Anomaly Detection in Labeled and Unlabeled Datasets using Supervised and Unsupervised Algorithms

Anjena Daswani
Dept. of Mech. Eng
BU: U49162792
anjenad@bu.edu

Priscila Rubio
Dept. of Mech. Eng
BU: U44040224
prubio@bu.edu

Kshitij Duraphe
Dept. of Elec and Comp. Eng
BU: U60743361
kshitijd@bu.edu

Kashyap Panda
Dept. of Mech. Eng
BU: U40571904
kpanda@bu.edu

Abstract

In this paper, we compare the performance of semi-supervised, supervised, and unsupervised anomaly detection algorithms, specifically Autoencoders, One-Class Support Vector Machines (OCSVMs), K-Nearest Neighbors (KNN), Unsupervised KNN, and Isolation Forest, on both labeled and unlabeled datasets. For a comprehensive analysis, we generate Receiver Operating Characteristic (ROC) curves for all datasets by adding labels to the unlabeled data and report the Area Under the Curve (AUC) values. Our findings indicate that Autoencoders and Isolation Forests exhibit moderate performance on pre-labeled data but underperform on arbitrarily labeled data. Conversely, OCSVMs and KNN demonstrate better performance on arbitrarily labeled data but are less effective on pre-labeled data.

1 Introduction

Anomaly detection is a technique used in data analysis to identify rare or unusual data points, patterns, or events that deviate significantly from the norm or expected behavior [1]. Anomaly detection is typically used to detect outliers or anomalies in large datasets, where the data is often complex and multi-dimensional, making it difficult to manually identify anomalies. The goal of anomaly detection is to identify and isolate these unusual patterns or data points, which can be caused by errors in data collection, system malfunctions, fraud, or other abnormal events. Once identified, these anomalies can be analyzed further to understand their causes and take appropriate actions to correct them or prevent similar events in the future. While it can be relatively easy for humans to point out unusual patterns, it is relatively difficult for machines due to the different types of anomalies that can occur. Anomalies can be of various types, for the purpose if our task, we particularly focus on time-series anomaly and point anomaly.

2 Background

Anomaly detection has a long history dating back to the early days of statistics and data analysis. The earliest approaches to anomaly detection were based on simple statistical methods, such as the use of mean and standard deviation to identify data points that were significantly different from the average of the dataset. These methods were limited in their ability to detect complex anomalies in high-dimensional and non-linear datasets. A famous failure of these methods occurred when the reputed astronomer Edwin Hubble posited his namesake law. Even though new observations at the same suggested that some galaxies moved relative to the Earth at speeds not proportional to their distance to it, Edwin Hubble stated that the rate of change of speed was linear. Even though all the observations available at the time said that Hubble was wrong, he turned out to be right as observing tools became more and more powerful.

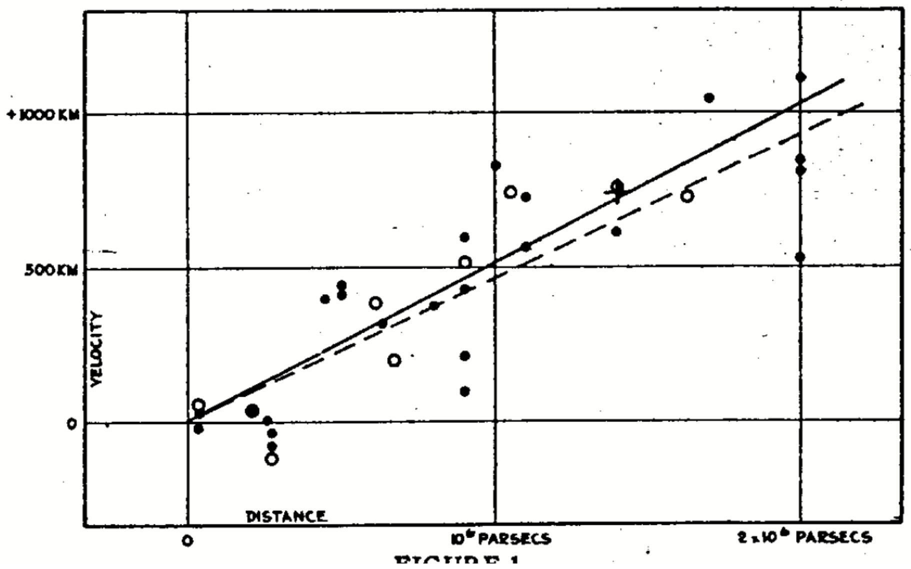

Figure 1: Observations supporting Hubble's law. Initially, only the bottom left points were observed, leading to linear fits being declared an anomaly.

As data analysis techniques evolved, more sophisticated statistical methods were developed for anomaly detection, including the use of distribution fitting, clustering, and principal component analysis (PCA). These techniques aimed to capture the underlying structure of the data and identify anomalies based on their deviation from this structure. However, these methods were still limited by their reliance on assumptions about the distribution and structure of the data [1].

