE is still able to generalize the more difficult normal examples better than the anomalies. MSE (3.23) serves as the base loss function for all evaluations in this paper; however, the application of PL can easily be extended to other reconstruction-based training objectives.

3.2.3 Early Stopping via Knee Detection

Despite the protections afforded by CES and PL, the AE degrades slowly over many training steps as PL can frustrate, but not fully stop, anomalous examples from contributing to parameter updates. CES and PL reduce the sensitivity to the number of training steps, but a means of reliably halting training is still required. In most applications of ANN, the loss metric of a hold-out validation set of the training data is used to determine the cross-over point of under to over-fitting. This cannot be transferred to the case of DAD. The rarity of anomalies cause them to contribute little to the overall loss metric. Furthermore, the value of the loss is tied to the value of the input features. The proposed of early stopping that is not sensitive to the magnitude of the loss.

Averaging the CES of all examples at the end of each epoch creates a smoother loss statistic as seen in Figure 3.5. The CES is divided by the number of epochs so that it has the same concavity as the conventional MSE loss. Figure 3.5 shows a run from the *stop-sign* dataset discussed in Section 4.3. The thin red line indicates the conventional MSE loss. The bold red line indicates the CES loss (average loss), which is the average cumulative error normalized by the epoch. The knee in the CES Loss curve curve then serves as a reliable criterion to

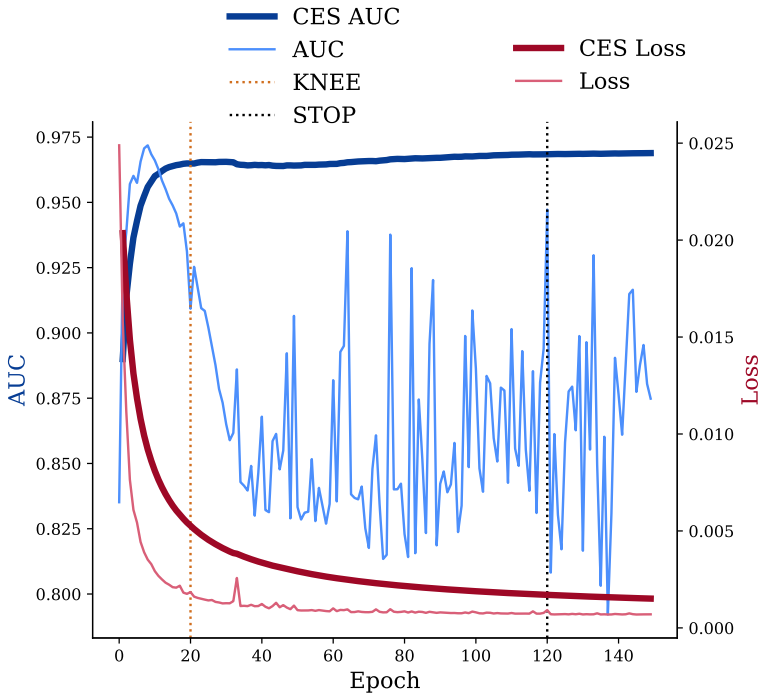


Figure 3.5: Early stopping via knee detection on the CES loss statistic. PL is also applied in this example.

end training. The Kneedle algorithm [85] is used to determine the knee in an online fashion at the end of each epoch.

The Kneedle algorithm defines a mathematical definition of curvature as the basis for knee detection. For a continuous function f the curvature, $C_f(x)$, at any point, x is a function of the first and second derivatives of f,

$$C_f(x) = \frac{f''(x)}{(1 + f'(x)^2)^{1.5}}. \tag{3.27}$$

The knee is then defined as the point of maximum curvature. Yet due to the discrete nature of the data forming the curve, the formulation in [85] cannot be directly applied. Instead,

because the maximum curvature can be defined as a maximum distance from a straight line, the Kneedle algorithm tracks a literal measure of distance between each point and line connecting the end-points of the discrete curve. The algorithm then attempts to estimate the point at which the distance begins to decrease after a period of increasing.

Because of noise, a threshold is used based on a number of points. The sensitivity parameter S, search for a consecutive number of *flat* points, or approximate local extreme, before declaring a knee. A smaller S declares a knee more quickly, while a larger value for S indicates more patience [85]. For detection on the AE's cumulative loss statistic, the sensitivity parameter for algorithm is set to $S = 5$.

The location of the knee changes over the course of training, but the drift is typically slow and consistent. The stopping epoch, j_{stop}, is determined according to the location of the knee epoch, j_{knee}, by defining a parameter B. If $J > B \times j_{knee}$, then $J = j_{knee}$ and training is halted. Some buffer a head of the knee is required as there is a latency in detection. Generally, selecting B between 2 and 8 works well.

3.3 Summary

Overall, we introduced several methods to better adapt KPCA and AEs for unsupervised anomaly detection. First, for KPCA, Algorithm 1 casts the σ tuning objective introduced by Evangelista and Embrechts [27] to a mini-batch gradient descent framework, allowing for the unsupervised selection of the kernel parameter. The cubic time cost from the eigendecomposition of the full kernel matrix is avoided by the model averaging method outlined in Algorithm 2. Algorithms 1 and 2 together, create Unsupervised Ensemble KPCA (UE-KPCA), and allows KPCA to be used in the unsupervised setting without *any* full evaluations of the kernel matrix, resulting in an approximately linear time efficiency. This could provide

significant computational efficiencies on larger datasets.

Next, we introduced a combined approach to address the problems of overgeneralization in AEs, namely: cumulative error scoring (CES), percentile loss (PL), and early stopping via knee detection. CES leverages the history of training errors to better separate anomalous and background points. PL diminishes the influence of anomalies on parameter updates, undermining the ability of AEs to learn anomalous examples. Lastly, the smooth cumulative loss statistic is leveraged as a reliable means of early stopping to prevent over-training. AEs also scale well to large volumes. The time complexity is $\mathcal{O}(J_T N D h)$, where h is the number of hidden nodes in the network and J_T is the total number of training and scoring epochs [71].

Chapter 4

Experimental Methods

The two new methods proposed in our research, the Unsupervised Ensemble Kernel Principal Component Analysis (UE-KPCA) outlined in 3.1 and the Modified Training and Scoring Autoencoder (MTS-AE) described in 3.2, are compared to a number of other state-of-the-art unsupervised anomaly detection methods across a number of benchmark datasets. This chapter outlines the methods used for this comparison evaluation. Section 4.1 details the other algorithms used for comparison, while Section 4.3 describes the sources and nature of the benchmark datasets. Section 4.2 explains the parameter settings used to draw a fair comparison and the specifics of the hardware and software tools employed. Lastly, Section 4.4 describes the metrics used to evaluate the anomaly detection performances of the different methods.

4.1 Baseline Algorithms

Though a huge number other algorithms exist, many with several different extensions and variations, the baselines chosen have been demonstrated to be among the generally best-performing and widely adopted methods on tabular multivariate (point) datasets [12, 32, 71]. For the vector notation, each data example $\boldsymbol{x}$ belongs to a dataset $\boldsymbol{X} \subset \mathbb{R}^{D \times N}$, where N is the number of examples and D is the dimensionality. Anomaly scores for each baseline are formulated so that a higher value indicates a more anomalous point.

4.1.1 k^{th} Nearest Neighbor

The k^{th}-nearest-neighbor (k^{th}NN) algorithm is an unsupervised method to detect anomalies [73]. First, the pairwise Euclidean distance between all points is calculated. Then a parameter, k specifies the k closest points, or neighbors, for each example. Among these neighbors, the distance to the k^{th} furthest point serves as the anomaly score. An alternative version uses the average distances to all k neighbors [32].

If the choice of k is too small, then the measure is very susceptible to noise and may not capture broader patterns in the data. Too large of a choice for k, then finer changes may be missed. In the unsupervised setting, a value within the range $10 < k < 50$ has been shown to be a descent rule-of-thumb for good detection performance [12, 32]. The algorithm is equivalent to sorting each of the rows of the adjacency matrix by ascending and selecting the entries of $k+1$ column as the anomaly score. The computational complexity is $\mathcal{O}(N^2D)$.

4.1.2 Local Outlier Factor

The simple distance measure used in k^{th}NN assumes that each neighborhood of surrounding points can be treated equally when scoring anomalies. Yet, in many datasets, different partitions of the data exhibit different patterns, so that a measure of anomaly is specific to the surrounding region. These *local* outliers can be identified by assigning each example a Local Outlier Factor (LOF) [10].

LOF can be though of as a measure of relative density between an example and its neighbors. The first step involves calculating the k-distance, $d_k(\boldsymbol{x})$, between an example and its k^{th} nearest neighbor. The neighborhood, $N_k(\boldsymbol{x})$ around $\boldsymbol{x}$ can then be defined as all the examples contained in this radius. A reachability distance, d_R, can then be defined between the

example of interest and each point, $\boldsymbol{x}'$, in $N_k(\boldsymbol{x})$,

$$d_R(\boldsymbol{x}, \boldsymbol{x}') = \max\left(d_k(\boldsymbol{x}'), d(\boldsymbol{x}, \boldsymbol{x}')\right), \tag{4.1}$$

where $d(\boldsymbol{x}, \boldsymbol{x}')$ is the Euclidean distance between the two points and $d_k(\boldsymbol{x}')$ is the k-distance for $\boldsymbol{x}'$. Notably, this *distance* is not symmetric because $d_R(\boldsymbol{x}, \boldsymbol{x}')$ a does not necessarily equal $d_R(\boldsymbol{x}', \boldsymbol{x})$.

The local reachability density (LDR) of an example is found by inverting the average reachability distance in the neighborhood,

$$LDR(\boldsymbol{x}) = \left(\frac{\|N_k\|}{\sum_{\boldsymbol{x}' \in N_k} d_R(\boldsymbol{x}, \boldsymbol{x}')}\right). \tag{4.2}$$

Finally, the LOF for $\boldsymbol{x}$ is determined by comparing the LDR of the example to those calculated for neighborhood,

$$s_{LOF}(\boldsymbol{x}) = \left(\frac{\sum_{\boldsymbol{x}_k \in N_k} \frac{LDR(\boldsymbol{x})}{LDR(\boldsymbol{x}')}}{\|N_k\|}\right). \tag{4.3}$$

A higher LOF indicates a lower relative density compared to the neighborhood and serves as the anomaly score. As with, k^{th}NN the choice of k can significantly affect performance. The same heuristic is followed, where k is selected in the range $10 < k < 50$. The computational cost is comparable to that of k^{th}NN, as the most expensive step involves obtaining the pairwise distances.

4.1.3 Unweighted Cluster-Based Local Outlier Factor

The Unweighted Cluster-Based Local Outlier Factor (uCBLOF), uses clustering to estimate areas of greater density [40]. The *unweighted* refers to a divergence of the original formulation,

where the scores are no longer weighted by the populations of the cluster, as this modification has shown to generally improve performance [4, 32]. The initial step of the algorithm is to classify examples into either a small cluster (SC) or a large cluster (LC). The anomaly score of $\boldsymbol{x}$ belonging to the cluster C_i is then the Euclidean distance to the C of the nearest LC,

$$s_{uCBLOF}(\boldsymbol{x}) = \begin{cases} \min(d(\boldsymbol{x}, C_j)), & \text{if } C_i \in SC, C_j \in LC \\ d(\boldsymbol{x}, C_i), & \text{if } C_i \in LC \end{cases} \tag{4.4}$$

The kMeans algorithm is used to for cluster assignment [57], making the uCBLOF algorithm non-deterministic. Two parameters, α and β decide if a cluster is considered larger or small. For a list of all clusters sorted in descending order by membership size, $C_1, C_2, \ldots C_n$, b is defined as the boundary of large and small clusters. If either of the conditions $\sum_{i=1}^{b} |C_i| \leq \alpha N$ or $C_k/(C_k - 1) \leq \beta$ are satisfied while traversing through the list, then b is declared [40].

In this evaluation, the values for the cluster-separating parameters are set to $\alpha = 0.95$ and $\beta = 5$, as per the author's suggestion [32, 40]. Additionally, The number of clusters to compute represents an important parameter and again follows the same rule-of-thumb used for selecting k. Because clustering is a less common approach, there is not a well established heuristic. Clustering assignment does not require the calculation of all pairwise distances; therefore, the complexity is linear with N at $\mathcal{O}(NDki)$, where i is the number of iterations by kMeans until convergence.

4.1.4 Linear Principal Component Analysis

The linear formulation of principal component analysis (PCA) is a commonly used in ML for dimensionality reduction and data exploration. However, PCA can also be applied to anomaly detection following a similar procedure described in 3.1. The inclusion of this baseline is also useful for an ablation study to determine the impact of the non-linear, kernel extension. The goal of PCA is to find a lower-dimensional representation that accurately reconstructs a majority of the original, mean-centered data [45].

$$\min_{\boldsymbol{W},\, \boldsymbol{z}} \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{x}_i - \bar{\boldsymbol{X}} + \boldsymbol{W}^T \boldsymbol{z}_i\|^2, \tag{4.5}$$

subject to:

$$\boldsymbol{W}\boldsymbol{W}^T = \mathbb{I}, \tag{4.6}$$

where $\boldsymbol{W}$ is the transformation matrix, $\boldsymbol{z}$ is the reduced representation of the data, and $\bar{\boldsymbol{X}}$ is the data mean.

The minimum reconstruction is accomplished by aligning the data along orthogonal axes with the highest variance. To do so, the data is first centered, $\tilde{\boldsymbol{X}} = \boldsymbol{X} - \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{X}$. Next, the eigendecomposition of the covariance matrix, $\boldsymbol{\Sigma}$, yeilds the eigenvectors, $\boldsymbol{V}$,

$$\boldsymbol{\Sigma} = \tilde{\boldsymbol{X}}\tilde{\boldsymbol{X}}^T = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^T, \tag{4.7}$$

where $\boldsymbol{\Lambda}$ is the diagonal matrix whose elements correspond to the eigenvalues, $\boldsymbol{\Lambda}_{ii} = \lambda_i$. A set of r eigenvectors corresponding to the sorted highest eigenvalues, $\boldsymbol{V}_M = [\boldsymbol{V}^1, \boldsymbol{V}^2, \ldots, \boldsymbol{V}^M]$ such that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r$. This set of eigenvectors forms the transformation matrix

$\boldsymbol{W} = \boldsymbol{V}_M^T$. Data can be projected into the reduced latent space by $\boldsymbol{Z} = \boldsymbol{W}\tilde{\boldsymbol{X}}$, so that the number of retained eigenvectors, M corresponds to the dimensionality of the transformed data, $\boldsymbol{z} \in \boldsymbol{Z} \subset \mathbb{R}^{M \times N}$.

Anomaly detection is performed by calculating the reconstruction error (residuals) between the full and reduced data,

$$s_{PCA}(\boldsymbol{x}) = (\boldsymbol{x} - \bar{\boldsymbol{X}})^T(\boldsymbol{x} - \bar{\boldsymbol{X}}) - ((\boldsymbol{x} - \bar{\boldsymbol{X}})\boldsymbol{V}_M^T)^T((\boldsymbol{x} - \bar{\boldsymbol{X}})\boldsymbol{V}_M^T). \tag{4.8}$$

Because the eigenvectors are more indicative of the normal data, anomalies should produce greater reconstruction errors [45]. The number of eigenvectors to retain corresponding to the latent dimensionality, M, represents the crucial parameter. With no established guidelines for selecting this parameter, initial experiments showed that $M = D \,/\, 2$, rounded down, was a reasonable choice. The computational complexity of PCA is reasonable for low dimensional data. The calculation of the covariance matrix is $\mathcal{O}(D^2N)$; and its eigendecomposition is $\mathcal{O}(D^3)$.

4.1.5 Mahalanobis Distance

The Mahalanobis distance (MD) assumes a normal model of the data in order to create a unitless, scale-invariant, multivariate generalization of a Z-score [58]. There is a strong connection to PCA, where calculating MD can be thought of as first orienting the data along the principal components and then standardizing so that the data has unit variance along each principal component axis. The anomaly score is then the distance from the origin in this transformed representation given by

$$s_{MD}(\boldsymbol{x}) = \sqrt{(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}-\boldsymbol{\mu})}, \tag{4.9}$$

where $\boldsymbol{\Sigma}$ is the covariance matrix and $\boldsymbol{\mu}$ is a vector of the feature means.

The main appeal of using MD is that it not parameter selection; nevertheless, its effectiveness relies on a single, multivariate-Gaussian model fitting the data. Again, the calculation of the covariance matrix is $\mathcal{O}(D^2N)$; and its inversion is $\mathcal{O}(D^3)$.

4.1.6 Kernel Density Estimator

The Kernel Desnity Estimator (KDE), also known as the Parzen-Rosenblatt Density Window, constructs a probability density function (pdf) from the summation of the kernel functions centered at each point [43]. This can be thought of as superimposing the functions in Figure 3.2. An anomaly score is generated by measuring the value of the pdf at each point and reversing the sign,

$$s_{KDE}(\boldsymbol{x}) = -\frac{1}{N}\sum_{i=1}^{N} \kappa(\boldsymbol{x}_i, \boldsymbol{x}). \tag{4.10}$$

Scores closer to zero indicate anomalies. For a Gaussian kernel, the effectiveness of the KDE relies on the paramter choice, σ.

Conceptually, the KDE produces the same ranking as the spherical potential (3.10), or the distance from the origin in the feature space, $\mathcal{F}$, after mean-centering the data. That is, the KDE is equivalent to KPCA if no principal components are retained ($M = 0$) [43]. Because the KDE does not required the eigendecomposition step, but still requires construction of the full kernel matrix, the time complexity is $\mathcal{O}(N^2D)$.

4.1.7 One Class Support Vector Machines

For any kernel where $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_i) = 1$, such as the Gaussian kernel, the data in the feature space has a unit norm. As a result the points in the transformed space can be lie on the surface of a hypersphere in feature space. A maximal-margin hyperplane can be constructed to separate a majority of the points from the origin in $\mathcal{F}$ [9, 54]. The One-class Support Vector Machine (OC-SVM) is a method of constructing a soft margin separating hyperplane in the feature space to identify anomalous points [86]. For Gaussian Kernel, the OC-SVM is geometrically equivalent to the Support Vector Data Description (SVDD), which finds the smallest hypersphere that encloses the data in $\mathcal{F}$ [94], referring back to Figure 2.2 for a simplified visualization.

The problem can be solved via quadratic programming,

$$\min_{w,\ \xi_i,\ \rho} \frac{1}{2}\|w\|^2 + \frac{1}{\nu n}\sum_{i=1}^{N}\xi_i - \rho \tag{4.11}$$

subject to:

$$w \cdot \phi(x_i) \geq \rho - \xi_i \qquad \text{for all } i = 1, \ldots, n, \tag{4.12}$$

$$\xi_i \geq 0 \qquad \text{for all } i = 1, \ldots, n, \tag{4.13}$$

where w and ρ define the hyperplane, ξ_i are the slack variables to relax the constraint of the margin, and ν characterizes the soft margin solution.