Over time, machine learning algorithms began to be applied to anomaly detection. These methods showed promising results in detecting anomalies in large and complex datasets and were able to learn patterns and relationships in the data without requiring strong assumptions about the data distribution. However, these methods were computationally expensive and required significant amounts of training data.

More recently, deep learning-based approaches have been developed for anomaly detection, including variational autoencoders and different neural networks. These methods leverage the power of deep neural networks to learn complex representations of the data and identify anomalies based on their deviation from this representation. These methods have shown promising results in detecting anomalies in various datasets and are capable of handling high-dimensional and non-linear data.

3 Methodology

We have tested four different algorithms on two datasets. The first three algorithms are *Autoencoders*, *Isolation Trees*, and *OCSVMs*. Autoencoders are unsupervised, but training is done by comparing predicted anomalies to real anomalies on validation datasets to fine-tune the model. Isolation Trees and OCSVMs are fully unsupervised The final algorithm, *K-NN*, is split into supervised and unsupervised algorithms.

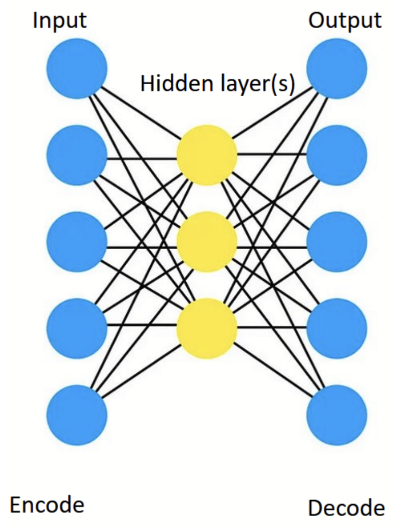

Figure 2: A high-level overview of the functioning of an autoencoder

3.1 Autoencoders

An autoencoder is a specialized variant of a neural network architecture that is primarily built to encode the input data into a compressed and meaningful representation [2]. The main purpose of this representation is to extract the most important features from the input data, while minimizing the loss of information.

After the encoding process, the autoencoder proceeds to decode the compressed representation back into the original input data. This reconstruction process is optimized to produce an output that is as similar as possible to the original input data. The autoencoder network is trained to reduce the reconstruction error between the original and the decoded output.

The concept of autoencoders was first introduced in [3] as an attempt to train a neural network to learn 'information' about the data in an unsupervised manner.

Formally, an autoencoder can be defined as an algorithm that learns the functions $E : R^n \to R^d$ (the *encoder*) and $D : R^d \to R^n$ (the *decoder*), that satisfy

$$\text{arg min}_{E,D} E_{loss}[\Delta(\boldsymbol{x}, D \cdot E(\boldsymbol{x})] \tag{1}$$

where $E_{loss}[\Delta]$ is just the expectation over the distribution of x, and the loss function Δ is the so-called *reconstruction loss* [2]. Δ measures the difference between the output of the decoder and the input of the encoder. Generally, this difference is set to be the l_2 norm. The autoencoder is visualized below:

One of the most popular ways to implement autoencoders is with neural networks. In this case, E and D are neural networks, which can be linear or non-linear. If both E and D are linear, then the algorithm would effectively function the same way as Principal Component Analysis (PCA) (more formally, the latent representation of the autoencoder and the PCA algorithm would be the same). Therefore, non-linear neural networks distinguish an autoencoder from PCA, and hence autoencoders can broadly be called as a generalization of PCA. Instead of compressing data into low dimensions, autoencoders can learn a non-linear manifold.

Autoencoders can be trained layer-by-layer or end-to-end. There are advantages and disadvantages to training autoencoders in either way. On one hand, training end-to-end would train a 'deeper' autoencoder. On the other hand, training this way would significantly increase the computational time required.

An obvious solution for E and D is just the identity operator, which would map the data back to itself. Therefore, autoencoders are generally used to compress data and decompress it. This step achieves not only data compression, but can also be used for feature extraction and other forms of analysis. However, one must be careful to not *overfit* the data. This is possible even if the data is compressed to one dimension, if E and D are capable to mapping and unmapping each element to and from an index.

Autoencoders can also be used for anomaly detection. The assumption behind the reasoning is that once trained, decoding 'normal' data would result in a significantly lower reconstruction error compared to decoding anomalous data. This also implies that the autoencoder learns the latent space of 'normal' samples. Therefore, autoencoders could potentially be applied in cases where it is important to detect anomalies such as bank fraud detection and system monitoring.