The formulation can be modified to use Lagrange techniques and replace inner products with the kernel function (3.6).After introducing the multipliers $\boldsymbol{\alpha}_i, \boldsymbol{\beta}_i \geq 0$, the Lagrangian becomes,

$$L = \frac{1}{2}\|w\|^2 + \frac{1}{\nu n}\sum_{i=1}^{N}\xi_i - \rho - \sum_{i=1}^{N}\alpha_i((w \cdot \Phi(\boldsymbol{x_i})) - \rho + \xi_i) - \sum_{i=1}^{N}\beta_i\xi_i \tag{4.14}$$

The derivatives of L with respect to the primal variables w, $\boldsymbol{\xi_i}$, and ρ are set equal to zero, resulting in

$$w = \sum_{i=1}^{N}\alpha_i\Phi(\boldsymbol{x_i}), \tag{4.15}$$

$$\alpha_i = \frac{1}{\nu n} - \beta_i \leq \frac{1}{\nu n}, \quad \sum_{i=1}^{N}\alpha_i = 1. \tag{4.16}$$

After substituting Equations (4.15) and (4.16) back into (4.14) and then applying kernel trick (3.5) to replace the inner products, the dual objective becomes,

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{4.17}$$

subject to:

$$0 \leq \alpha_i \leq \frac{1}{\nu n} \qquad \text{for all } i = 1, \ldots, n, \tag{4.18}$$

$$\sum_{i=1}^{N}\alpha_i = 1 \qquad \text{for all } i = 1, \ldots, n. \tag{4.19}$$

The small number of non-zero $\boldsymbol{\alpha}$ are the support vectors that define the hyperplane. The hyperplane's constant bias does not affect ranking, therefore by using equation (4.15) the anomaly score can be expressed as the projection distance onto the hyperplane's normal vector, in terms of the support vectors and kernel,

$$s_{OCSVM}(\boldsymbol{x}) = -\sum_{i=1}^{N}\alpha_i\kappa(\boldsymbol{x}_i, \boldsymbol{x}), \tag{4.20}$$

where a value closer to zero indicates a more anomalous example. The combined primal and dual optimization yields a complexity that is $\mathcal{O}(\max(N, D)\min(N, D)^2)$ [17] making it impractical for very large datasets.

The parameter ν indicates 1) an upper bound on the fraction of training errors and 2) a lower bound of the fraction of support vectors, such that

$$\frac{Outliers}{N} \leq \nu \leq \frac{SV}{N} \tag{4.21}$$

It is useful conceptually to note that for $\nu = 1$ the two optimization constraints, allow only for one solution where $\alpha_1 = \alpha_2 = \cdots = \alpha_n = 1/N$. In this case where the number of support vectors is equal to the number of training examples, the anomaly score (4.20) produces an equivalent ranking to that of the KDE. Alternatively, this condition is equivalent to averaging all the transformed points so that the normal of the hyperplane extends through the center of the data. Sparsity in the model, gained by reducing the number of support vectors, acts to regularize the solution by ignoring outliers.

There is not an accepted method for generally selecting this parameter. The optimal choice is data dependent and difficult to select in the absence of labeled validation data. To represent the variation caused from manually selecting ν, this evaluation follows [32], where ν is varied across a range of $0.2 < \nu < 0.8$. While the Gaussian kernel is a good general choice for the One-Class Support Vector Machines (OC-SVM), as with the KDE and KPCA, the results are also sensitive to the kernel bandwidth, σ [11, 27].

4.1.8 Isolation Forest

Isolation Forest (iForest) is a tree ensemble method that aims to explicitly isolate anomalies rather than model background data [56]. Each Isolation Tree in the ensemble is built by first

selecting a random feature $q \in Q$ and a random value p between the min and max of q in $\boldsymbol{X}$. The data is then partitioned on either side of p. The process is then repeated recursively on partitions to form the branches of the tree. The tree is fully grown when each partition contains only one example, separating every point in $\boldsymbol{X}$. In practice, it is not necessary to isolate all normal instances, instead the ensemble works well when each iTree is built from a different subsampling of the data.

Anomalies, which take less partitions to isolate, should have shorter *path lengths*, $h(\boldsymbol{x})$ in a tree. The average path length for all points in a tree is given as,

$$\bar{c}(\Psi) = 2\ln(\Psi - 1) - \frac{2(\Psi - 1)}{N} + 2\gamma \tag{4.22}$$

where $\gamma = 0.5772$, the Euler-Masheroni constant, and Ψ is the sample size from $\boldsymbol{X}$ used to form each tree. A particular examples expected path length, $E(h(\boldsymbol{x}))$, across an ensemble of t iTrees can then be normalized by the average found by 4.22, to form an anomaly score,

$$s_{IF}(\boldsymbol{x}) = 2^{\frac{-E(h(\boldsymbol{x}))}{\bar{c}(\Psi)}} \tag{4.23}$$

Values of $s_{IF}(\boldsymbol{x})$ approaching 1 strongly indicate an anomaly. If $s_{IF}(\boldsymbol{x})$ is smaller than 0.5, then $\boldsymbol{x}$ is likely normal.

Following the authors' suggestion, the parameters were set to $\Psi = 256$ and $t = 100$ [56]. The generation of the trees is non-deterministic; therefore, during evaluation, multiple runs are performed to express the variation. The sampling allows for a constant memory complexity, but the evaluation still requires a linear time complexity.

4.2 Parameter Settings and Implementation

Creating a fair evaluation of different anomaly algorithms is challenging. Often different reviews take different approaches. Goldstein *et al.* advocates for attempting to find a generally good set of parameters, while expressing some of the variability due to parameter selection and initializations [32]. Because labels are not present in unsupervised anomaly detection, cross-validation cannot be used for optimal parameter selection. As a result, rules-of-thumb for these choices must be adopted. To simulate this uncertainty found in practice, methods were evaluated across the reasonable parameter ranges described for each method. Multiple trials (10) were performed for each baseline.

For a fair comparison, the same σ found for UE-KPCA by the Algorithm 1 outlined in 3.1.4 is used for the KDE and OC-SVM which both use a Gaussian Kernel. The RMSProp optimizer with a learning rate of 0.001, a patience of $p = 1000$, and batch size of $N_b = 100$ is used for tuning. Section 5.1 justifies the settings of $N_m = 100$ and $N_s = 256$ for UE-KPCA.

Though Algorithm 1 removes the requirement of manually selecting σ, there is still the important parameter choice of M, the number of eigenvalues to retain. The ideal choice of M best models the normal data, while failing to capture anomalous points. This is impossible to know without validation, but setting M equal to D is a logical choice for most datasets with no prior knowledge. However, as D grows, this choice becomes less reasonable as often many of the features of very high dimensional data, such as images, are highly correlated. Therefore, and arbitrary maximum of $M = 75$ is set. More reliably specifying M is remains an open question.

The MTS-AE has a large number of hyperparameter settings. Again, these cannot be optimized for any particular dataset without labeled information. In practice, MTS-AE would likely only be used in a *weakly* supervised setting where a limited number of labeled data

from a similar domain could be sued to verify these choices. A small number of initial experiments on the *forest* and *shuttle* datasets (Section 4.3) identified a set of generally reliable, but in no way optimal, set of choices. The same general architecture is used for all datasets: FC($D, D/2$) to FC($D/4, D/2$) to FC($D/2, D$), where FC(i, o) indicates the input and output dimensionality of each fully connected (FC) layer and D is again the ambient dimensionality. Non-integer values are rounded up. This scheme ensures that datasets with more features, and likely more complicated patterns, have a network with a greater expressive power [71]. A sigmoid activation is applied to the output of each hidden layer, while the final layer has a linear activation. The learning rate is set to 0.001 and an L_2 regularization of $1e-5$ is applied to output of each layer. Again, the Adam optimizer is used. To account for widely different dataset sizes, one epoch is defined as 200 training steps with a batch size of 256. Before beginning CES or early stopping detection, networks are pretrained with a $e_b = 10$ epoch *burn-in* period using a standard MSE loss function. This allows for some meaningful ranking of the reconstruction errors to occur before PL is applied. The knee-multiple parameter is set to $B = 5$.

AEs were built using the Keras 2.3.1 API using the Tensorflow-GPU 2.0 backend in Python 3.7.4 [1, 18]. The Isolation Forest, One-Class Support Vector Machine, Local Outlier Factor, and k^{th}-nearest-neighbor methods are implemented using the *sklearn* and *Pyod* libraries in Python [102]. Experiments are run on a desktop with a i7-9700F CPU with 16 GB of RAM and an NVIDIA RTX 2060 Super GPU.

4.3 Datasets

Unsupervised anomaly detection does not use labeled data in the determination of anomalies; however, labeled data are required for evaluation in order to determine accuracy and

specificity. The datasets used in this evaluation vary in size, dimensionality, anomaly abundance, and domain, and were selected from those that appear in meta-reviews of anomaly detection methods.

From the most recent broad review of it's kind by Goldstein and Uchidal, the datasets –*b-cancer*, *pen-global*, *pen-local*, *satellite*, *letter*, *shuttle*, and aloi are used [32]. For a broader comparison, *stamps* and *waveform* were obtained from the a separate repository [12]. Lastly, four additional benchmark datasets-*glass* and *vowels*-were sourced from the Outlier Detection datasets (ODDS) [75]. Many of these *curated* anomaly datasets originated from data retrieved from the UCI machine learning repository [25], where some data has been down sampled to constitute anomalous instances. The very high volume *forest* dataset was taken from a common baseline in remote sensing literature [70].

Lastly, *stop-sign* and *speed-sign* datasets were generated by sampling normal data from the German Traffic Signs Detection Benchmark (GTSDB) [91] adding anomalous examples. Table 4.1 provides a summary of the number of examples, the number of features, and the percentage of anomalies.

A description of each dataset follows:

- *glass*: Attributes describe six different types of glass, one class was down sampled and identified as anomalous.
- *stamps*: The dataset contains forged, anomalous and genuine, normal examples of stamps. The features are based on color and printing properties.
- *b-cancer*: The features were extracted from a fine needle aspirate of normal healthy and anomalous cancerous cells.
- *pen-global*: Based on the UCI database of 4×4 pixel handwritten digits 0-9 from 45

writers, the digit 8 represents the normal class, and a small number of examples from each of the remaining classes serve as the anomalies.

- *stop-sign*: The Stop Sign class of the GTSDB was first selected. A manual filtering process removed examples that were incorrectly cropped or significantly distorted by motion blur to create a normal set centered 32×32 RGB images of stop signs under different conditions. A mixed anomaly class was generated by combining the most distorted of the original images with new images taken of a stop sign that is either heavily occluded or contains simulated-graffiti made by covering portions of the sign with colored tape. Each image was flattened to form a data vector. See Figure 5.9 for example images.

- *speed-sign*: Similar to the *stop-sign* dataset, this collection contains images from the $30km/hr$ Speed Limit Sign class of the GTSDB. The same procedure of manually removing images was applied to create a normal set. The anomalies consisted of the worst removed examples and generated obscured or graffiti examples.

- *vowels*: The original Japanese Vowels dataset contains time series data of nine male speakers uttering the two /ae/ vowel sounds successively. Three speakers constitute the normal examples, while one is down-sampled to create the anomalous class.

- *letter*: Sixteen features are extracted from the 26 letters of the English dataset. Three letters form the normal class, while anomalies have been sampled from the rest. To increase the challenge, the dimensionality was doubled by randomly concatenating normal features to all the normal and anomalous examples.

- *waveform*: Three classes of waves are described by 21 numeric, engineered features. The anomalies are formed from the down-sampled first class.

- *pen-local*: The previous dataset is reused, but now all of the digit classes are kept in the normal class, except the first 10 instances of the anomalous digit 4 class which are considered anomalies.

- *satellite*: The features were extracted from green, red, and infrared light bands of a satellite images. Observations taken from Red, Gray, Damp, and Very Damp Gray soil areas serve as the normal class, which differ semantically from the Cotton Crop and Vegetation Stubble anomalous examples.

- *shuttle*: Nine features describe radiator positions in a NASA space shuttle. The normal Radiator Flow class differs from examples drawn from five different anomalous positions.

- *aloi*: The Amsterdam Library of Object Images collection characterizes images of 1000 different small objects by a 27 dimensional feature vector extracted using HSB color histograms. Some object classes were down sampled to serve as anomalies.

- *forest*: A portion of the hyperspectral Image (HSI) from the Forest Radiance I (run05) of the (HYDICE) imagine spectrometer [78]. The sensor captures data from 210 equally spaced bands from approximately 400-2500 nm. Noisy bands were removed, the remaining 158 bands serve as features. The background contains mostly, grass, dirt, and forest cover. Pixels where vehicles and tents appear serve as anomalies.

The three datasets with the largest number of examples, *shuttle*, *aloi*, and *forest* are used in only in the evaluation of the linear complexity methods. This reduction is meant to narrow the comparison as the computational costs of other methods eliminate their practical application. For example, simply storing the full adjacency matrix for the *forest* dataset would require almost 2TB of memory.

dataset	N	D	θ (%)
glass	214	9	4.21
stamps	315	9	1.90
b-cancer	367	30	2.72
pen-global	809	16	11.12
stop-sign	920	3072	3.59
speed-sign	1398	3072	3.29
vowels	1456	12	3.43
letter	1600	32	6.25
waveform	3443	21	2.90
satellite	5100	36	1.47
pen-local	6724	16	0.15
shuttle	46464	9	1.89
aloi	50000	27	3.02
forest	175800	158	0.87

Table 4.1: Dataset properties where N is the number of examples, D is the number of features, and θ is the percentage of anomalies

4.4 Evaluation Metrics

Measures of anomaly detection are more nuanced than the familiar categorical classification notion of accuracy. For example, in cases where anomalies are exceedingly rare, a method that always identifies examples to be normal with a simple binary, may superficially achieve a higher accuracy. Though there is the final distinction between normal and anomalous classes, most detection methods output a continuous *anomaly score*. A score is often more useful because of the often ambiguous distinction between normal and anomalous examples as described in Chapter 1 [41, 72].

The anomaly scores for the anomalous and normal examples will produce two (typically overlapping) distributions as seen in Figure 4.1. In order to make a final distinction to separate the classes and identify anomalous examples, a threshold must be set. Without access to labeled data, this is non-trivial and naive choices can lead to superficial classification. A very low threshold might be successful in identifying every anomaly in a dataset, but if

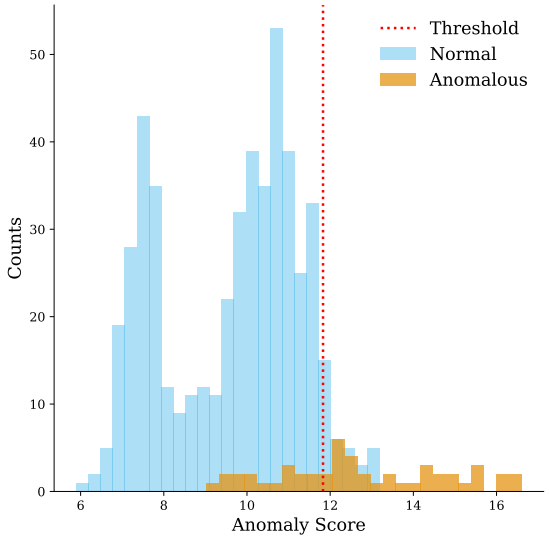


Figure 4.1: Histogram of anomaly scores. Orange indicates scores of anomalous examples, while blue represents normal ones. The red vertical line indicates a specified threshold for classification

it is at the cost of misidentifying a large number of normal examples as anomalous it is of little practical use. A normal example above the threshold is considered a false positive (FP), whereas an anomaly below the threshold is a false negative (FN). Similarly, a true positive (TP) is a correctly identified anomaly above the threshold and a true negative TN is a correctly classified normal example below the threshold.

From this we can define a true positive rate

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \tag{4.24}$$

and a false positive rate,

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{4.25}$$

and examine the interplay. TP and TN indicate the number of true positives and true

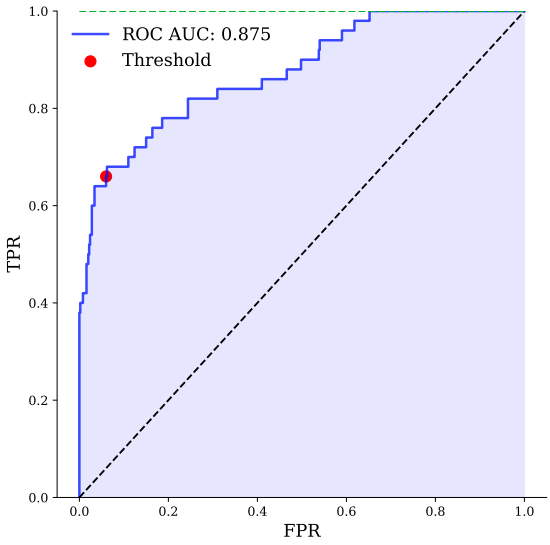


Figure 4.2: A ROC curve corresponding to the anomaly score histogram if 4.1. Each threshold generates a corresponding FPR and TPR. The green dashed-horizontal line indicates perfect detection, while the black-dashed angled line indicates random scoring. The shaded-blue AUC summarizes the ROC curve.

negatives respectively. The TPR is also known as *sensitivity* or *recall* and FPR is equal to one minus the specificity. For anomaly detection, it is desirable to establish a threshold that guarantees a high true positive rate at a low false positive rate [72].

Each anomaly score that a method produces can be thought of as a potential threshold, with a corresponding TPR and FPR. The pairs of TPR and FPR can be compared even when the actual anomaly scores and thresholds for different methods exist on very different scales. There is an obvious trade-off between TPR and FPR for most cases where the two distributions cannot be perfectly separated. Increasing the threshold, decreases the FPR, but also decreases the TPR, while decreasing the threshold has the opposite effect. A Receiver Operating Characteristic (ROC) curve plots the TPR versus FPR pairs for each threshold so that the number of points on the ROC curve is equal to the number of examples. Figure 4.2 illustrates the ROC curve from the sample distribution in Figure 4.1.

The shaded region in Figure 4.2 represents the Area Under the ROC curve (AUC or AUROC), which provides a threshold-independent performance evaluation for comparing anomaly detection methods [23]. In the unsupervised setting, AUC is most usefully interpreted as the expectation that a randomly drawn anomalous example will be scored higher than a randomly drawn normal example [29]. A method that randomly produces scores will generate thresholds with equal TPR and FPRs. A perfect method will produce a 100% TPR at 0% FPR. Consequently, AUCs for methods will range between 0.5 (random scoring) and 1.0 (perfect detection). The black dashed line in Figure 4.2 corresponds to an AUC of 0.5; the green dashed line represents an AUC of 1.0.