We have trained an autoencoder with a small regularizer on the bank dataset using gradient descent. The parameters we have used are in the following table:

Parameter	**Description**	**Value/Range**
m	Number of points	41188
d	Dimensions per point	62
h	Hidden layer size (Dimension size to compress to)	10
λ	Regularization Parameter	0.001
ϵ	Parameter to initialize weights between	0.1
num_iters	Number of iterations	1000
α	Gradient descent parameter	[0.1,1]

The non-linear encoding function is the sigmoid function:

$$s(x) \doteq \frac{1}{1+e^{-x}} \tag{2}$$

The backpropagation (see [5]) step requires the derivative of the sigmoid, and it is easy to see that

$$\frac{ds}{dx} = s(x)(1 - s(x)) \tag{3}$$

For the sake of reducing computational time, there is only one hidden layer, trained using gradient descent according to the number of iterations. The weight matrices that encode and decode data are randomly initialized between $-\epsilon$ and ϵ. The algorithm is as follows:

Algorithm 1 Autoencoder

```
Require: Input data $X_{train}$, number of iterations $num\_iters$, learning rate $\alpha$, regularization parameter $\lambda$, number of hidden layer neurons $hls$. weight limit $\epsilon$
 1: Initialize weights $W_1 \in \mathbb{R}^{62 \times hls}$ and $W_2 \in \mathbb{R}^{hls \times 62}$ randomly between $[-\epsilon, \epsilon]$
 2: for $i = 1$ to $num\_iters$ do
 3:     Forward propagation:
 4:         $z_2 = X_{train} * W_1'$
 5:         $a_2 = s(z_2)$
 6:         $z_3 = a_2 * W_2'$
 7:         $a_3 = s(z_3)$
 8:     Calculate the reconstruction error:
 9:         $error = a_3 - X_{train}$
10:     Backpropagation:
11:         $\delta_3 = error * gradient(z_3)$
12:         $\delta_2 = (\delta_3 * W_2) * gradient(z_2)$
13:         Add regularization error to $W_1$ and $W_2$ using $\delta_2$, $\delta_3$, and $\lambda$
14:         Update the weights $W_1$ and $W_2$ according to gradient descent.
15: end for
16: return $W_1$ and $W_2$
```

Then, W_1 and W_2 can be trained on the training data. Using the values of W_1 and W_2 obtained from this training, the reconstruction error of the validation data can be evaluated and the Area Under the (Performance) Curve (AUC) be stored. The maximum AUC on the validation set is then used to select the optimum α which is then trained on the test set to give the final AUC, which is reported.

It should be noted that the values of ϵ, λ, num_iters, and hls are completely arbitrary. We obtain a rather poor AUC for our data, but we cannot say whether this is due to the regularization penalty, weight initialization, or compression size without extensive testing. However, each run of the autoencoder takes significantly large computational time, so we present our curves as a proof of concept rather than a completely trained autoencoder. In theory, autoencoders should provide better performance if allowed to train over a large number of hyperparameters.

3.2 Isolation Forests

The Isolation Forest algorithm was introduced in 2008 by Liu et al as a faster and more efficient alternative to already-existing anomaly detection methods such as Random Forests and Local Outlier Factor (LOF) methods [4]. Existing anomaly detection methods started by creating a profile of 'normal' data points and checked whether every new point conformed to the profile. However, the big drawback in these methods was that the detector was optimized for detecting normal data points and not anomalies. To overcome this, it was proposed to take advantage of the scarcity of attribute points among normal data and the fact that their feature values were different from normal points. This makes the points more susceptible to isolation. The paper showed that it was possible to effectively construct a tree structure which would isolate all anomalies in a dataset, with the anomalies being closer to the root of the tree and the normal points being deeper in the tree.

The Isolation Forest algorithm builds an ensemble of trees, hereafter called *isolation trees*, on the data. The anomalies are then identified as those which have a short average path length on the trees. It uses only two variables: the number of trees to build, and the size of the subsamples. Subsamples are considered because there is no need to construct the deeper parts of the tree for the dataset because one would find more normal points when traversing deeper. Therefore, only a part of the tree must be constructed, greatly reducing computational time. There are also no density or distance measurements considered. This means that the usual computational effort needed to find the distance between pairs of points that are required in distance-based and density-based methods. The time complexity is also linear and has a low memory requirement, which makes it easier to handle large datasets.

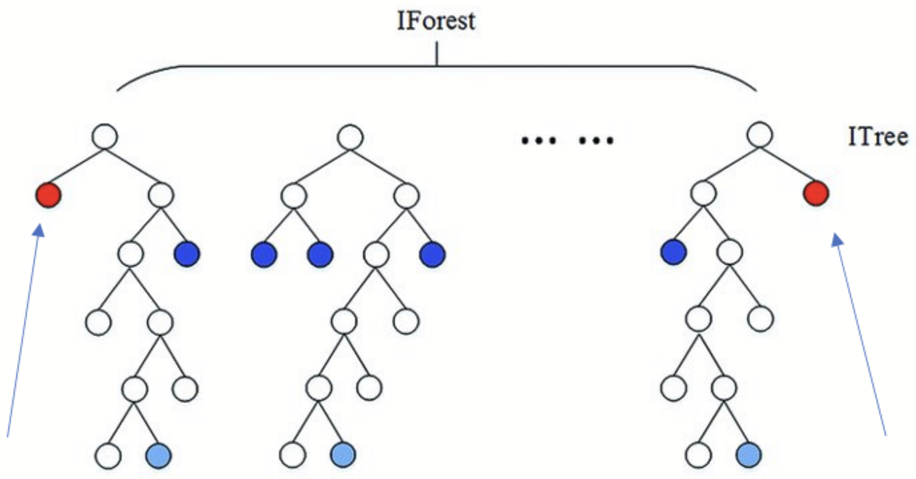

Figure 3: Visualizing an Isolation Forest

3.2.1 Isolation

Points are isolated when they are separated from other points. A visual explanation of isolating a point using with a tree is given below:

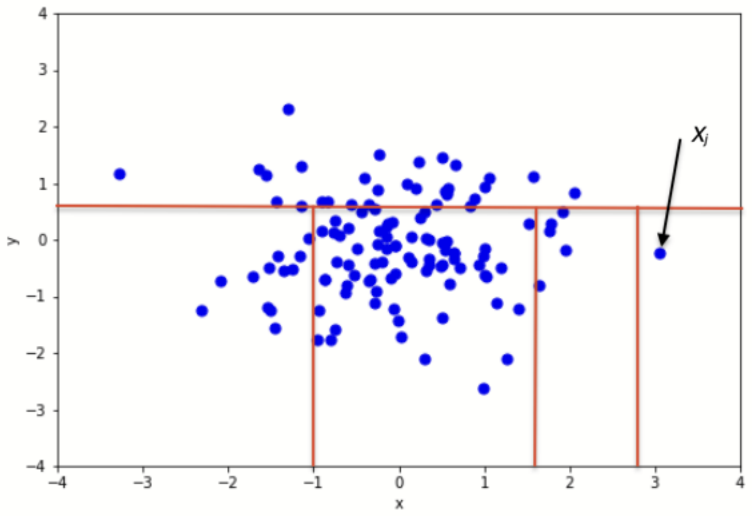

Figure 4: Separating a point in a dataset using a tree

Observe that to isolate a point towards the middle of the image, it would require a bigger tree. However isolating a point towards the edges would require a tree with fewer splits. This forms the basis of the algorithm. However, since the splitting is random, we need to calculate the *average* path length required to isolate a point and predict anomalies based on that value.

Let us define an Isolation Tree explicitly. An **Isolation Tree** is defined by its nodes. For every node **N** of an Isolation Tree, N either has no child (in which case N is an external node) or has two children (in which case N is internal) and N stores a *test*. A *test* is defined by an attribute *a* and a *split value v*, such that the operation $a < p$ divides the data points along N_l and N_r. Obviously, an Isolation Tree is a proper binary tree.

Given data $X = [x_1, x_2, ...x_m]$ from a distribution D, we divide X recursively by *random* selection of a and p, until $|X| = 1$ or the tree reaches the maximum height, or if all data in X have the same values. This means that for distinct values in X, the tree will be constructed fully with m external nodes and $m - 1$ internal nodes. Therefore, there are $2m - 1$ nodes and the memory requirement is linear in m.

How do we define an anomaly here? We can define the path length $h(x)$ as the number of edges traversed to reach a target node from the root. However, defining an anomaly score is difficult. Since the maximum height of the tree is $log_2(n)$ where n is the number of nodes, any growth in height is of logarithmic order. However, we know that in a Binary Search Tree (BST), the average path length required to unsuccessfully search n elements is

$$c(n) = 2H(n-1) - 2(n-1)/n \tag{4}$$

where $H(k)$ is the harmonic number and can be estimated by $ln(k) + 0.5772156649$, the latter number being Euler's constant. $c(n)$ is the average path length given n. Therefore, the anomaly score of an instance is defined as

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{5}$$

Therefore, if s is close to 1, then the point is an anomaly. If it is close to 0.5, then some other algorithm has to be employed to find anomalies. If it is close to 0 then the point is 'normal'.

3.2.2 Analysis

Some analysis of isolation trees is required. What happens when the anomalies are clustered together, as shown in the figure below?

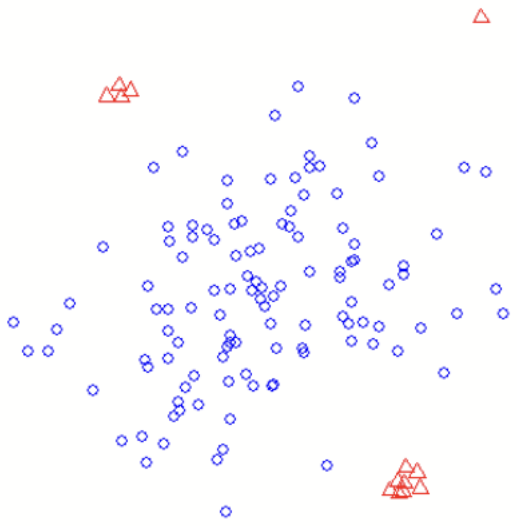

Figure 5: Cluster of anomalies

This is a specific case of a problem known as *masking*. Masking refers to the fact that the presence of too many anomalies leads to the algorithm not detecting them as anomalies. As seen in the figure, red triangle points take more splits to isolate, even though they are very clearly outliers. Isolation Forest provides an advantage in this scenario because it works well even with subsamples of the data. Another phenomenon, known as swamping, occurs when too many normal instances are identified as anomalies. This occurs when normal instances are too close to anomalies, such as the tail end of a normal distribution. Subsampling reduces the effect of swamping and masking.

3.2.3 Algorithms

Three algorithms - one for constructing the Isolation Forest, one for constructing the Isolation Tree, and one for finding the path length - are given.

Algorithm 2 Building Isolation Forests

```
Require: Input data $X$, number of trees $t$, subsampling size $s$
1: Set height limit $h$ as $log_2(s)$
2: Initialize $Forest$
3: for $i = 1$ to $t$ do
4:     $Y = sample(X, s)$
5:     $Forest = Forest \bigcup iTree(Y, 0, h)$
6: end for
7: return $Forest$
```

Algorithm 3 iTree(X,c,h)

```
Require: Input data $X$, current tree height $c$, height limit $h$
1: if $c \geq h\ OR\ |X| \leq 1$ then return $extNode(Size \leftarrow |X|)$
2: else
3:     Randomly select an attribute $a$
4:     Randomly select a split point $P \in [-a, a]$
5:     Partition the data into $X_l$ and $X_r$
6: return intNode($Left \leftarrow iTree(X_l, c+1, h)$, $Right \leftarrow iTree(X_r, c+1, h)$), $SplitAtt$,
   $SplitValue$
7: end if
```

Algorithm 4 PathLength(x,T,e)

```
Require: Element $x$ in a tree $T$, current path length $e$
1: if $T$ is external node then return $e + c(size(T))$
2: end if
3: $a \leftarrow T.SplitAtt$
4: if $x_a < T.SplitValue$ then return $PathLength(x, T.Left, e+1)$
5: else
6:     return $PathLength(x, T.Right, e+1$
7: end if
```

The basic operation of the algorithm is then as follows:

1. Select a subsample size and the number of trees
2. Break the data into subsamples
3. Construct isolation trees over each collection of subsamples
4. Evaluate the expected path length for an anomaly by passing each sample through the tree, for all trees
5. Compute anomaly scores using the equation defined above after obtaining the expected path length
6. Sort in descending order and the top k elements are the top k anomalies.

The original paper selected a subsample size of 256 and the 100 trees per forest, claiming that the values gave decent performance over multiple datasets. During our testing, we arrived at a value of 50 trees per forest and 500 elements per subsample. However it should be noted that since the splits are random, you can end up with different values. We obtained marginally better AUCs given the true labels in the categorical data for 50 trees. It seems as though the subsample size affects the performance more than the number of trees, as long as the number of trees isn't very low. The complexity of training the forest is $O(nslog_2(s))$, where n is the number of trees and s is the subsample size.

3.3 OCSVM

The One Class Support Vector Machine (OCSVM) algorithm, introduced in [6], is a semi-supervised machine learning algorithm that addresses the problem of anomaly detection by learning a decision boundary that separates normal data points from anomalous ones, using only a single class of normal data during the training phase. It is a variation of the traditional Support Vector Machine (SVM) algorithm that is designed for situations where labeled data is unavailable. This makes it ideal for stock data, as there is no explicit definition of a stock market anomaly.

3.3.1 Formal Definition

The formal definition was first outlined in [6].

Given a dataset $D = x_1, x_2, \ldots, x_n$ with n samples, where each sample $x_i \in \mathbb{R}^d$, One Class SVM aims to find a decision function $f(x)$ that maps the input data points x to a binary output, indicating whether the data point is considered normal ($f(x) \geq 0$) or anomalous ($f(x) < 0$).

The decision function $f(x)$ can be defined as:
$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i K(x_i, x) - b\right)$
where α_i are the dual variables, $K(x_i, x)$ is a kernel function that computes the inner product of the input data points x_i and x, and b is a threshold parameter that determines the decision boundary.

OCSVM aims to maximize the margin between the origin and the decision boundary in the feature space while minimizing the fraction of data points that are considered anomalous, controlled by a hyperparameter ν, where $0 < \nu \leq 1)$. This is achieved by solving the following optimization problem:
minimize: $\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j K(x_i, x_j) - \sum_{i=1}^{n} \alpha_i$
s.t. $\sum_{i=1}^{n} \alpha_i = 1,\ 0 \leq \alpha_i \leq \frac{1}{n\nu}, \forall i = 1, \ldots, n$

The solution of the optimization problem is given by the dual formulation. The support vectors are the data points x_i with non-zero α_i values, which determine the decision boundary. The bias term b can be computed as the average of the decision function values for the support vectors:
$b = \frac{1}{|\text{SV}_{\text{indices}}|} \sum_{i \in \text{SV}_{\text{indices}}} \left(\sum_{j=1}^{n} \alpha_j K(x_j, x_i) - 1\right)$
where $\text{SV}_{\text{indices}}$ denotes the set of indices of the support vectors.