In addition the AUC, the FPR at 95% TPR (FPR@95%), the mean run time, and the average precision (AP) are also reported. FPR@95% is the expectation that an anomalous (positive) example is misclassified as a normal (negative) example when the threshold is selected at a TPR as high as 95% [41]. AP is the TPR-weighted mean of the precision at each threshold:

$$\text{AP} = \sum_n (TPR_n - TPR_{n-1}) P_n \tag{4.26}$$

where n indicates a threshold and precision (P) is defined as

$$\text{P} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{4.27}$$

Precision can be understood as the rate of correctly identify anomalies given a number of anomaly predictions. The weighting rewards correct anomaly identifications at thresholds where fewer are identified. The AP score can be more informative in the cases of very imbalanced data because it removes the influence of correctly predicting a normal example when normal examples are very abundant [59]. AP varies from 0-1, where 1 is closer to perfect

detection. AP is a useful metric for cases like cancer detection capturing all anomalies is of greater importance. Yet, AP does not reward correctly identifying the more abundant TN, which is important in domains such as remote sensing and surveillance.

Chapter 5

Results and Discussion

The purpose of this chapter is to present comparisons of both UE-KPCA (Section 3.1) and MTS-AE (Section 3.2) and to their constituent methods as well as a broader evaluation against the baseline techniques described in Section 4.1. Section 5.1 demonstrates the effectiveness of the mini-batch σ tuning described by Algorithm 1 and the impact of the parameters in Algorithm 2 on detection performance and efficiency. Next, Section 5.2 compares the impact of CES and PL as part of the overall MTS-AE detection. Following these ablation studies, Section 5.3 compares the proposed methods to other popular baseline algorithms.

5.1 Unsupervised Ensemble KPCA

5.1.1 Batch Sigma Tuning

Evangelista *et al.* suggested that maximizing the index of dispersion of the off-diagonal entries of the kernel matrix serves as a reasonable unsupervised means of selecting the Gaussian kernel bandwidth σ for OC-SVMs [27]. Algorithm 1 proposed in Section 3.1.4 of our reserach suggests inverting this objective to serve as a loss function to be minimized by mini-batch stochastic gradient descent and applying this σ tuning technique to KPCA.

First, the convergence of the mini-batched method is examined. The major drawback of

Evangelista's original formulation is the storage and multiple constructions of the full-rank kernel matrix, K. The goal of the proposed method is to converge on the correct σ by subsampling the data to form reduced-size approximations of K.

The goal is to test if: 1) the algorithm converges on a single value of σ and 2) what is the smallest batch sampling size, N_b, that leads to reliable converges while supporting computational efficiency. Again, a batch size of N_b produces $({N_b}^2 - N_b) \,/\, 2$ number off diagonal entries used for calculating the loss function.

Two of the datasets were selected, *shuttle* and *forest*. A log-space grid search of ten batch sizes between 1 and 1000 were tested across ten different runs. The computational time and final converged σ are reported in Figure 5.1. The error bars indicate $\pm$ 1 standard deviation across the 10 runs. Though a subsampling of $N_b = 100$ represents only 0.06 % of the total number of examples in the *forest* dataset, the small sample appears to be more than sufficient for reliable convergence. The *shuttle* dataset showed the same trend.

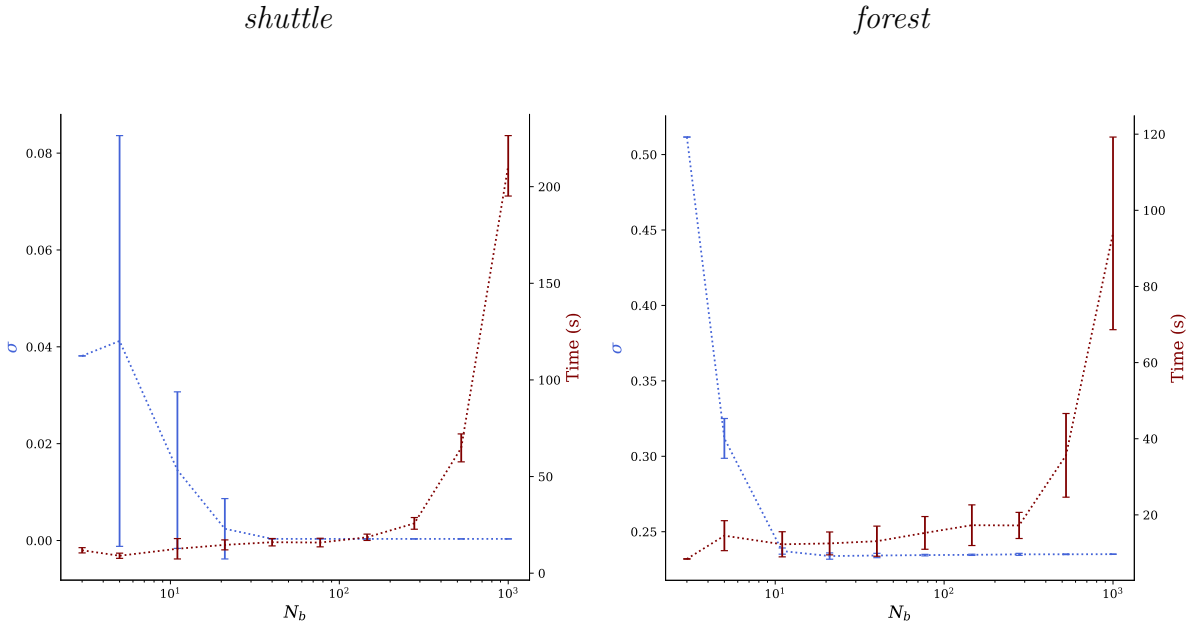


Figure 5.1: A relatively small batch size of $N_b = 100$ results in a stable σ at low computational time.

Next, it is appropriate to establish how closely the choice of σ that minimizes the loss function (3.19) is to the *ideal* choice, σ^*, that would result in the highest AUC. For each tested σ, in a log space grid of 50 values ranging from 1e$-$4 to 1, the AUC is calculated from the ROC curve corresponding to the KPCA reconstruction errors for all points in the dataset (no ensemble). Next, the loss function (3.19) is evaluated on randomly drawn batches using the same choices of σ. Figure 5.2 shows the relationship between AUC and the loss on nine of the benchmark datasets. The solid blue curve shows the variation in AUC (left axis), while the solid red curve represents the value of the loss function (right axis). The red vertical line indicates σ, which is the converged output of applying Algorithm 1. Because of sampling, this does not always appear at the global minimum of the red loss curve. The σ corresponding to the maximum AUC is shown by a blue vertical line. The figure illustrates that AUC is very sensitive to σ, and the loss function is strongly convex. While an ideal objective function would produce overlapping red and blue vertical lines, σ is near the σ^* that maximizes the AUC on across a wide variety of datasets, indicating a good generalization.

5.1.2 Ensemble Parameters

Algorithm 2 of Section 3.1.5 details how a simple ensemble of models based on sampled data can be used to greatly improve the computational efficiency of KPCA. This subsection evaluates the impact of the skeleton sample size, N_s, and the number of models, N_m, on both the AUC and run time. The goal is to maintain a high detection accuracy while keeping the computational time reasonably low. Nine sampling sizes are tested, $\{N_s = 8, 16, 32, 64, 128, 256, 512, 1024, 2048\}$, at a fixed model number of $N_m = 100$ over ten random initializations on two different datasets, *shuttle* and *forest*. Again, anomaly scores are calculated by averaging the reconstruction error of each example produced by all the models in the ensemble.

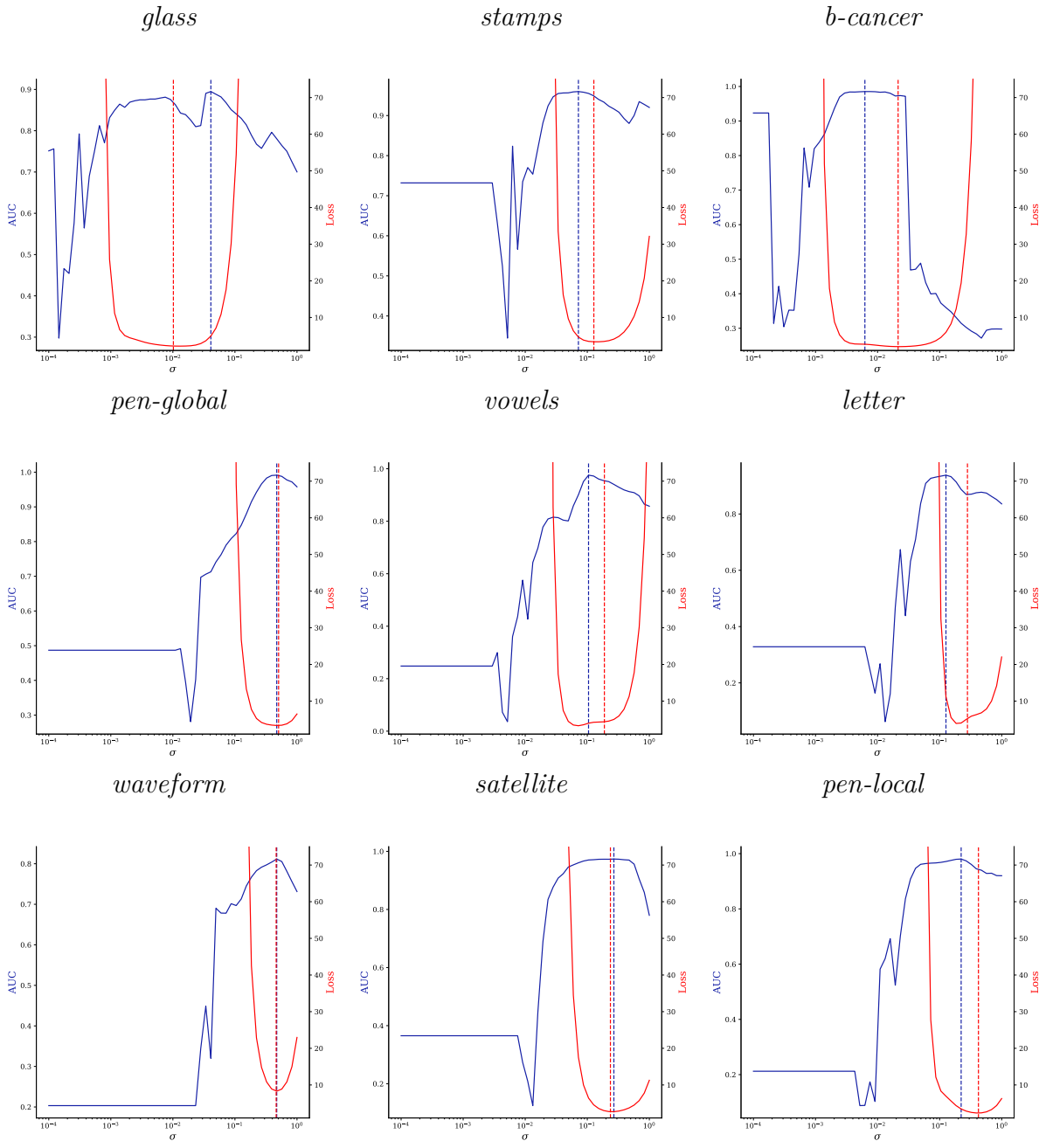


Figure 5.2: The value of the loss function (red) compared to AUC (blue) across a grid search of 50 σ choices for the Gaussian kernel on nine real datasets. The location of maximum AUC and minimum loss are indicated by dashed vertical lines.

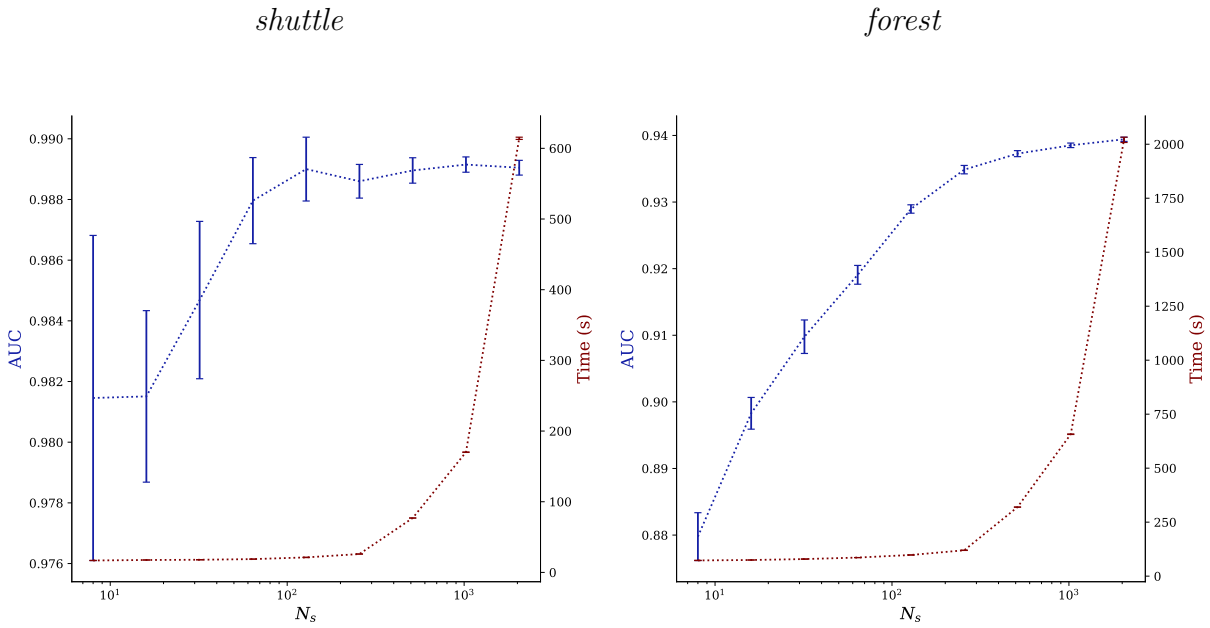


Figure 5.3: Stable results are achieved with $N_s = 256$ ofr $N_m = 100$. Computational time increases cubically with the size of the individual models in the ensemble.

Figure 5.3 shows that the AUC increases with larger sampling sizes, but the benefit quickly diminishes when considering the sharp increase in run time. This rise is expected as the eigendecomposition incurs a cubic complexity with the number of examples used in the construction of the kernel matrix. Again, the error bars indicate $\pm$ 1 standard deviation. The results show that a moderate size of $N_s = 256$ achieves near optimal performance at a low computational cost. For *forest* this represents a very small portion of the data, each skeleton is only a 0.15 % sampling.

Next, the impact of the number of models in the ensemble, N_m, is evaluated. The skeleton size is held constant at $N_s = 256$. Twenty different choices of N_m, linearly spaced from 1 to 200, are tested on the same datasets using 10 different initializations. Figure 5.4 shows that the impact on AUC is less significant than that of N_s and that run time increases linearly as expected. Though even a single model can be used, the ensemble provides a higher AUC with far less variance. The conservative choice of $N_m = 100$ is high enough to reduce the

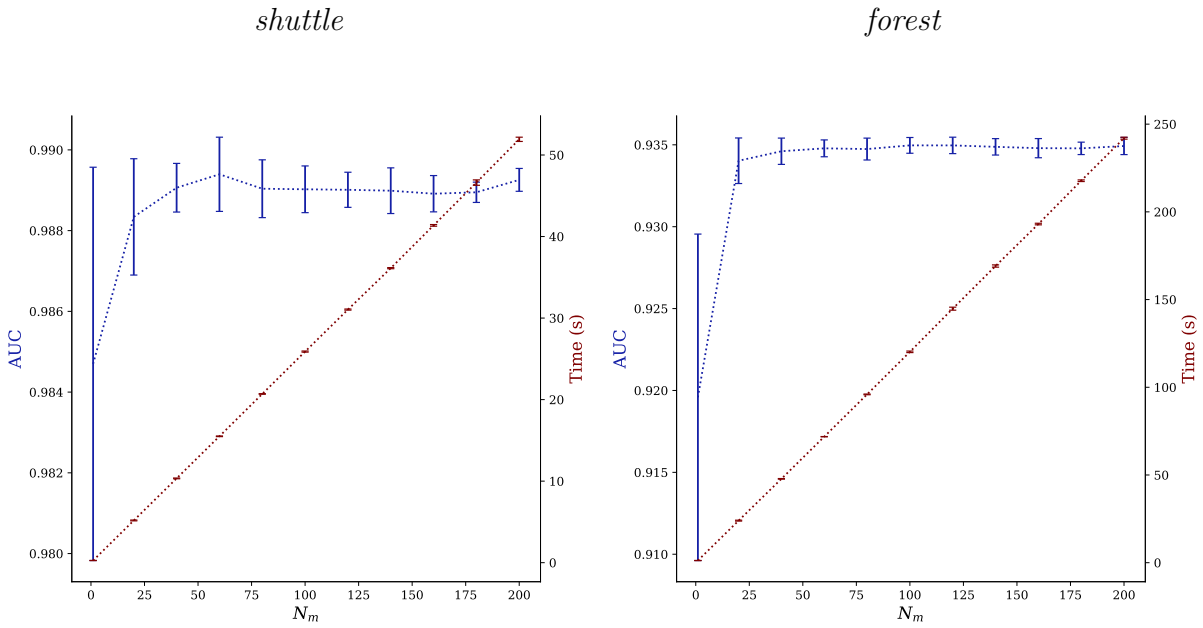


Figure 5.4: Stable results are achieved with $N_m = 100$ ofr $N_s = 256$. Computational time increases linearly with the number of models in the ensemble.

variation in AUC caused by the sampling, while not being overly taxing. These settings give an expected cross over point at $N \approx 1200$, where the ensemble method quickly becomes the computationally favorable choice compared to the original version of KPCA that uses the full kernel matrix.

Figure 5.5 shows the ROC curves of the runs on *forest*. The bold red line indicates the ensemble's result, as the thinner lines represent the individual models in the ensemble. The green lines represent the 5 out of the 100 total models in the ensemble that resulted in higher AUCs than the ensemble. Clearly, averaging the anomaly scores produces a higher AUC than the average AUC of the individual models. Not only is the variance reduced, but the detection is improved.

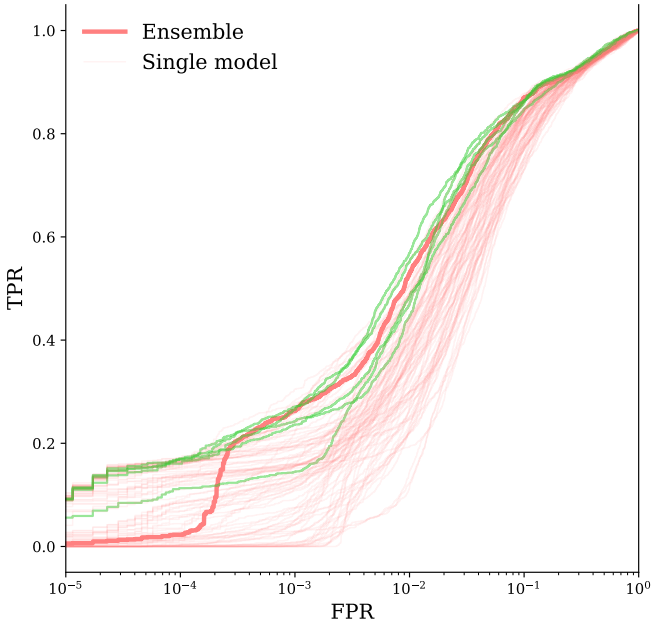


Figure 5.5: The individual results from the models in an ensemble on the *Forest* dataset. Only 5 out of the 100 individual models outperformed the ensemble average.

5.1.3 Comparisons with KPCA, KDE, and Linear PCA

UE-KPCA is first compared to the most similar methods: standard KPCA, which uses a single full kernel matrix, KDE, and linear PCA. Again, KDE is equivalent to KPCA if no principal components are retained. Table 5.1 displays the AUC. For UE-KPCA, which is non-deterministic, the reported value is the mean of 10 initializations. Later results show the variation is very small. The results show that UE-KPCA generally outperforms or matches KPCA. The Linear PCA results clearly demonstrate the effectiveness of the non-linear transformation provided by the kernel method. The generally worse performance of KDE shows that the principal components better model trends in the data.

dataset	UE-KPCA	KPCA	KDE	PCA
glass	**0.870**	**0.870**	0.851	0.627
stamps	0.948	**0.950**	**0.950**	0.207
b-cancer	**0.980**	0.974	0.960	0.263
pen-global	0.977	**0.991**	0.973	0.574
stop-sign	**0.953**	0.948	0.952	0.606
speed-sign	0.812	**0.819**	0.797	0.551
vowels	**0.955**	0.954	0.902	0.581
letter	0.907	0.909	**0.919**	0.432
waveform	0.778	**0.812**	0.760	0.525
satellite	**0.973**	**0.973**	0.963	0.626
pen-local	**0.953**	0.940	0.927	0.770
shuttle	**0.989**	n/a	n/a	0.510
aloi	**0.618**	n/a	n/a	0.524
forest	**0.935**	n/a	n/a	0.637

Table 5.1: AUC results for an ablation study on UE-KPCA

5.2 Modified Training and Scoring

The goal of MTS is to avoid the loss in detection performance caused by an autoencoder overgeneralizing the anomalies present in the training data. This section provides a comparison between MTS and standard, MSE based training. Figure 5.6 illustrates the separate roles of the two components of MTS, PL and CES, on the *stop-sign* and *speed-sign*. Even though the architecture and hyperparameters are the same in both runs, and the datasets are similar in both normal and anomalous data; the best stopping epoch is very different. This implies that any attempt to set a rule-of-thumb to end training may fail to generalize to even very similar cases. Also problematic, in both datasets the scores for anomalies invert early into standard MSE training, so that anomalies are ranked below normal examples. As this behavior is not seen on smaller capacity architectures, it is likely the result of the AE memorizing anomalous examples based on their earlier high losses, which contributed disproportionately to the gradient. The capacity of the network, relative to the number of

examples, allows the reconstruction errors of the all the examples to become exceedingly small, which also explains the flat CES curves.

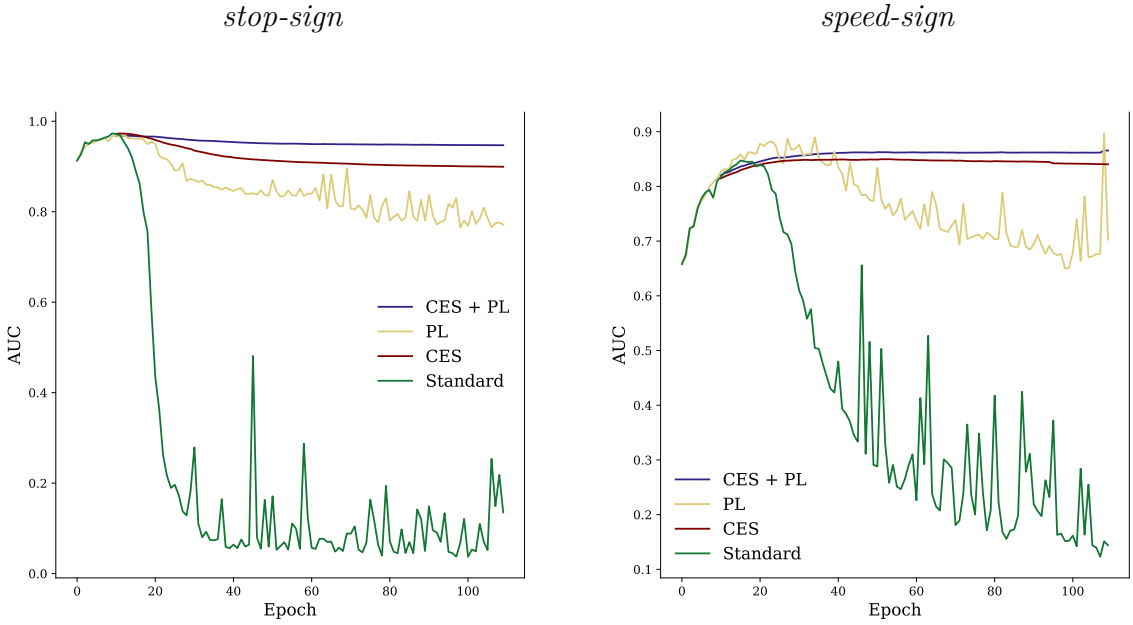


Figure 5.6: The seperate and combined effects of PL and CES

PL prevents this sharp drop in AUC by acting to exclude anomalous examples from the calculation of the loss. Figure 5.7 demonstrates how standard MSE training leads to the undesirable reconstructions of anomalies after only 50 epochs, while PL is able to frustrate the network from learning anomalies while still allowing normal examples to be reconstructed. However, Figure 5.6 shows that even with PL, overtime enough anomalies will fall below the threshold and contribute to parameter updates. Separately, CES alone provides a buffer from the erratic epoch-to-epoch performance of the standard MSE training, but also leads to a slow decline as anomalies are no longer reliably have higher reconstruction errors. Nearly optimal performance is achieved by applying both PL and CES and using knee detection to end training early. In the case of *speed-sign* a small amount of performance is sacrificed, but the detection AUC is very consistent across a large number of epochs.

Figure 5.7: After only 50 epochs the anomaly is reconstructed under the standard MSE objective. With PL, the AE does not reconstruct the anomalous example.

As discussed in Section 4.2, providing a fair comparison between MTS and standard MSE training is difficult in the unsupervised setting. In practice, different domains are likely to employ specific architectures and perform fine tuning on a small number of labeled anomalies are artificially generated ones. The best architectures and hyperparameter settings are likely to vary depending on the task. However, this evaluation attempts to draw some conclusions

based on a shared set of conditions. The same architectures and hyperparameters are used for both. As the best stopping point often occurs between 10 and 100 training epochs, standard MES was randomly stopped at one of those points. Each method was run with ten different initializations. Table 5.2 shows the results of the comparison. In every dataset but *glass*, MTS shows an equal, or better performance than standard AE training with a far lower variation in AUC.

dataset	MTS-AE	MSE-AE
glass	0.648 ± 0.013	**0.674** ± 0.062
stamps	**0.875** ± 0.018	0.831 ± 0.088
b-cancer	**0.984** ± 0.009	0.951 ± 0.066
pen-global	**0.977** ± 0.008	0.939 ± 0.015
stop-sign	**0.957** ± 0.003	0.275 ± 0.307
speed-sign	**0.860** ± 0.007	0.423 ± 0.249
vowels	**0.872** ± 0.036	**0.872** ± 0.086
letter	**0.843** ± 0.012	0.808 ± 0.047
waveform	**0.523** ± 0.025	**0.523** ± 0.059
satellite	**0.954** ± 0.004	0.885 ± 0.023
pen-local	**0.846** ± 0.025	0.839 ± 0.107
shuttle	**0.993** ± 0.000	0.992 ± 0.002
aloi	**0.558** ± 0.003	0.553 ± 0.008
forest	**0.921** ± 0.001	0.889 ± 0.022

Table 5.2: AUC

5.3 Comparison with Baseline Algorithms

In this section the two proposed methods, UE-KPCA and the MTS-AE, are compared against six popular baselines used in unsupervised anomaly detection across fourteen benchmark datasets. Each baseline is intended to demonstrate a different approach to anomaly detection. The mean AUC and standard deviation for each method and dataset is reported in Table 4.1. Following Goldstein *et al.* [32], the results represent ten runs with varying

parameter settings as described in Section 4.2. Again, the purpose is to represent a random-parameter-selection strategy within the given reasonable interval, which is often used in practice when labels are unavailable. For MD, which does not require any parameters, only one value of AUC is available. For the three largest datasets, only the linear time methods are reported for a narrower comparison.

dataset	UE-KPCA	MTS-AE	k^{th}-NN	LOF	uCBLOF	OC-SVM	iForest	MD
glass	**0.870**	0.649	0.804	0.784	0.761	*0.849*	0.700	0.584
	± 0.000	± 0.013	± 0.038	± 0.029	± 0.038	± 0.028	± 0.016	-
stamps	**0.948**	0.875	*0.922*	0.862	0.903	0.901	0.909	0.896
	± 0.001	± 0.018	± 0.001	± 0.034	± 0.053	± 0.093	± 0.008	-
b-cancer	0.980	**0.985**	0.952	*0.983*	0.980	0.980	0.982	0.954
	± 0.002	± 0.009	± 0.019	± 0.006	± 0.004	± 0.003	± 0.002	-
pen-global	**0.984**	*0.977*	*0.977*	0.837	0.918	0.952	0.922	0.930
	± 0.001	± 0.008	± 0.015	± 0.082	± 0.035	± 0.042	± 0.010	-
stop-sign	*0.954*	**0.957**	0.892	0.913	0.914	0.562	0.886	0.516
	± 0.009	± 0.003	± 0.023	± 0.009	± 0.021	± 0.089	± 0.009	-
speed-sign	0.822	**0.863**	0.669	*0.857*	0.720	0.577	0.688	0.500
	± 0.009	± 0.007	± 0.022	± 0.019	± 0.021	± 0.097	± 0.016	-
vowels	**0.955**	0.872	*0.954*	0.941	0.932	0.923	0.754	0.912
	± 0.000	± 0.036	± 0.008	± 0.004	± 0.023	± 0.017	± 0.031	-
letter	**0.907**	0.844	0.839	*0.875*	0.818	0.687	0.635	0.804
	± 0.004	± 0.012	± 0.023	± 0.026	± 0.019	± 0.193	± 0.018	-
waveform	**0.778**	0.523	*0.749*	0.735	0.737	0.720	0.730	0.574
	± 0.002	± 0.026	± 0.002	± 0.006	± 0.021	± 0.015	± 0.022	-
satellite	**0.973**	0.953	**0.973**	0.815	*0.966*	0.964	0.947	0.914
	± 0.000	± 0.004	± 0.001	± 0.110	± 0.003	± 0.002	± 0.003	-
pen-local	0.953	0.846	*0.976*	**0.987**	0.938	0.953	0.757	0.771
	± 0.001	± 0.025	± 0.007	± 0.002	± 0.021	± 0.013	± 0.026	-
shuttle	0.989	*0.993*	n/a	n/a	0.859	n/a	**0.997**	0.856
	± 0.001	± 0.000	n/a	n/a	± 0.163	n/a	± 0.001	-
aloi	**0.618**	*0.558*	n/a	n/a	0.556	n/a	0.539	0.521
	± 0.002	± 0.003	n/a	n/a	± 0.005	n/a	± 0.004	-
forest	**0.935**	*0.921*	n/a	n/a	0.908	n/a	0.859	0.906
	± 0.000	± 0.001	n/a	n/a	± 0.018	n/a	± 0.012	-

Table 5.3: The AUC under the ROC and std. deviation for each method across all baseline datasets. Higher scores are better. **Bold** indicates the best performing method, while *Italics* indicates the second best performing method

UE-KPCA achieved a high AUC with very low variance across a number of datasets, representing the best or second best method in all but three of the datasets: *cancer*, *pen-local*, and *shuttle*. Notably, UE-KPCA outperformed or tied the most popular kernel-based anomaly

method, the OC-SVM, on every benchmark. The OC-SVM showed a strong sensitivity to ν, which made it less reliable. The OC-SVM also performed very poorly on the high-dimensional *sign* datasets across all parameter settings.

The MTS-AE was less consistent, but performed well on several datasets, particularly those that had a large number of features such as *forest*, *stop-sign*, and *speed-sign*. The MTS-AE generally performed better than iForest, the most commonly used baseline on very high dimensional data. The results show the promise of deep methods in combating the curse-of-dimensionality. Figure 5.8 shows the full ROC curves for the two sign datasets. The shaded region indicates ± 1 standard deviation. The MTS-AE is able to capture far more anomalies at a low FPR. UE-KPCA is able to capture more of the anomalies at a higher FPR, a characteristic shown across a number of the datasets. The full ROC curves for each dataset are presented in Appendix A.

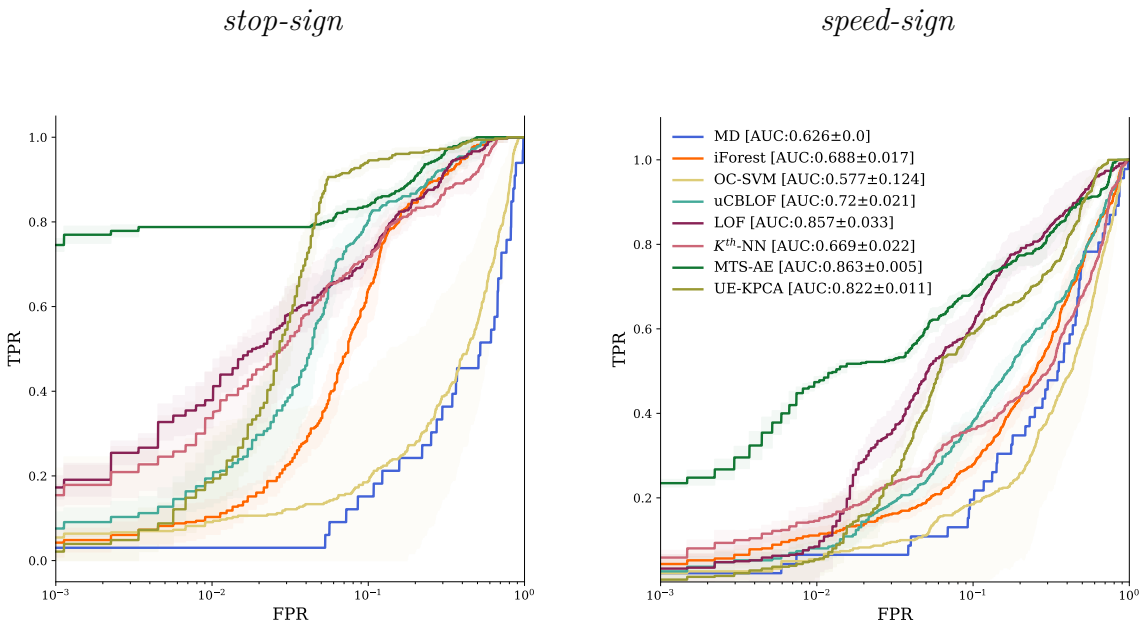


Figure 5.8: ROC Curves comparing different methods on the *stop-sign* and *speed-sign* datasets

Figure 5.9 shows the *most normal* and *most anomalous* examples from the two sign datasets

as ranked by the MTS-AE. A red border indicates a ground truth anomaly label and a green border indicates a normal label. Almost all the examples with the highest anomaly scores are labeled anomalies. The method can identify many different types of anomalies: incorrect cropping, graffiti present, partial obstruction, heavy motion blur, and irregular lighting. Some false positives among the speed limit signs display text rotation or strong highlights. Again, what constitutes an anomaly is difficult to define as there is a subjective element.

Figure 5.9: Rankings produces by the MTS-AE on the *stop-sign* and *speed-sign* datasets. Anomaly scores in each grid increase form left to right and from top to bottom. A red boundary indicates an anomaly, while a green boundary indicates a normal example.

Table 5.4 and Table 5.5 report the mean FPR@95% and AP respectively. A low FPR@95%

and high AP are desirable. The results generally follow the same trends as reflected by the AUC score. Table 5.6 shows the average run time of each method. Both UE-KPCA and MTS-AE hare less efficient when the number of examples are low, but as the dimensionality and cardinality increase, both become more favorable. UE-KPCA is also well suited for parallelization, as each model and evaluation in the ensemble can be calculated independently. Though, MD has unquestionably the fastest run-time, the assumption of linearly-correlated features leads to very poor detection performance on a number of datasets.

dataset	UE-KPCA	MTS-AE	k^{th}-NN	LOF	uCBLOF	OC-SVM	iForest	MD
glass	**0.214**	0.698	0.303	0.537	0.703	0.381	0.503	0.869
stamps	**0.618**	0.740	0.840	0.692	0.844	0.900	0.852	0.961
b-cancer	0.070	0.070	0.185	0.048	0.070	**0.062**	0.070	0.143
pen-global	**0.057**	0.111	0.103	0.608	0.515	0.253	0.206	0.220
stop-sign	**0.107**	0.269	0.550	0.362	0.415	0.900	0.395	0.936
speed-sign	**0.618**	0.740	0.840	0.692	0.844	0.900	0.852	0.961
vowels	**0.110**	0.592	0.137	0.202	0.257	0.280	0.713	0.459
letter	**0.269**	0.426	0.530	0.472	0.536	0.845	0.788	0.665
waveform	0.682	0.928	0.733	0.725	0.766	0.774	**0.647**	0.879
satellite	**0.106**	0.250	0.136	0.875	0.142	0.135	0.336	0.433
pen-local	0.180	0.314	0.092	**0.055**	0.147	0.187	0.479	0.540
shuttle	0.018	0.010	n/a	n/a	0.912	n/a	**0.000**	0.587
aloi	0.928	**0.915**	n/a	n/a	0.931	n/a	0.9306	0.935
forest	0.445	0.447	n/a	n/a	**0.420**	n/a	0.442	0.463

Table 5.4: The FPR at 95% TPR (FPR@95%) for each method across all baseline datasets. Lower scores are better. **Bold** indicates the best performing method.

dataset	UE-KPCA	MTS-AE	k^{th}-NN	LOF	uCBLOF	OC-SVM	iForest	MD
glass	**0.202**	0.081	0.11	0.136	0.142	0.189	0.104	0.080
stamps	0.196	0.150	0.156	**0.234**	0.149	0.176	0.124	0.134
b-cancer	0.677	**0.778**	0.624	0.720	0.683	0.643	0.665	0.430
pen-global	**0.861**	0.846	0.851	0.504	0.815	0.758	0.605	0.558
stop-sign	0.409	**0.811**	0.4619	0.510	0.402	0.136	0.228	0.068
speed-sign	0.181	**0.486**	0.177	0.239	0.146	0.065	0.137	0.034
vowels	**0.455**	0.170	0.449	0.362	0.438	0.412	0.158	0.361
letter	0.324	0.210	0.238	**0.431**	0.230	0.216	0.092	0.226
waveform	0.109	0.035	0.138	0.097	**0.173**	0.067	0.058	0.036
satellite	0.469	0.595	0.601	0.222	0.601	0.548	**0.641**	0.379
pen-local	0.039	0.004	0.061	**0.107**	0.015	0.044	0.003	0.004
shuttle	0.430	0.606	n/a	n/a	0.495	n/a	**0.978**	0.193
aloi	**0.082**	0.038	n/a	n/a	0.046	n/a	0.033	0.037
forest	**0.404**	0.379	n/a	n/a	0.087	n/a	0.039	0.153

Table 5.5: The Average Precision (AP) for each method across all baseline datasets. Higher scores are better. **Bold** indicates the best performing method.

dataset	UE-KPCA	MTS-AE	k^{th}-NN	LOF	uCBLOF	OC-SVM	iForest	MD
glass	0.925	33.7	0.002	0.002	0.069	0.002	0.101	0.001
stamps	0.963	94.2	0.003	0.003	0.075	0.003	0.114	0.001
b-cancer	0.951	172	0.003	0.003	0.165	0.006	0.125	0.001
pen-global	1.145	105	0.015	0.015	0.203	0.018	0.147	0.001
stop-sign	2.83	120.	3.31	3.44	5.04	4.83	1.98	0.545
speed-sign	3.65	286	8.16	8.54	8.07	11.2	3.35	0.645
vowels	1.63	74.9	0.033	0.034	0.256	0.049	0.167	0.001
letter	1.97	150.	0.097	0.097	0.274	0.175	0.194	0.001
waveform	2.62	76.7	0.361	0.362	0.580	0.377	0.267	0.001
satellite	3.82	91.8	0.660	0.660	0.904	1.18	0.446	0.004
pen-local	5.33	113	0.422	0.424	0.966	1.32	0.429	0.003
aloi	40.2	508	n/a	n/a	15.0	n/a	4.08	0.034
shuttle	28.8	35.0	n/a	n/a	3.53	n/a	2.42	0.012
forest	184	139	n/a	n/a	151	n/a	70.3	0.891

Table 5.6: The run time for each method in seconds.