3.3.2 OCSVM Implementation

The choice of hyperparameters can significantly impact the performance of the algorithm and requires careful tuning before use. Furthermore, One Class SVMs are sensitive to the presence of noisy data and work best when there is a clear distinction between normal and anomalous data.

To train a OCSVM function, the algorithm first separates the training data from the rest of the space, maximizing the margin between the origin and decision boundary. The hyperparameter controls outlier tolerance and margins, which are formulated as a Quadratic Programming Problem in MATLAB. Support vectors are identified from the quadprog output, which are training samples with non-zero alpha values. The time complexity of OCSVM can be high, with a worst-case scenario of $O(n^2)$.

3.3.3 Algorithm

The algorithm has the following specifications for input and output data:

- **Input parameters:**
  - `train_data`: A matrix containing the training data, where each row represents a data point and each column represents a feature.
  - ν: A hyperparameter that controls the trade-off between maximizing the margin and minimizing the fraction of data points considered anomalous $(0 < \nu \leq 1)$.
  - `kernel_function`: A function handle representing the kernel function used to compute the inner product of the feature mappings in the higher-dimensional feature space.
- **Returns:**
  - α: A vector containing the dual variables obtained by solving the dual Quadratic Programming Problem.
  - b: The bias term, used in the decision function to determine the decision boundary.

The full algorithm is shown below. In addition, a separate testing script was also created to train, validate, and test the stock data with OCSVM.

Algorithm 5 One Class SVM

```
procedure ONE_CLASS_SVM($train\_data$, $\nu$, $kernel\_function$)
    $n \leftarrow$ size($train\_data, 1$)
    $H \leftarrow$ zeros($n, n$)                                      ▷ Compute the kernel matrix
    for $i = 1$ to $n$ do
        for $j = 1$ to $n$ do
            $H(i, j) \leftarrow kernel\_function(train\_data(i, :), train\_data(j, :))$
        end for
    end for                                         ▷ Define the quadratic programming problem
    $f \leftarrow -$ones($n, 1$)
    $Aeq \leftarrow$ ones($1, n$)
    $beq \leftarrow 1$
    $lb \leftarrow$ zeros($n, 1$)
    $ub \leftarrow$ ones($n, 1$) $* \left(\frac{1}{n*\nu}\right)$                    ▷ Solve the quadratic programming problem
    $\alpha \leftarrow$ quadprog($H, f$, [], [], $Aeq, beq, lb, ub$)              ▷ Find support vectors
    $support\_vector\_indices \leftarrow$ find($\alpha > 1e{-}6$)               ▷ Compute the bias term
    $b \leftarrow 0$
    for $i = 1$ to length($support\_vector\_indices$) do
        $k \leftarrow kernel\_function(train\_data(support\_vector\_indices(i), :), train\_data)$
        $b \leftarrow b + (\alpha' * k' - 1)$
    end for
    $b \leftarrow \frac{b}{\text{length}(support\_vector\_indices)}$
    return $\alpha, b$
end procedure
```

Algorithm 6 One-Class SVM Test

```
Load training, validation, and testing data
Extract feature(s) from data
Initialize $nu\_values$, kernel_function
Initialize best_nu, best_validation_score
for each $nu$ in $nu\_values$ do
    Train One-Class SVM on training data with current $nu$
    Compute decision scores for validation data
    Update best_nu if validation_score is greater than best_validation_score
end for
Train One-Class SVM on training data with best_nu
Compute decision scores for testing data
Determine threshold for detecting anomalies
Detect anomalies in testing data using threshold
Calculate and output the percentage of detected anomalies
Plot the data with anomalies highlighted
```

3.4 K-Nearest Neighbour (K-NN)

K-nearest neighbors algorithm (k-NN) is a supervised learning method that is used for classification and regression. Unlike other machine learning algorithms k-NN does not require training. K-NN is a distance-based classifier, it assumes that two points are similar to each other if the points are closed in distance. There are a few distance metrics to use for k-NN, such as Manhattan distance, Minkowski distance, and Euclidean distance. For the purpose of this analysis the Euclidean distance will be used to calculate the pairwise distance between points. After calculating the Euclidean distances, the points are then sorted from least to greatest distance to identify the closest neighbors.

$$EuclideanDistance(d) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{6}$$

The k-value in k-NN is the number of nearest neighbors that should be considered when classifying a point. The k-value is a user define constant. It is not ideal to set k=1 because it would assign the output to the value of that single nearest neighbor. This would produce zero training error making the system too complex and overfit. A larger k-value would make the prediction more stable due to voting. To select the optimal k-value for supervised k-NN, k-values ranging from 2 to 21 were evaluated. The k-value that produced the minimum error rate was selected as the optimal value.