Chapter 6

Conclusions

The goal of unsupervised anomaly detection methods is to identify abnormal examples without the use of labels based only on the intrinsic properties of the data. Unsupervised anomaly detection is critical in areas where labeled data is difficult to obtain, or the pattern of normal and anomalous data changes unpredictably. Our research introduces two novel techniques, Unsupervised Ensemble Kernal Principal Analysis (UE-KPCA) and the Modified Training and Scoring Autoencoder (MTS-AE), that better adapt existing anomaly detection methods to the unsupervised framework.

UE-KPCA features two adaptations to standard KPCA. First, mini-batch sigma tuning allows for a near optimal choice of the Gaussian Kernel parameter, σ, without the use labeled data. Evangelista *et al.* showed that a maximally disperse kernel matrix showed good results for OC-SVMs [27]. Our research shows that the same holds true for KPCA, and uses the inverse, index-of-dispersion as a loss function in order to tune the σ parameter using mini-batch stochastic gradient descent, greatly reducing the computational time required.

Secondly, skeleton ensembles eliminate the cubic training complexity of KPCA. In standard KPCA, an eigendecomposition of the full kernel matrix is required. This is avoided by iterative sampling of the dataset to form a number of much smaller kernel matrices. Projections onto the eigenvectors of these much smaller kernel matrices approximate those of the full evaluation. As with standard KPCA, the reconstruction error in feature space serves as the anomaly scores [43]. By averaging the anomaly scores over the ensemble, UE-KPCA

reliably approaches or exceeds the detection performance of KPCA while drastically reducing the run time on large datasets. Ultimately UE-KPCA, is able to detect anomalies in data with highly non-linear distributions and our empirical evaluation shows that UE-kPCA generally outperforms k-NN, LOF, uCBLOF, iForest, MD, and OC-SVMs.

The other method outlined in this thesis, the MTS-AE, addresses a problem of conventional autoencoders (AE) used for unsupervised anomaly detection. When AEs are trained with anomalies present in the data, the networks are prone to *overgeneralize* and learn to reconstruct the anomalies as well as normal examples. This reduces the ability of AEs to identify abnormal data when using reconstruction error as an anomaly score. To address this shortcoming, the MTS-AE incorporates several novel methods, namely cumulative error scoring (CES), percentile loss (PL), and early stopping via knee detection. CES leverages the history of training errors to better separate anomalous and background points. PL diminishes the influence of anomalies on parameter updates, undermining the ability of AEs to generalize anomalous examples. Lastly, the smooth cumulative loss statistic provides a reliable means of early stopping. The results show a general improvement over the conventional AE as well as a significant increase in performance over other baseline algorithms on high dimensional benchmark datasets.

6.1 Future Work

There are several directions for future work. The framework of UE-KPCA can be potentially used to find anomalies in streaming data. The mini-batch σ tuning presented can be easily trained online with new data to update the parameter. Moreover, the individual models in UE-KPCA can be swapped out over time to evolve the ensemble to a changing pattern of data.

Other future work aims to reliably select other sensitive parameters for AEs and KPCA. For instance in KPCA, the selection of the number of M, still relies on heuristics that may not generalize to all use cases. Another potential shortcoming, sensitivity to extreme global outliers, is a known feature in both linear PCA and KPCA [28, 65]; however, no robust versions of KPCA were used in this analysis. Another possible direction is to combine UE-KPCA with the feature extraction abilities of deep learning models, as others have done with similar traditional methods such as the OC-SVM [5].

Similarly, the AE architectures and hyperparameters are difficult to justify in the fully unsupervised case. There is no consensus on how to reliably set these with validation data. Additionally, the presented version of the MTS-AE does not yield a final trained model that can be applied to unseen new data. Instead, MTS-AE can only be applied to identifying anomalies as part of a full dataset during the act of training. One possible, but costly solution is to retain a history of the previously trained models. Incorporating the training history into a single model instance, perhaps through the use of distillation [42], is an interesting idea for future exploration.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[2] Charu C. Aggarwal and Saket Sathe. Theoretical foundations and algorithms for outlier ensembles. *SIGKDD Explor. Newsl.*, 17(1):24–47, September 2015. ISSN 1931-0145. doi: 10.1145/2830544.2830549. URL https://doi.org/10.1145/2830544.2830549.

[3] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. In Jan Van den Bussche and Victor Vianu, editors, *Database Theory — ICDT 2001*, pages 420–434, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-44503-6.

[4] Mennatallah Amer and Markus Goldstein. Nearest-neighbor and clustering based anomaly detection algorithms for rapidminer. 08 2012. doi: 10.5455/ijavms.141.

[5] Jerone Andrews, Edward Morton, and Lewis Griffin. Detecting anomalous data using

auto-encoders. *International Journal of Machine Learning and Computing*, 6:21, 01 2016.

[6] C. M. Bachmann, T. L. Ainsworth, and R. A. Fusina. Exploiting manifold geometry in hyperspectral imagery. *IEEE Trans. on Geoscience and Remote Sensing*, 43(3): 441–454, 2005.

[7] Laura Beggel, Michael Pfeiffer, and Bernd Bischl. Robust anomaly detection in images using adversarial autoencoders. 01 2019.

[8] Y. Bengio, J.-F. Paiement, P. Vincent, O. Delalleau, N. Le Roux, and M. Ouimet. Out-of-sample extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering. In *Advances in Neural Information Processing Systems*, volume 16. The MIT Press, Cambridge, MA, USA, 2004.

[9] A. Bounsiar and M. G. Madden. One-class support vector machines revisited. In *2014 International Conference on Information Science Applications (ICISA)*, pages 1–4, May 2014. doi: 10.1109/ICISA.2014.6847442.

[10] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, page 93–104, New York, NY, USA, 2000. Association for Computing Machinery. ISBN 1581132174. doi: 10.1145/342009.335388. URL https://doi.org/10.1145/342009.335388.

[11] A. Budynkov and S. Masolkin. The problem of choosing the kernel for one-class support vector machines. *Automation and Remote Control*, 78:138–145, 01 2017. doi: 10.1134/S0005117917010118.

[12] Guilherme O. Campos, Arthur Zimek, Jörg Sander, Ricardo J. G. B. Campello,

Barbora Micenková, Erich Schubert, Ira Assent, and Michael E. Houle. On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. *Data Mining and Knowledge Discovery*, 30(4):891–927, Jul 2016. ISSN 1573-756X. doi: 10.1007/s10618-015-0444-8. URL https://doi.org/10.1007/s10618-015-0444-8.

[13] Raghavendra Chalapathy and Sanjay Chawla. Deep learning for anomaly detection: A survey. *CoRR*, abs/1901.03407, 2019. URL http://arxiv.org/abs/1901.03407.

[14] Raghavendra Chalapathy, Aditya Krishna Menon, and Sanjay Chawla. Robust, deep and inductive anomaly detection. *CoRR*, abs/1704.06743, 2017. URL http://arxiv.org/abs/1704.06743.

[15] Raghavendra Chalapathy, Aditya Krishna Menon, and Sanjay Chawla. Anomaly detection using one-class neural networks. *CoRR*, abs/1802.06360, 2018. URL http://arxiv.org/abs/1802.06360.

[16] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), July 2009. ISSN 0360-0300. doi: 10.1145/1541880.1541882. URL https://doi.org/10.1145/1541880.1541882.

[17] O. Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19(5):1155–1178, May 2007. ISSN 0899-7667. doi: 10.1162/neco.2007.19.5.1155.

[18] François Chollet et al. Keras. https://keras.io, 2015.

[19] Ronald R. Coifman and Stéphane Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5 – 30, 2006. ISSN 1063-5203. doi: https://doi.org/10.1016/j.acha.2006.04.006. URL http://www.sciencedirect.com/science/article/pii/S1063520306000546. Special Issue: Diffusion Maps and Wavelets.

[20] D. Cozzolino and L. Verdoliva. Single-image splicing localization through autoencoder-based anomaly detection. In *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 1–6, Dec 2016. doi: 10.1109/WIFS.2016.7823921.

[21] Tal Daniel, Thanard Kurutach, and Aviv Tamar. Deep variational semi-supervised novelty detection, 2019.

[22] Dario D'Avino, Davide Cozzolino, Giovanni Poggi, and Luisa Verdoliva. Autoencoder with recurrent neural networks for video forgery detection. *CoRR*, abs/1708.08754, 2017. URL http://arxiv.org/abs/1708.08754.

[23] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, page 233–240, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933832. doi: 10.1145/1143844.1143874. URL https://doi.org/10.1145/1143844.1143874.

[24] Taylor Denouden, Rick Salay, Krzysztof Czarnecki, Vahdat Abdelzad, Buu Phan, and Sachin Vernekar. Improving reconstruction autoencoder out-of-distribution detection with mahalanobis distance, 12 2018.

[25] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[26] G. Enderlein. Hawkins, d. m.: Identification of outliers. chapman and hall, london – new york 1980, 188 s., £ 14, 50. *Biometrical Journal*, 29(2):198–198, 1987. doi: 10.1002/bimj.4710290215. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/bimj.4710290215.

[27] Paul Evangelista, M. Embrechts, and Boleslaw Szymanski. Some properties of the

gaussian kernel for one class learning. pages 269–278, 09 2007. doi: 10.1007/978-3-540-74690-4_28.

[28] Jicong Fan and Tommy W. S. Chow. Exactly robust kernel principal component analysis. *CoRR*, abs/1802.10558, 2018. URL http://arxiv.org/abs/1802.10558.

[29] Tom Fawcett. Introduction to roc analysis. *Pattern Recognition Letters*, 27:861–874, 06 2006. doi: 10.1016/j.patrec.2005.10.010.

[30] Markus Goldstein and Andreas Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. 09 2012.

[31] Markus Goldstein and Seiichi Uchida. Behavior analysis using unsupervised anomaly detection. 2014.

[32] Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLoS ONE*, Apr 2016. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0152173.

[33] Dong Gong, Lingqiao Liu, Vuong Le, Budhaditya Saha, Moussa Reda Mansour, Svetha Venkatesh, and Anton van den Hengel. Memorizing normality to detect anomaly: Memory-augmented deep autoencoder for unsupervised anomaly detection. *CoRR*, abs/1904.02639, 2019. URL http://arxiv.org/abs/1904.02639.

[34] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[35] J. Guo, G. Liu, Y. Zuo, and J. Wu. An anomaly detection framework based on autoencoder and nearest neighbor. In *2018 15th International Conference on Service*

Systems and Service Management (ICSSSM), pages 1–6, July 2018. doi: 10.1109/ICSSSM.2018.8464983.

[36] Simon Günter, Nicol Schraudolph, and S. Vishwanathan. Fast iterative kernel principal component analysis. *Journal of Machine Learning Research*, 8:1893–1918, 08 2007.

[37] Fredrik Hallgren and P. Northrop. Incremental kernel pca and the nyström method. 01 2018.

[38] Sahand Hariri, Matias Carrasco Kind, and Robert J. Brunner. Extended isolation forest. *CoRR*, abs/1811.02141, 2018. URL http://arxiv.org/abs/1811.02141.

[39] Mahmudul Hasan, Jonghyun Choi, Jan Neumann, Amit K. Roy-Chowdhury, and Larry S. Davis. Learning temporal regularity in video sequences. *CoRR*, abs/1604.04574, 2016. URL http://arxiv.org/abs/1604.04574.

[40] Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster based local outliers. *Pattern Recognition Letters*, 24:1641–1650, 06 2003. doi: 10.1016/S0167-8655(03)00003-5.

[41] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *Proceedings of International Conference on Learning Representations*, 2017.

[42] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[43] Heiko Hoffmann. Kernel pca for novelty detection. *Pattern Recognition*, 40(3):863 – 874, 2007. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2006.07.009. URL http://www.sciencedirect.com/science/article/pii/S0031320306003414.

[44] Wen Jin, Anthony K. H. Tung, Jiawei Han, and Wei Wang. Ranking outliers using symmetric neighborhood relationship. In Wee-Keong Ng, Masaru Kitsuregawa, Jianzhong Li, and Kuiyu Chang, editors, *Advances in Knowledge Discovery and Data Mining*, pages 577–593, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33207-7.

[45] Ian Jolliffe. *Principal Component Analysis*, pages 1094–1096. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-04898-2. doi: 10.1007/978-3-642-04898-2_455. URL https://doi.org/10.1007/978-3-642-04898-2_455.

[46] Fabian Keller, Emmanuel Muller, and Klemens Bohm. Hics: High contrast subspaces for density-based outlier ranking. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, ICDE '12, page 1037–1048, USA, 2012. IEEE Computer Society. ISBN 9780769547473. doi: 10.1109/ICDE.2012.88. URL https://doi.org/10.1109/ICDE.2012.88.

[47] Kwang Kim, Matthias Franz, and Bernhard Schölkopf. Iterative kernel principal component analysis for image modeling. *IEEE transactions on pattern analysis and machine intelligence*, 27:1351–66, 10 2005. doi: 10.1109/TPAMI.2005.181.

[48] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Outlier detection in axis-parallel subspaces of high dimensional data. In Thanaruk Theeramunkong, Boonserm Kijsirikul, Nick Cercone, and Tu-Bao Ho, editors, *Advances in Knowledge Discovery and Data Mining*, pages 831–838, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-01307-2.

[49] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Outlier detection in axis-parallel subspaces of high dimensional data. In Thanaruk Theeramunkong, Boonserm Kijsirikul, Nick Cercone, and Tu-Bao Ho, editors, *Advances in Knowledge*

Discovery and Data Mining, pages 831–838, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-01307-2.

[50] Hans-Peter Kriegel, Peer Kröger, Erich Schubert, and Arthur Zimek. Loop: Local outlier probabilities. pages 1649–1652, 01 2009. doi: 10.1145/1645953.1646195.

[51] Roland Kwitt and Ulrich Hofmann. Unsupervised anomaly detection in network traffic by means of robust pca. *2007 International Multi-Conference on Computing in the Global Information Technology (ICCGI'07)*, pages 37–37, 2007.

[52] Donghwoon Kwon, Hyunjoo Kim, Jinoh Kim, Sang Suh, Ikkyun Kim, and Kuinam Kim. A survey of deep learning-based network anomaly detection. *Cluster Computing*, 22, 01 2019. doi: 10.1007/s10586-017-1117-8.

[53] S. Lafon, Y. Keller, and R. R. Coifman. Data fusion and multicue data matching by diffusion maps. *IEEE Transactions on pattern analysis and machine intelligence*, 28 (11):1784–1797, 2006.

[54] Christoph H. Lampert. Kernel methods in computer vision. *Foundations and Trends® in Computer Graphics and Vision*, 4(3):193–285, 2009. ISSN 1572-2740. doi: 10.1561/0600000027. URL http://dx.doi.org/10.1561/0600000027.

[55] Yingwei Li, Yi Li, and Nuno Vasconcelos. Resound: Towards action recognition without representation bias. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 520–535, Cham, 2018. Springer International Publishing. ISBN 978-3-030-01231-1.

[56] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 413–

422, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3502-9. doi: 10.1109/ICDM.2008.17. URL https://doi.org/10.1109/ICDM.2008.17.

[57] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press. URL https://projecteuclid.org/euclid.bsmsp/1200512992.

[58] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

[59] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, USA, 2008. ISBN 0521865719.

[60] Federico Di Mattia, Paolo Galeone, Michele De Simoni, and Emanuele Ghelfi. A survey on gans for anomaly detection. *CoRR*, abs/1906.11632, 2019. URL http://arxiv.org/abs/1906.11632.

[61] Kishan Mehrotra, Chilukuri Mohan, and HuaMing Huang. *Clustering-Based Anomaly Detection Approaches*, pages 41–55. 10 2017. ISBN 978-3-319-67524-4. doi: 10.1007/978-3-319-67526-8_4.

[62] R. T. Meinhold, C. C. Olson, and T. Doster. Kernel PCA for anomaly detection in hyperspectral images using spectral-spatial fusion. In Miguel Velez-Reyes and David W. Messinger, editors, *Algorithms and Technologies for Multispectral, Hyperspectral, and Ultraspectral Imagery XXIV*, volume 10644, pages 601 – 608. International Society for Optics and Photonics, SPIE, 2018. doi: 10.1117/12.2306359. URL https://doi.org/10.1117/12.2306359.

[63] Manpreet Singh Minhas and John S. Zelek. Anomaly detection in images. *CoRR*, abs/1905.13147, 2019. URL http://arxiv.org/abs/1905.13147.

[64] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. *CoRR*, abs/1802.09089, 2018. URL http://arxiv.org/abs/1802.09089.

[65] Minh Hoai Nguyen and Fernando De la Torre. Robust kernel principal component analysis. In *Advances in Neural Information Processing Systems*. 2009.

[66] C. C. Olson and T Doster. A parametric study of unsupervised anomaly detection performance in maritime imagery using manifold learning techniques. In *SPIE Defense+ Security*, pages 984016–984016. International Society for Optics and Photonics, 2016.

[67] C. C. Olson and T. Doster. A novel detection paradigm and its comparison to statistical and kernel-based anomaly detection algorithms for hyperspectral imagery. In *Proc. CVPRW*, pages 302–308. IEEE, 2017.

[68] C.C. Olson, K.P. Judd, and J.M. Nichols. Manifold learning techniques for unsupervised anomaly detection. *Expert Systems with Applications*, 91:374 – 385, 2018. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2017.08.005. URL http://www.sciencedirect.com/science/article/pii/S0957417417305328.

[69] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip Gibbons, and Christos Faloutsos. Loci: Fast outlier detection using the local correlation integral. pages 315–326, 01 2003. doi: 10.1109/ICDE.2003.1260802.

[70] Zhimin Peng, Prudhvi Gurram, Heesung Kwon, and Wotao Yin. Sparse kernel learning-based feature selection for anomaly detection. *IEEE Transactions on Aerospace and Electronic Systems*, 51:1698–1716, 2015.

[71] Mark Pijnenburg and Wojtek Kowalczyk. *Extending an Anomaly Detection Benchmark with Auto-encoders, Isolation Forests, and RBMs*, pages 498–515. 10 2019. ISBN 978-3-030-30274-0. doi: 10.1007/978-3-030-30275-7_39.

[72] Foster Provost, Tom Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. *Proceedings of the Fifteenth International Conference on Machine Learning*, 04 2001.

[73] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, page 427–438, New York, NY, USA, 2000. Association for Computing Machinery. ISBN 1581132174. doi: 10.1145/342009.335437. URL https://doi.org/10.1145/342009.335437.

[74] Daniel Ramotsoela, Adnan Abu-Mahfouz, and Gerhard Hancke. A survey of anomaly detection in industrial wireless sensor networks with critical water system infrastructure as a case study. *Sensors*, 2018:2491, 08 2018. doi: 10.3390/s18082491.

[75] Shebuti Rayana. Outlier detection datasets: ODDS, 2016. URL http://odds.cs.stonybrook.edu.

[76] I. S. Reed and X. Yu. Adaptive multiple-band cfar detection of an optical pattern with unknown spectral distribution. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(10):1760–1770, Oct 1990. ISSN 0096-3518. doi: 10.1109/29.60107.

[77] Manassés Ribeiro, André Eugênio Lazzaretti, and Heitor Silvério Lopes. A study of deep convolutional auto-encoders for anomaly detection in videos. *Pattern Recognition Letters*, 105:13 – 22, 2018. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2017.07.016. URL http://www.sciencedirect.com/science/article/pii/S0167865517302489. Machine Learning and Applications in Artificial Intelligence.

[78] Lee J. Rickard, Robert W. Basedow, Edward F. Zalewski, Peter R. Silverglate, and Mark Landers. HYDICE: an airborne system for hyperspectral imaging. In Gregg Vane, editor, *Imaging Spectrometry of the Terrestrial Environment*, volume 1937, pages 173 – 179. International Society for Optics and Photonics, SPIE, 1993. doi: 10.1117/12.157055. URL https://doi.org/10.1117/12.157055.

[79] Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956. ISSN 00034851. URL http://www.jstor.org/stable/2237390.

[80] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. Deep one-class classification. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4393–4402, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/ruff18a.html.

[81] Alaa Sagheer and Mostafa Kotb. Unsupervised pre-training of a deep lstm-based stacked autoencoder for multivariate time series forecasting problems. *Scientific Reports*, 9:19038, 12 2019. doi: 10.1038/s41598-019-55320-6.

[82] Anshuman Sahu, George Runger, and Daniel Apley. Image denoising with a multiphase kernel principal component approach and an ensemble version. pages 1–7, 10 2011. doi: 10.1109/AIPR.2011.6176339.

[83] Hamed Sarvari, Carlotta Domeniconi, Bardh Prenkaj, and Giovanni Stilo. Unsupervised boosting-based autoencoder ensembles for outlier detection, 2019.

[84] Saket Sathe and Charu C. Aggarwal. Lodes: Local density meets spectral outlier detection. In *SDM*, 2016.

[85] V. Satopaa, J. Albrecht, D. Irwin, and B. Raghavan. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pages 166–171, June 2011. doi: 10.1109/ICDCSW.2011.20.

[86] Bernhard Schölkopf, John C. Platt, John C. Shawe-Taylor, Alex J. Smola, and Robert C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Comput.*, 13(7):1443–1471, July 2001. ISSN 0899-7667. doi: 10.1162/089976601750264965. URL https://doi.org/10.1162/089976601750264965.

[87] B. Schölkopf, A. Smola, and K. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, July 1998. doi: 10.1162/089976698300017467.

[88] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, USA, 2004. ISBN 0521813972.

[89] Vít Skvára, Tomás Pevný, and Václav Smídl. Are generative deep models for novelty detection truly better? *CoRR*, abs/1807.05027, 2018. URL http://arxiv.org/abs/1807.05027.

[90] Michael R. Smith, Tony Martinez, and Christophe Giraud-Carrier. An instance level analysis of data complexity. *Mach. Learn.*, 95(2):225–256, May 2014. ISSN 0885-6125. doi: 10.1007/s10994-013-5422-z. URL https://doi.org/10.1007/s10994-013-5422-z.

[91] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, pages 1453–1460, 2011.

[92] Jian Tang, Zhixiang Chen, Ada Wai-chee Fu, and David W. Cheung. Enhancing effectiveness of outlier detections for low density patterns. In Ming-Syan Chen, Philip S. Yu, and Bing Liu, editors, *Advances in Knowledge Discovery and Data Mining*, pages 535–548, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-47887-4.

[93] David Tax and Robert Duin. Support vector data description. *Machine Learning*, 54: 45–66, 01 2004. doi: 10.1023/B:MACH.0000008084.60811.49.

[94] David M. J. Tax and Robert P. W. Duin. Support vector domain description. *Pattern Recognition Letters*, 20:1191–1199, 1999.

[95] H. Wang, M. J. Bah, and M. Hammad. Progress in outlier detection techniques: A survey. *IEEE Access*, 7:107964–108000, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2932769.

[96] Wei Wang, Zheng Dang, Yinlin Hu, Pascal Fua, and Mathieu Salzmann. Backpropagation-friendly eigendecomposition. *CoRR*, abs/1906.09023, 2019. URL http://arxiv.org/abs/1906.09023.

[97] Liang Xiong, Barnabás Póczos, and Jeff Schneider. Group anomaly detection using flexible genre models. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 1071–1079, Red Hook, NY, USA, 2011. Curran Associates Inc. ISBN 9781618395993.

[98] Xiaodan Xu, Huawen Liu, and Minghai Yao. Recent progress of anomaly detection. *Complexity*, 2019:1–11, 01 2019. doi: 10.1155/2019/2686378.

[99] Kai Zhang, Ivor W. Tsang, and James T. Kwok. Improved nyström low-rank approximation and error analysis. In *Proceedings of the 25th International Conference on*

Machine Learning, ICML '08, page 1232–1239, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605582054. doi: 10.1145/1390156.1390311. URL https://doi.org/10.1145/1390156.1390311.

[100] Yiru Zhao, Bing Deng, Chen Shen, Yao Liu, Hongtao Lu, and Xian-Sheng Hua. Spatio-temporal autoencoder for video anomaly detection. In *Proceedings of the 25th ACM International Conference on Multimedia*, MM '17, page 1933–1941, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349062. doi: 10.1145/3123266.3123451. URL https://doi.org/10.1145/3123266.3123451.

[101] Yue Zhao and Maciej Hryniewicki. Xgbod: Improving supervised outlier detection with unsupervised representation learning. pages 1–8, 07 2018. doi: 10.1109/IJCNN.2018.8489605.

[102] Yue Zhao, Zain Nasrullah, and Zheng Li. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019. URL http://jmlr.org/papers/v20/19-011.html.

[103] Alice Zheng and Amanda Casari. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O'Reilly Media, Inc., 1st edition, 2018. ISBN 1491953241.

[104] Xiangxin Zhu, Carl Vondrick, Charless C. Fowlkes, and Deva Ramanan. Do we need more training data? *CoRR*, abs/1503.01508, 2015. URL http://arxiv.org/abs/1503.01508.

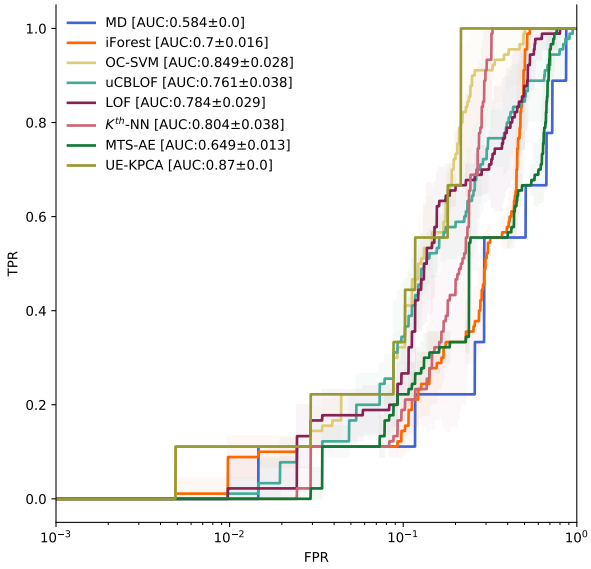
[105] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. A survey on unsupervised outlier detection in high-dimensional numerical data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 5(5):363–387, 2012. doi: 10.1002/sam.11161. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sam.11161.

[106] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Dae ki Cho, and Haifeng Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *ICLR*, 2018.

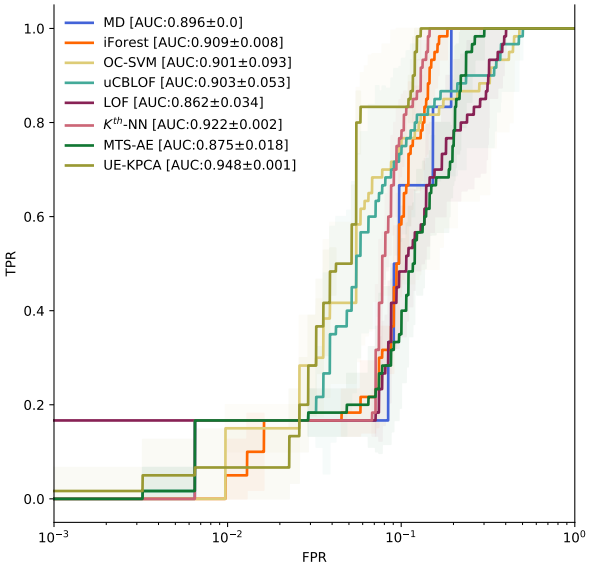
Appendices

Appendix A

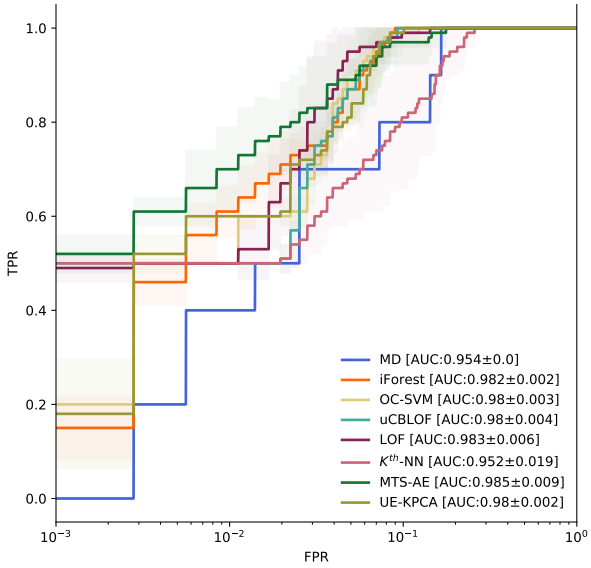
ROC Curves



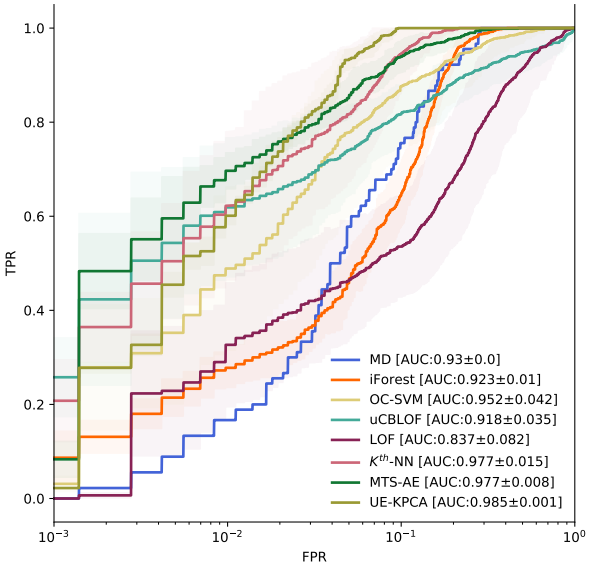
a) glass



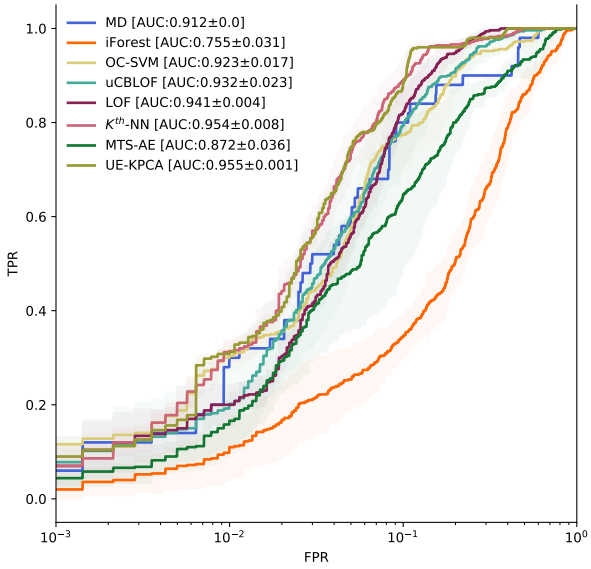
b) stamps



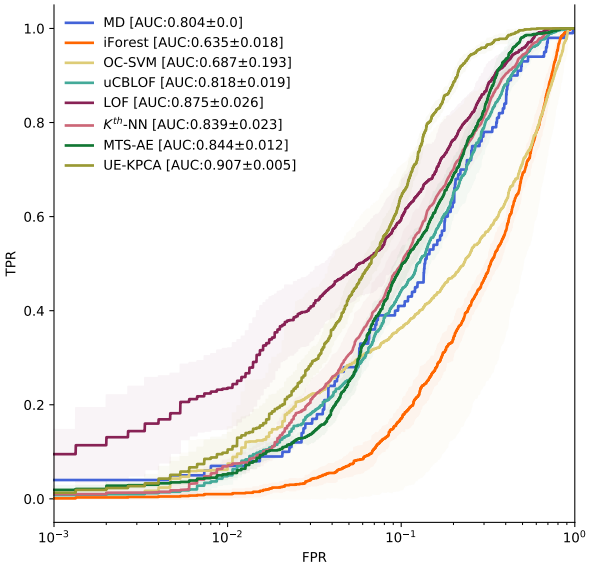
c) b-cancer



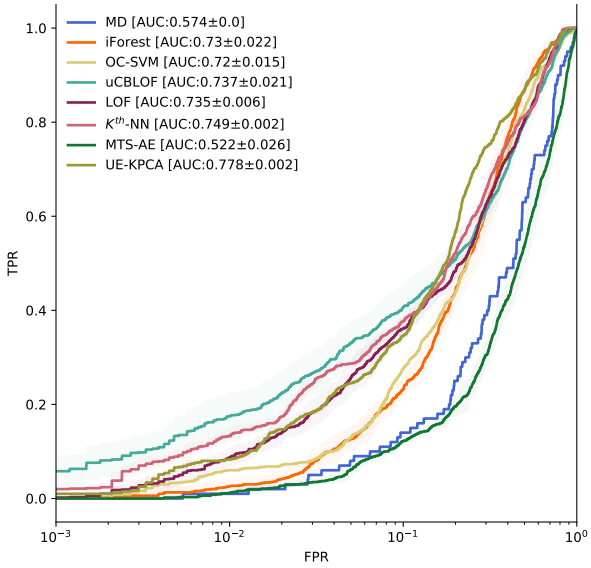
d) pen-global



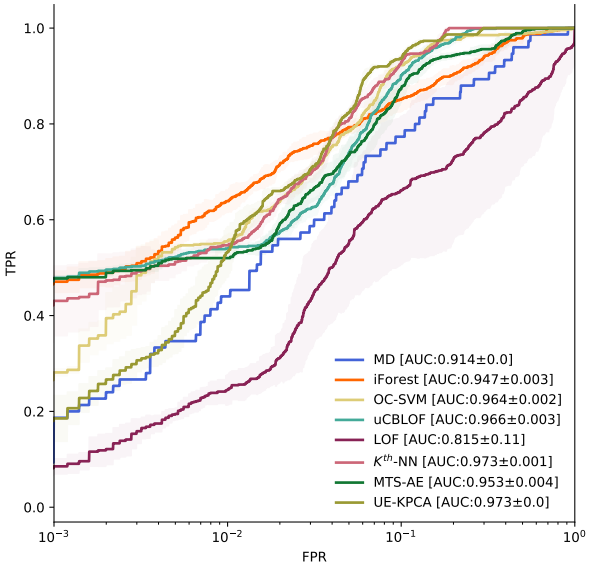
e) vowels



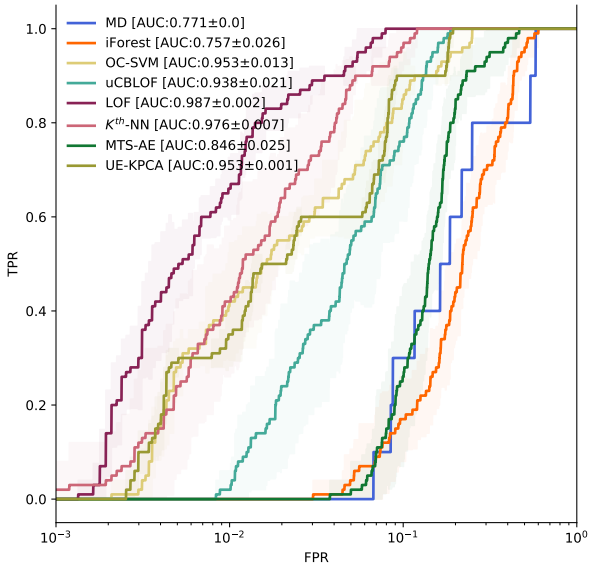
f) letter



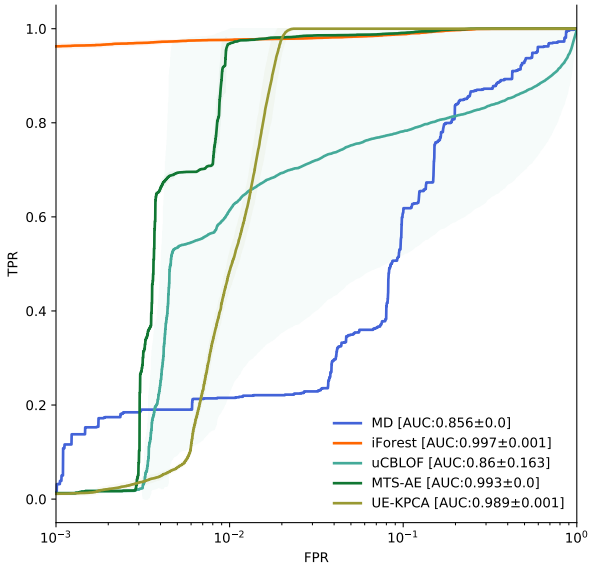
g) waveform



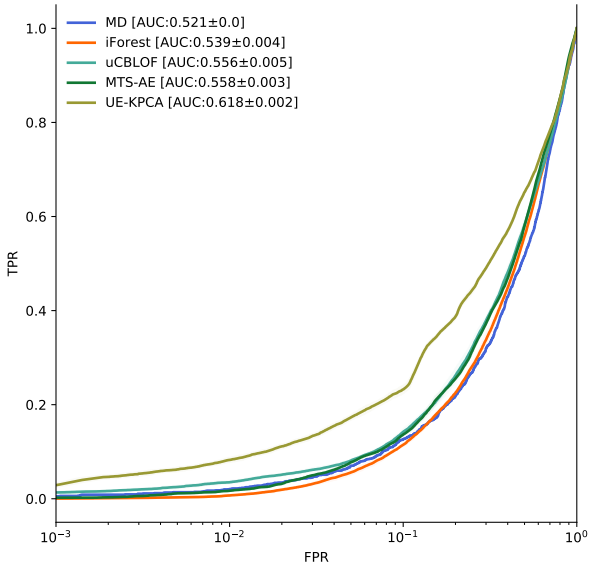
h) satellite



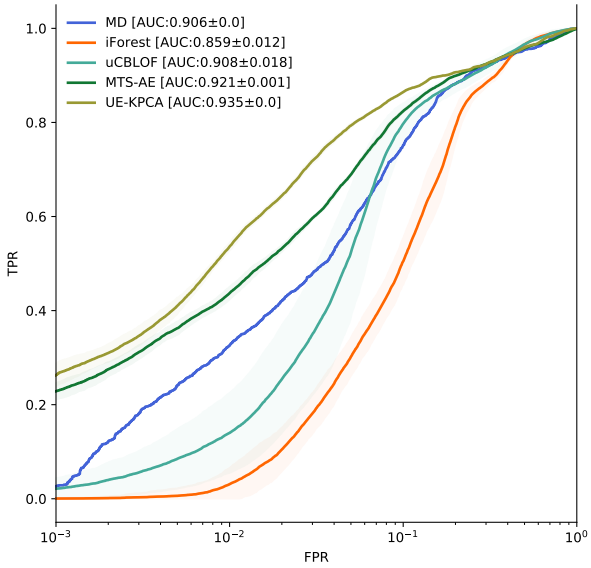
i) pen-local



j) shuttle



k) aloi



l) forest

Figure A.1: ROC Curves showing the detection performance of each method. FPRs and TPRs are interpolated to form an average curve. The shaded area indicates ± 1 std. deviation.

Appendix B

Python Code

This appendix chapter provides the Python code for each of the proposed methods. Section B.1 includes the code for the UE-KPCA and KPCA methods and Section B.1 provides the code for the MTS-AE.

B.1 UE-KPCA Code

```
import numpy as np
from scipy.stats import pearsonr
import warnings
import keras.backend as kb
from tensorflow import keras
import tensorflow as tf
import gc
import numpy as np
from scipy.linalg import eigh
import warnings

tf.compat.v1.disable_eager_execution()
```

```
class kPCA:
    """
    author: nmerrill@ut.edu

    Calculate the principle components in feature
    space and return the reconstruction errors.

    The implementation is based on Heiko Hoffman.
    http://www.heikohoffmann.de/kpca.html

    Parameters
    ----------
    kernel : string, optional (default='rbf')
         Specifies the kernel type to be used in the algorithm.
         It must be one of 'poly' or 'rbf'
         If none is given, 'rbf' will be used.

    sigma : float, optional (default=1)
        Kernel coefficient for 'rbf'

    order : int, optional (default=3)
        Kernel degree for 'poly'

    q : int, optional (default='same')
        number of retained eigenvectors (alphas)
        for the reconstruction error
```

```
        If q is 'auto' then n_features
        will be used instead. For a 'linear
        kernel this will result in zero reconstruction error

    sample_pct : float, optional (default: 1.0)
        What percentage of the data to use
        to determine the principle components

    block_size : int, optional (default: 500)
        The number of examples per batch during testing

    contamination : float in (0., 0.5), optional (default=0.1)
        The amount of contamination of the data set,
        i.e. the proportion of outliers in the data set.
        Used when fitting to define the threshold
        on the decision function.

    verbose : bool, (default: False)
        Prints out runtime and feedback



    Attributes
    ----------

    model_X_S : numpy array of shape (n_sub,d_features)
```

```
        sample of X used for calulating alphas, typically
        the full sample is used so that X_S = X.
        Sampling set by sample_pct

    model_K_mat : numpy array of shape (n_sub,sub)
        uncentered gram matrix after training

    model_alphas : numpy array of shape (n_samples, q)
        retained eigen values after training

    decision_scores_ : numpy array of shape (n_samples,)
        The outlier scores of the training data.
        The higher, the more abnormal.
        Outliers tend to have higher scores.
        This value is available once the detector is fitted.

    threshold_ : float
        The threshold is based on ``contamination``. It is the
        ``n_samples * contamination`` most abnormal samples in
        ``decision_scores_``. The threshold is calculated for generating
        binary outlier labels.

    labels_ : int, either 0 or 1
        The binary labels of the training data. 0 stands for inliers
        and 1 for outliers/anomalies. It is generated by applying
        ``threshold_`` on ``decision_scores_``.
```

```
    """

    def __init__(self, kernel='rbf', order = 3, q = 'same', sigma = 1.0,
                 sample_pct = 1.0, shrinking=True, block_size = 500,
                 contamination = 0.1, verbose=False):
        self.kernel = kernel
        self.sigma = sigma
        self.gamma = 1/2/sigma/sigma
        self.order = order
        self.q = q
        self.sample_pct = sample_pct
        self.block_size = block_size
        if contamination <= 0 or contamination > 0.5:
                warnings.warn("Contamination must be between (0,0.5)\
                              contamination set to 0.1")
                self.contamination = 0.1
        else:
            self.contamination = contamination
        self.verbose = verbose


    def subsample_data(self,X, sample_pct = None):
        """Returns X_S a random subsample of X".

        Parameters
        ----------
```

```
        X : numpy array of shape (n_samples, d_features)
            The input samples.
        sample_pct: float, percentage (0,1) to sample from X
        """

        if sample_pct == None:
              n_s = self.n_bag #use class default if none is specified
        else:
            assert not (sample_pct <= 0 or sample_pct > 1),\
            "Sampling must be between (0,1)"
            n_s = int(sample_pct * X.shape[0])

        idx = np.random.permutation(X.shape[0])
        X_s = X[idx[:n_s]]

        return X_s


    def sqrd_euclid(self, X,X_S):
        """Returns the (n_S x n) matrix of pairwise distances.

        Parameters
        ----------
        X : numpy array of shape (n, d_features)
            The input samples.
```

```
        X_S : numpy array of shape (n_S, d_features)
            The input samples.

        params: float, kernel parameter, defaults to sigma or order if None
        """
        a = \
        np.expand_dims(np.diag(X_S.dot(X_S.T)),\
                       axis = 1).dot(np.ones((1, X.shape[0])))
        b = \
        np.ones((X_S.shape[0],\
                 1)).dot(np.expand_dims(np.diag(X.dot(X.T)),axis = 0))

        sqrd_dists = a + b - 2*X_S.dot(X.T)

        return sqrd_dists


    def gramMatrix(self,X, X_S, kernel = None, params = None):
        """Returns the (n x n) gram matrix based on the kernel.

        Parameters
        ----------
        X : numpy array of shape (n_samples, d_features)
            The input samples.

        kernel: string,  {'rbf, 'poly'}
```

```
        kernel function to use options

    params: float, kernel parameter, defaults to sigma or order if None
    """

    #use class default if none is specified
    if kernel == None:
        kernel = self.kernel

    sqrd_dists = self.sqrd_euclid(X,X_S)

    if kernel == 'rbf':
        if params == None:
            params = self.gamma
        K = np.exp(-params*sqrd_dists)
    elif kernel == 'poly':
        if params == None:
            params = self.order
        K = np.power((sqrd_dists +1),params)
    else:
        assert False, "Specify an available kernel {'rbf, 'poly'}"

    return K


def eigenDecomp_gramMatrix(self, K_centered, numev = None):
```

```
        """Returns the leading q number of
        eigenvectors from the eigendecomposition
        of the centered gram matrix

        Parameters
        ----------
        K : numpy array of shape (n_samples,n_samples)
            centered gram matrix

        numev: int, number of eigenvectors (alphas) to retain
        """
        K_centered = (K_centered + K_centered.T) / 2


        if numev == None:
            numev = self.q

        w_, v = eigh(K_centered)
        w_ = w_.reshape(-1,1)
        w_ = np.flipud(w_)
        v = np.fliplr(v)
        alphas = v[:,:numev]

        #Each column is an eigen vector
        lambdas = w_[:numev]
        #from biggest to smallest
```

```
        alphs = alphas*np.squeeze(1/np.sqrt(lambdas))

        return alphs,lambdas


    def calc_reconstructionErrors(self, X,  X_S, K_mat, alphs):
        """Returns the reconstruction error projecting onto alphas.

        Parameters
        ----------
        X : numpy array of shape (n_samples, d_features)
            The input samples.

        alphs: numpy array of shape (n_samples, q)
            eigenvectors of centered gram matrix


        X_S : numpy array of shape (n_subsamples, d_features),
        optional (default=None)
            Subsampling of the data.
            If none, the full kernel is used for projection
        """
        n_samples, d_features = X.shape
        n_sub = K_mat.shape[0]

        numev = alphs.shape[1]
```

```
        #helper calcs
        Krow = K_mat.sum(axis = 0)/n_sub #not normed!
        Ksum = (Krow).sum()/n_sub
        sumalphs = np.ones(n_sub).dot(alphs)

        reconstruction_errs = np.zeros(n_samples)

        for block_i in (range(0,n_samples,self.block_size)):

            X_block = X[block_i:block_i+self.block_size,:]
            n_block = X_block.shape[0]

            k_L = self.gramMatrix(X_block, X_S)

            f_L = \
            np.dot(k_L.T,alphs) - (sumalphs*np.ones((n_block,numev)) * \
            np.expand_dims((np.sum(k_L,axis=0).T/n_sub - Ksum),axis=1)) \
            -  np.ones((n_block, numev))*np.dot(Krow,alphs)

            d_p =  ( 1 - (2*np.sum(k_L,axis = 0)/n_sub).T + Ksum )

            f_L_ = np.diag(np.dot(f_L,f_L.T))
            errs_block = (d_p - f_L_)

            reconstruction_errs[block_i:block_i+self.block_size] = errs_block
```

```
        return  reconstruction_errs

    def threshold(self,scores,contamination):
        """Fit detector. y is optional for unsupervised methods.

        Parameters
        ----------
        scores : numpy array of shape (n_samples, d_features)
        The input samples.

        contamination : float in (0., 0.5), optional (default=0.1)
            The amount of contamination of the data set,
            i.e. the proportion of outliers in the data set.
            Used when fitting to define the threshold
            on the decision function.
        """
        threshold_ = np.quantile(scores,1-contamination)
        labels = np.ones(scores.shape[0])
        labels[scores < threshold_] = 0

        self.threshold_ = threshold_

        return labels
```

```
    def fit(self, X, y=None):
        """Fit detector. y is optional for unsupervised methods.

        Parameters
        ----------
        X : numpy array of shape (n_samples, d_features)
        The input samples.

        y : numpy array of shape (n_samples,), optional (default=None)
        The ground truth of the input samples (labels).
        """
        n_samples, d_features = np.shape(X)
        self.d_features = d_features

        if self.q == 'same':
            self.q = d_features

        if self.sample_pct < 1:
            X_S = self.subsample_data(X)
        else:
            X_S = X

        K_mat = self.gramMatrix(X_S,X_S)

        n_sub = K_mat.shape[0]
```

```
one_n = np.ones((n_sub,n_sub)) / n_sub
K_centered = K_mat - one_n.dot(K_mat - K_mat.dot(one_n)) \
+ one_n.dot(K_mat).dot(one_n)
K_centered = (K_centered + K_centered.T) / 2

if self.verbose:
    print("Computed gram matrix")

alphs,lambdas = self.eigenDecomp_gramMatrix(K_centered)

if self.verbose:
        print("Computed alphas","\n")

self.model_alphas = alphs
self.model_lambdas = lambdas
self.model_X_S = X_S
self.model_K_mat = K_mat

reconstruction_errs = self.calc_reconstructionErrors(X,
                                                     X_S,
                                                     K_mat,
                                                     alphs)

self.decision_scores_ = reconstruction_errs
```

```
        self.labels_ = self.threshold(self.decision_scores_,
                                      self.contamination)

        return self


    def decision_function(self, X_test):
        """predict anomaly scores (reconstruction error) using model \
        gram matrix and alphas

        Parameters
        ----------
        X_test : numpy array of shape (n_test_samples, d_features)
        The test samples.
        """
        # correct dimension if a single example is given
        if X_test.ndim == 1:
            X_test = np.expand_dims(X_test,axis = 0)

        scores = \
        self.calc_reconstructionErrors(X_test,
                                       self.model_X_S,
                                       self.model_K_mat,
                                       self.model_alphas)

        return scores
```

```
    def predict(self,X_test,threshold = None):
        """predict anomaly label (reconstruction error)
        using model gram matrix and alphas

        Parameters
        ----------
        X_test : numpy array of shape (n_test_samples, d_features)
        The test samples.

        threshold: float, optional, default to
        threshold calculated by .fit()
        """

        # correct dimension if a single example is given
        if X_test.ndim == 1:
            X_test = np.expand_dims(X_test,axis = 0)

        if threshold == None:
            threshold = self.threshold_

        scores = \
        self.calc_reconstructionErrors(X_test,
                                       self.model_X_S,
                                       self.model_K_mat,
```

```
                                        self.model_alphas)

        labels = np.ones(scores.shape[0])
        labels[scores < threshold] = 0

        return labels


class UEKPCA:
    """
    Parameters
    ----------
    initial_sigma : float, optional (default=1e-3)
        Initialize the Kernel coefficient for 'rbf'

    q : int, optional (default='auto')
        number of retained eigenvectors (alphas)
        for the reconstruction error
        If q is 'auto' then n_features
        will be used instead up to a maximum of
        75.

    learning_rate: float, optional (default = 0.005)
        learning rate for gradient descent

    n_bag : int, optional (default: 100)
```

```
        The number of examples used to construt
        the kernel matrix in for sigma tuning

    n_model : int, optional (default: 100)
        Number of kPCA models in the ensemble

    n_skel : int, optional (default: 256)
        The number of examples used to build
        the kernel matrix for finding alphas
        in each model of the ensemble

    batch_size : int, optional (default: 1)
        Number of batches per training step

    contamination : float in (0., 0.5), optional (default=0.1)
        The amount of contamination of the data set,
        i.e. the proportion of outliers in the data set.
        Used when fitting to define the threshold
        on the decision function.

    patience : int, optional (default:1000)
        number of training steps to wait for
        a change in sigma before deciding
        convergence and early stopping

    verbose : bool,  optional (default: False)
```

```
        Prints out runtime and feedback

    max_steps : int, optional (default: -1)
        maximum numer of steps for sigma tuning. -1, places no limit

    weighted_ensemble_avg : bool, optional (default: False)
        Uses a weighted averaging, rather than
        simple averaging for ensemble scores

    initial_sigma : float, optional (default: 1.0)
        Starting point for sigma tuning

    contamination : float in (0., 0.5), optional (default=0.1)
        The amount of contamination of the data set,
        i.e. the proportion of outliers in the data set.
        Used when fitting to define the threshold on
        the decision function.



    Attributes
    ----------
    sigma: float
        The sigma parameter in the rbf kernel

    sigma_hist: list of floats with len = training steps
        A history of sigma for each training step
```

```
    loss_list: list of floats with len = training steps
        A history of losses for each training step

    model_X_S : numpy array of shape (n_sub,d_features)
        sample of X used for calulating alphas, typically
        the full sample is used so that X_S = X.
        Sampling set by sample_pct

    model_K_mat : numpy array of shape (n_sub,sub)
        uncentered gram matrix after training

    model_alphas : numpy array of shape (n_samples, q)
        retained eigen values after training

    decision_scores_ : numpy array of shape (n_samples,)
        The outlier scores of the training data.
        The higher, the more abnormal.
        Outliers tend to have higher scores.
        This value is available once the detector is fitted.

    threshold_ : float
        The threshold is based on ``contamination``. It is the
        ``n_samples * contamination`` most abnormal samples in
        ``decision_scores_``. The threshold is calculated
        for generating binary outlier labels.
```

```
    labels_ : int, either 0 or 1
        The binary labels of the training data. 0 stands
        for inliers and 1 for outliers/anomalies.
        It is generated by applying ``threshold_``
        on ``decision_scores_``.
    """

    def __init__(self,
                 q = 'same',
                 initial_sigma = 1,
                 shrinking=True,
                 sigma = 'auto',
                 n_bag= 100,
                 n_skel = 256,
                 n_models = 100,
                 contamination = 0.1,
                 patience = 1000,
                 batch_size = 1,
                 verbose = True,
                 learning_rate = 0.005,
                 max_steps = -1,
                 weighted_ensemble_avg = False
                 ):

        self.initial_sigma = initial_sigma
```

```
        self.n_skel = n_skel
        self.batch_size = batch_size
        self.q = q
        self.n_bag = n_bag
        self.n_skel = n_skel
        self.verbose = verbose
        self.patience = patience
        self.learning_rate = learning_rate
        self.max_steps = max_steps
        self.n_models = n_models
        self.weighted_ensemble_avg = weighted_ensemble_avg
        self.n_iter = 0
        self.sigma = sigma


        if contamination <= 0 or contamination > 0.5:
            assert False
        else:
            self.contamination = contamination


    def check_settings(self,X):
        """checks the parameter settings and initializes
        self.n_samples and self.d_features

        Parameters
```

```
        ----------
        X : numpy array of shape (n_samples, d_features)
            The input samples.

        """
        self.n_samples,self.d_features = np.shape(X)

        assert X.max() < 1.01 and X.min() > -0.01, \
        "Norm data [0,1]"

        if self.q == 'same':
            self.q = min(self.d_features,75)


        if self.verbose:
           print("n_samples:" ,self.n_samples, "d_features:", \
                 self.d_features, "bag size:", self.n_bag,\
                 "skel size:", self.n_skel, "numev:", self.q)

        return self


    def threshold(self,scores,contamination):
        """Fit detector. y is optional for unsupervised methods.

        Parameters
```

```
        ----------
        scores : numpy array of shape (n_samples, d_features)
        The input samples.

        contamination : float in (0., 0.5), optional (default=0.1)
            The amount of contamination of the data set,
            i.e. the proportion of outliers in the data set.
            Used when fitting to define the threshold
            on the decision function.
        """
        threshold_ = np.quantile(scores,1-contamination)
        labels = np.ones(scores.shape[0])
        labels[scores < threshold_] = 0

        self.threshold_ = threshold_
        return labels


    def subsample_data(self,X, n_s = None):
        """Returns X_S a random subsample of X".

        Parameters
        ----------
        X : numpy array of shape (n_samples, d_features)
            The input samples.
        sample_pct: float, percentage (0,1) to sample from X
```

```
        """

        if n_s == None:
            n_s = self.n_bag #use class default if none is specified

        idx = np.random.permutation(X.shape[0])
        X_s = X[idx[:n_s]]

        return X_s


    def sqrd_euclid(self, X,X_S):
        """Returns the (n_S x n) matrix of pairwise distances.

        Parameters
        ----------
        X : numpy array of shape (n, d_features)
            The input samples.

        X_S : numpy array of shape (n_S, d_features)
            The input samples.

        params: float, kernel parameter, defaults to sigma or order if None
        """
        a = \
        np.expand_dims(np.diag(X_S.dot(X_S.T)),\
```

```
                    axis = 1).dot(np.ones((1, X.shape[0])))
        b = \
        np.ones((X_S.shape[0],\
                 1)).dot(np.expand_dims(np.diag(X.dot(X.T)),axis = 0))

        sqrd_dists = a + b - 2*X_S.dot(X.T)

        return sqrd_dists


    def UE_kPCA_Loss(self,sd_mat_offDiag):
        """Returns the inverse index of dispersion
        of the off diagonal kernel parameters
         Parameters
        ----------
        sd_mat_offDiag :tensor of shape
        (batch_size, ((n_bag^2)-n_bag))
            The input sub_samples.
        """

        def loss(y_true,y_pred):

            sigma = kb.mean(y_pred,axis = 0)
            gamma = 1/(2*sigma*sigma)

            K_offDiag = kb.exp(-gamma*sd_mat_offDiag)
```

```
            m1 = kb.mean(K_offDiag)
            s2 = kb.mean(kb.square(K_offDiag-m1))

            # small eps to prevent div by zero
            inv_iD = m1/(s2+1e-20)

            return inv_iD
        return loss


    def sigma_model(self):
        """Create a model of a single unit that
        is trained by SGD to optimize sigma
        """
        #feeding zeros, needed to fit Keras framework
        dummy_input = keras.layers.Input(shape=(1,))

        def custom_activation(x):

            #A custom activation is necessary to bound the
            #output of sigma between approx 0 and inf
            #otherwise you will see errors if the sigma switch signs.

            return tf.math.log(1+tf.math.exp(x))

        #based on inverse of activation
```

```
        initial_bias = np.log(np.exp(self.initial_sigma)-1)
        sigma_output = \
        keras.layers.Dense(int(1),activation= custom_activation,\
            bias_initializer=\
            keras.initializers.Constant(initial_bias))(dummy_input)

        sd_mat_offDiag = \
        keras.layers.Input(shape=(self.n_bag*(self.n_bag-1))//2)
        sigmaModel = \
        keras.models.Model([dummy_input,sd_mat_offDiag], sigma_output)

        sigmaModel.compile(optimizer = \
                           keras.optimizers.RMSprop(self.learning_rate),#, clipvalue=1)
                           loss = self.UE_kPCA_Loss(sd_mat_offDiag))

        return sigmaModel


    def optimize_sigma(self, X):
        """Returns the optimized sigma by perfomring batch gradient descent

        Parameters
        ----------
        X: numpy array of shape (n_samples, d_features)
            The input data set
```

```
        """

        self.check_settings(X)#setting 'auto' and checks

        self.current_sigma = self.initial_sigma

        #Generate the model
        Model = self.sigma_model()

        self.sigma_hist = [self.initial_sigma]
        self.loss_hist = []

        #initialize main loop
        self.step_i = 0
        last_update = 0

        self.max_steps = -1

        best_loss = 1e10
        sigma_list = []

        #Main Sigma training loop
        while self.step_i < self.max_steps or self.max_steps == -1:


            X_batch = \
```

```
np.zeros((self.batch_size,(self.n_bag*(self.n_bag-1))//2))
for j in range(self.batch_size):
    X_s = self.subsample_data(X)

    sd_mat = self.sqrd_euclid(X_s,X_s)


    sd_mat_offDiag = sd_mat[np.triu_indices(self.n_bag,k=1)]
    X_batch[j,:] = sd_mat_offDiag


sigma_loss = \
Model.train_on_batch([np.zeros((self.batch_size ,1)), X_batch],\
                     np.zeros((self.batch_size ,1)))

bias = Model.layers[-1].get_weights()[1][0]
self.current_sigma = np.log(1+np.exp(bias))


if len(sigma_list) < self.patience:
    sigma_list.append(self.current_sigma)
else:
    sigma_list += [sigma_list.pop(0)]
    sigma_list[-1] = self.current_sigma
```

```
            if sigma_loss < best_loss:
                best_loss = sigma_loss
                self.low_sigma = self.current_sigma
                last_update = 0
            else:
                last_update += 1



            if last_update > self.patience:
                if best_loss > 10:
                    last_update = 0
                    warnings.warn("Did not converge.")
                else:
                    self.sigma = np.asarray(sigma_list.copy()).mean()


                    return self.sigma


            if self.verbose:
                print(f'Step:{self.step_i} Sigma:{self.current_sigma:.4f} \
                      Loss:{sigma_loss:.3f} Patience:{last_update}')


            self.step_i +=1

        #trainging stopped at max steps
        return self.sigma
```

```
def make_skel_model(self,X_):

    n_s = self.n_skel
    X_s = self.subsample_data(X_, n_s = n_s)

    skel_model = kPCA(
            sigma = self.sigma,
            q = self.q,
            verbose = self.verbose,
            contamination = self.contamination,
            )
    skel_model.fit(X_s)

    scores = skel_model.decision_function(X_)

    return scores,skel_model


def fit(self, X, y=None):
    """Fit detector. y is optional for unsupervised methods.

    Parameters
    ----------
    X : numpy array of shape (n_samples, d_features)
```

```
        The input samples.

        y : numpy array of shape (n_samples,), optional (default=None)
        The ground truth of the input samples (labels).
        """

        self.check_settings(X)

        if self.sigma == 'auto':
            self.sigma = self.optimize_sigma(X)
            gc.collect()

        self.ensemble = []
        self.ensemble_scores = np.zeros((self.n_samples,self.n_models))

        for n in range(self.n_models):

            scores, skel_model = self.make_skel_model(X)
            self.ensemble.append(skel_model)
            self.ensemble_scores[:,n] = scores

        if self.weighted_ensemble_avg:
            pseudo_GT = self.ensemble_scores.mean(axis=1)
            self.detector_weights = np.zeros(self.n_models)
            for n in range(self.n_models):
                self.detector_weights[n] = \
```

```
                    pearsonr(pseudo_GT, self.ensemble_scores[:,n])[0]
            self.decision_scores_ = \
                (self.detector_weights*self.ensemble_scores).mean(axis=1)
        else:
            self.decision_scores_ = \
                (self.ensemble_scores).mean(axis=1)

        self.labels_ = \
            self.threshold(self.decision_scores_,self.contamination)

        return self.decision_scores_


    def threshold(self,scores,contamination):
        """calculate the threshold based on
        contamination and scores.

        Parameters
        ----------
        scores : numpy array of shape (n_samples, d_features)
        The input samples.

        contamination : float in (0., 0.5), optional (default=0.1)
            The amount of contamination of the data set,
            i.e. the proportion of outliers in the data set.
            Used when fitting to define the threshold
```

```
            on the decision function.
        """
        threshold_ = np.quantile(scores,1-contamination)
        labels = np.ones(scores.shape[0])
        labels[scores < threshold_] = 0

        self.threshold_ = threshold_

        return labels


    def predictE(self, X_test, threshold = None):
        """predicting labels based on the ensemble of models

        Parameters
        ----------
        X_test : numpy array of shape (n_test_samples, d_features)
        The test samples.
        """

        # correct dimension if a single example is given
        if X_test.ndim == 1:
            X_test = np.expand_dims(X_test,axis = 0)

        scores = np.zeros(X_test.shape[0])
        for m in self.ensemble:
```

```
            scores += m.decision_function(X_test)

        scores = scores/len(self.ensemble)

        if threshold == None:
            threshold = self.threshold_

        labels = np.ones(scores.shape[0])
        labels[scores < threshold_] = 0

        return labels


    def decision_functionE(self,X_test,threshold = None):
        """predict anomaly scores (reconstruction error)
        using the ensemble of models

        Parameters
        ----------
        X_test : numpy array of shape (n_test_samples, d_features)
        The test samples.

        threshold: float, optional, default
        to threshold calculated by .fit()
        """
         # correct dimension if a single example is given
```

```
        if X_test.ndim == 1:
            X_test = np.expand_dims(X_test,axis = 0)

        if threshold == None:
            threshold = self.threshold_

        scores = np.zeros(X_test.shape[0])
        for m in self.ensemble:
            scores += m.decision_function(X_test)

        scores = scores/len(self.ensemble)

        return scores
```

B.2 MTS-AE Code

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.optimizers import *
import tensorflow_probability as tfp
import tensorflow.keras.backend as kb
from kneed import KneeLocator
from sklearn import metrics
import gc
import warnings
```

```
warnings.filterwarnings("ignore", message="No knee/elbow found")

class MTS_AE:
    """
    Unsupervised Autoencoder training class

    Parameters
    ----------

    verbose : bool,  optional (default: False)
        Prints out training information and AUC

    batch_size: int, optional (default: 256)
        number of examples per batch

    learning_rate: float, optional (default 0.001)
        learning rate for optimizer

    max_epochs: int, optional (default: 500)
        maximum number of epochs to wait before
        terminating training

    percentile_loss: bool, optional (default: True)
        Use percentile loss for training so
        that mini-batch updates are only performed
```

```
    on examples below a threshold

percentile_loss_quantile : float [0,100],
    optional (default: 95. 'percent')
    threshold for percentile Loss

M_knee_stop : float optional (default: 5)
    early stopping parameter, if a knee
    is detected and the current epoch
    exceed M_knee_stop * current epoch,
    training is halted

full_hist : bool optional (default: False)
    store a history of training anomaly scores

CES : bool optional (default: True)
    Use cumulative error scoring or not

pretrain: bool optional (default: True)
    Number of epochs to use MSE
    loss before PL loss


Attributes
----------
KNEE : int
```

```
        epoch that the knee is detected

    err_hist : list (N, num_epochs)
        if full_hist = True, then store
        MSE of each example at the end
        of each epoch

    sum_err_hist : list [N, num_epochs]
        if full_hist = True, then store
        unormalized CES of each example
        at the end of each epoch

    hist_auc_base :  list (num_epochs, )
        if mask = GT labels, then store
        the MSE AUC at the end of each epoch

    hist_auc_sum :  list (num_epochs, )
        if mask = GT labels, then store
        the CES AUC at the end of each epoch

    hist_loss : list (num_epochs)
        MSE loss at the end of each epoch

    hist_mean_err_sum : list (num_epochs)
        normalized (div by epoch) CES
        loss at the end of each epoch
```

```
    decision_scores_ : numpy array of shape (n_samples,)
        The outlier scores of the training data.
        The higher, the more abnormal.
        Outliers tend to have higher scores.
        This value is available once
        the detector is fitted.

    """




    def __init__(self,
                 model,
                 max_epochs = 500,
                 batch_size = 256,
                 verbose = True,
                 learning_rate = 0.001,
                 percentile_loss = True,
                 CES = True,
                 percentile_loss_quantile = 95.0,
                 steps_per_epoch = 200,
                 M_knee_stop = 5,
                 full_hist = False,
                 pretrain = 10
```

```
            ):
        self.model = model
        self.verbose = verbose
        self.batch_size = batch_size
        self.learning_rate = learning_rate
        self.max_epochs = max_epochs
        self.percentile_loss = percentile_loss
        self.percentile_loss_quantile = percentile_loss_quantile
        self.M_knee_stop = M_knee_stop
        self.full_hist = full_hist
        self.steps_per_epoch = steps_per_epoch
        self.CES = CES
        self.pretrain = pretrain


    def subsample_data(self,X, n_s = None):
        """Returns X_S a random subsample of X".

        Parameters
        ----------
        X : numpy array of shape (n_samples, d_features)
            The input samples.
        sample_pct: float, percentage (0,1) to sample from X
        """

        if n_s == None:
```

```
            n_s =self.batch_size


        idx = np.random.permutation(X.shape[0])
        X_s = X[idx[:n_s]]


        return X_s



    def compile_model(self,model,PL,learning_rate):
        """
        Compliles a model using either PL or MSE

        Parameters
        ----------
        model : tensorflow or keras model to compile

        PL : bool
        comiles with PL if True, else MSE

        learning_rate : float
        learning rate for Adam Optimizer
        """

        outshape = kb.int_shape(model.outputs[0])
        L = np.prod(outshape[1:])
```

```
def Percentile_Loss(y_true,y_pred):

    #needed for reshaping batch for boolean mask
    true = tf.reshape(y_true, [-1,L])
    pred = tf.reshape(y_pred, [-1,L])

    LOSS = tf.keras.losses.mse(
            true,pred,)

    #calculate the q-percentile error
    r = tfp.stats.percentile(LOSS,
                             q=self.percentile_loss_quantile)

    #remove examples above percentile
    iLT = tf.math.logical_not(tf.math.greater(LOSS,r))

    #use loss only of examples below threshold
    LOSS_LT = tf.boolean_mask(LOSS,iLT,)

    LOSS_LT = kb.mean(LOSS_LT)

    return LOSS_LT


def Base_Loss(y_true,y_pred):
    LOSS = tf.keras.losses.mse(
```

```
                    y_true,y_pred,)

            return LOSS

        opt = Adam(learning_rate,clipnorm=1.0)
        if PL:
            model.compile(loss = Percentile_Loss , optimizer=opt)
            if self.verbose:
                print('Using percentile loss')
        else:
            model.compile(loss = Base_Loss , optimizer=opt)

        return model


    def fit(self, data, mask = None):
        """
        Trains the autoencoder model

        Parameters
        ----------
        data : A (N x D) data matrix
        mask : GT labels for reporting AUC during training
        """

        #get the dimensionality of the data, used for MSE calc
```

```
self.data_shape = data.shape
self.N = self.data_shape[0]
if len(self.data_shape) > 2:
    W,H,C = self.data_shape[1:]

# compiles the model
self.model = self.compile_model(self.model,
                                 False,
                                 self.learning_rate)

#initialize list for training
self.hist_loss = []
self.hist_mean_err_sum = []
self.err_sum = np.zeros(self.N)
loss_sum = np.zeros(self.N)

if mask is not None:
    self.hist_auc_base = []
    self.hist_auc_sum = []
self.epoch = 1
self.Exit = False
if self.full_hist:
    self.err_hist = []
    self.sum_err_hist = []

#Upsample the dataset if necessary
```

```
        X_train = []
        if self.steps_per_epoch * self.batch_size > self.N:
            mult = (self.steps_per_epoch *self.batch_size)//self.N + 1
            for i in range(mult):
                X_train.append(data)

            X_train = np.vstack(X_train)
        else:
            X_train = data


        #begin the main trianing loop
        while self.epoch <= self.max_epochs and self.Exit == False:

            #shuffle each epoch
            X_train = X_train[np.random.permutation(X_train.shape[0])]

            #train on a number of steps
            self.model.fit(X_train,
                           X_train,
                           verbose=self.verbose,
                           batch_size = self.batch_size,
                           epochs=1)

            #predict on every example
            preds = self.model.predict(data,
```

```
                                batch_size = self.batch_size)

        #reshape for CNN models using image data
        if len(self.data_shape) > 2:
            self.err =(np.square(data.reshape(self.N,\
            int(W*H*C))-preds.reshape(self.N,int(W*H*C)))).mean(axis=1)
        else:
            self.err = np.square(data-preds).mean(axis=1)

        #get epoch loss
        loss = np.mean(self.err)

        #store
        loss_sum += self.err.copy()
        if self.epoch >= self.pretrain:
            self.err_sum += self.err.copy()
            if self.epoch == self.pretrain:
                self.model = self.compile_model(self.model,
                                                 self.percentile_loss,
                                                 self.learning_rate)

        #get current anomaly score
        if self.CES:
            self.decision_scores_ = self.err_sum/self.epoch
        else:
            self.decision_scores_ = self.err
```

```
lossM = np.mean(loss_sum.copy())/(self.epoch)

self.hist_loss.append(loss.copy())
self.hist_mean_err_sum.append(lossM.copy())

if self.full_hist:
    self.err_hist.append(self.err)
    self.sum_err_hist.append(loss_sum.copy())

if mask is not None:
        auc_base = metrics.roc_auc_score(mask,
                                         self.err)
        auc_sum = metrics.roc_auc_score(mask,
                                        self.err_sum/self.epoch)
        self.hist_auc_base.append(auc_base)
        self.hist_auc_sum.append(auc_sum)

# search for a knee in the CES Loss
if self.epoch >= 10 and self.M_knee_stop > 0:
    kn = KneeLocator(np.arange(self.epoch),
                     np.asarray(self.hist_mean_err_sum),
                     S=5, curve='convex',
                     direction='decreasing',
                     online=True)
    self.KNEE = kn.knee
```

```
        if self.KNEE != None and self.KNEE >= 5:
            if self.verbose:
                print('Knee:',self.KNEE)
            if self.epoch >= self.KNEE * self.M_knee_stop:
                self.Exit = True

    self.epoch += 1

    #clear cache
    gc.collect()

    if self.verbose and mask is not None:
        print(f"Epoch: {self.epoch} \t \
              Standard AUC: {round(auc_base,3)} \
              \t CES AUC: {round(auc_sum,3)} \t \
              PL bool: {self.percentile_loss}")
    else:
        print(f"Epoch: {self.epoch}")
```