$$ErrorRate = \frac{1}{N}\sum_{j=1}^{N} 1(Y_{pred} \neq Y_{label}) \tag{7}$$

To find the optimal k-value for K-NN classification, multiple k values were evaluated. The algorithm is as follows:

```
Algorithm 7 Optimal k-value
Require: Input data: X-features, Y-class labels, randomly partition the data into training and testing
    points
 1: $N \leftarrow$ size(test_data, 1)
 2: $k\_values \leftarrow \{2, 3, \ldots, 21\}$                                  ▷ k-values to evaluate
 3: $k\_length \leftarrow$ size($k\_values$, 1)
 4: for $i = 1$ to $k\_length$ do
 5:     $d = \sqrt{(x\_test\_1 - x\_train\_1)^2 + \ldots + (x\_test\_n - x\_train\_n)^2}$       ▷ calculate the
    Euclidean Distance
 6:     [, idx] = sort(d)                                     ▷ sort distance from least to greatest
 7:     $Y\_pred =$ mode($Y\_train(idx(:, 1 : i))$)                           ▷ index for every k value
 8:     Rate $= \sum(Y\_pred \neq Y\_test)/N$                                   ▷ find the error rate
 9: end for
10: [Error_rate, $k\_idx$] = min(Rate)                          ▷ find the minimum error rate return
    $K\_best = k\_values(k\_idx)$                  ▷ find the k-value that produced the min error rate
```

The point can then be classified based on the majority class of the nearest neighbors. Figure 6 demonstrates how a point is classified in k-NN. In this example the k-value is three and there are two classes. The red dot can either be classified as blue or green. Since the k-value is 3, only the three closest neighbors to the red dot will be considered. In this case the red dot will be classified as blue because blue is the majority vote of its nearest neighbors.

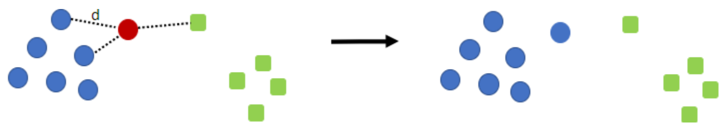

Figure 6: k-NN classification

K-NN algorithm with optimal k-value is as follows:

Algorithm 8 K-Nearest Neighbour

```
Require: Input data: X-features, Y-class labels, randomly partition the data into training and testing points
1: $d = \sqrt{(x\_test\_1 - x\_train\_1)^2 + \ldots + (x\_test\_n - x\_train\_n)^2}$  ▷ calculate the Euclidean Distance
2: [, idx] = sort(d)  ▷ sort distance from least to greatest
3: $k$ = optimal k  ▷ find optimal k-value return $Y_{\text{pred}} = \text{mode}(Y\_train(idx(:, 1 : k)))$  ▷ find the highest reoccurring class within your k nearest neighbors
```

3.5 Unsupervised K-Nearest Neighbour

Nearest Neighbour is the most popular distance-based technique that was first introduced in (E. Fix and J.L. Hodges.1951. An Important Contribution to Non-parametric Discriminant Analysis and Density Estimation.International Statistical Review Vol. 57, No. 3 (Dec., 1989), pp. 233-238).

The unsupervised K-Nearest Neighbour (K-NN) is a non-parametric approach that makes no assumptions about the distribution of data and can handle high-dimensional data-sets. It is a distance based approach categorizing points far from normal data to be anomalous, thus it measures how far a point is from the population, as shown in Figure 7, where the red dot represents an anomaly.

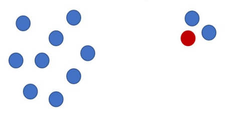

Figure 7: Working principle of Unsupervised K-NN

In cases of unsupervised anomaly detection, the algorithm uses only the feature values of the data-set, without the labels. The goal of the algorithm is to identify data points that deviate significantly from the expected behavior or patterns, which is accomplished by measuring the distance between each data point and its k nearest neighbors. By setting a threshold value, normal data points are distinguished from outliers. The computational complexity for density based K-NN is *O(mn+nlog(n)+n)*, where *mn* is the Euclidean for pairwise distances between the i-th and j-th observations, therefore number of training samples *m* and number of training features *n*; the *nlog(n)* is the sorting procedure, and *n* is the mean calculated for the distances.

For Unsupervised K-NN, no training data is required. The algorithm for Unsupervised K-NN is as follows:

Algorithm 9 Unsupervised K-Nearest Neighbour

```
Require: Load test set
1: Extract feature(s) from data
2: Specify K value
3: Compute pairwise distance
4: $[idx, ] = sort(d, 2)$  ▷ Sort distances and get indices for K+1 NN
5: $mean = mean(idx(:, 2 : k + 1), 2)$  ▷ Compute mean for the distances
6: Set threshold for outlier
7: Identify data points exceeding the threshold
8: Assign anomaly score
9: Assign a label of 1 to any anomaly score greater than 0
```

As with the supervised K-NN, the choice of K is crucial for the algorithm, as with the increasing K values, the boundary tends to get smoother. A good method to find the optimal K value is to use cross validation to analyze the effect of increasing K on the training and test error. The algorithm to find the optimal K value in an unsupervised setting is as follows:

Algorithm 10 Optimal K value for Unsupervised K-NN

```
Require: Load financial data set
 1: $k\_values \leftarrow \{1, 2, \ldots, 30\}$                                  ▷ k-values to evaluate
 2: Set number of folds for cross-validation
 3: Initialize array storing the average number of outliers for each k value
 4: for $i = 1$ to $k$_length do
 5:     Perform k-fold cross-validation:
 6:     Randomly divide data set into training and test sets
 7:     $d = \sqrt{(x\_\text{train1} - x\_\text{test1})^2 + \ldots + (x\_\text{train}n - x\_\text{test}n)^2}$     ▷ calculate the Euclidean
    Distance for each test point in the current fold
 8:     Calculate the mean distance for each test point and its neighbours
 9:     Identify outliers
10:     return Number of outliers in each fold
11: end for
```

In the case of unsupervised datasets, the error can be measured as the number of outliers. It is best to choose a k value that gives a constant number of outliers, in Figure 8 the optimal K value is 11.

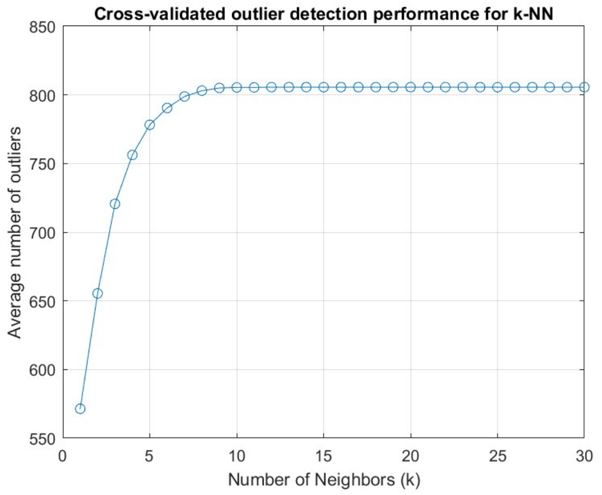

Figure 8: Choosing the optimal K value

An extension to Unsupervised K-NN, would be the Local Outlier Factor *LOF* algorithm which adds onto the unsupervised KNN by calculating Local Reachability Density *LRD* based on the distance to its k-nearest neighbor.The *LOF* would then be calculated based on *LRD* and outliers would be detected accordingly.

4 Experiments and Results

To test the efficiency of each algorithm, two data-sets were analysed: financial data and bank data. Anomaly detection in stock market data can help investors make better decisions about future stocks. Whereas bank datasets are often analysed to make decisions on whether a customer would be able to repay a loan based on various parameters.

Financial data was collected from Yahoo finance, with the use of Python. Rather than choosing one particular stock, data was collected from S&P500 index, as it comprises the 500 leading US companies. Financial data-sets are made of immense amounts of data accounting for volume, velocity and variety: there are many high-frequency interacting investors dealing and analysing data, whilst high-frequency algorithms reflect changes in prices within microseconds. However, despite its volatile nature, stock market data is not randomly generated and time-series observation at successive points can help with stock forecast. Stock market data was collected April 2007 until April 2023. Test data is the most recent one ranging between January 2019 until April 2023.

Bank data-set was collected using normalized data from GitHub, each customer is assigned an ID, and for each ID, there are 62 parameters such as occupation, number and amount of loans taken, marital status, education level, number of days taken to repay loans, cellular contract status and average salary. Then, there is an additional column for the labels assigned based on all parameters. The total number of customers is 41188 customers.

The following sections show results for the experiments conducted with each algorithm, as part of the analysis, ROC plots and AUC values have been measured for comparison.

4.1 Autoencoders

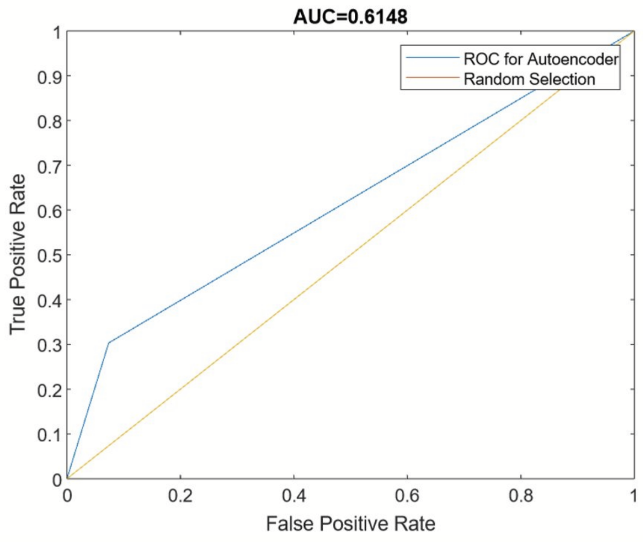

Figure 9: Bank Dataset: ROC for the single-layer autoencoder, with α=0.07

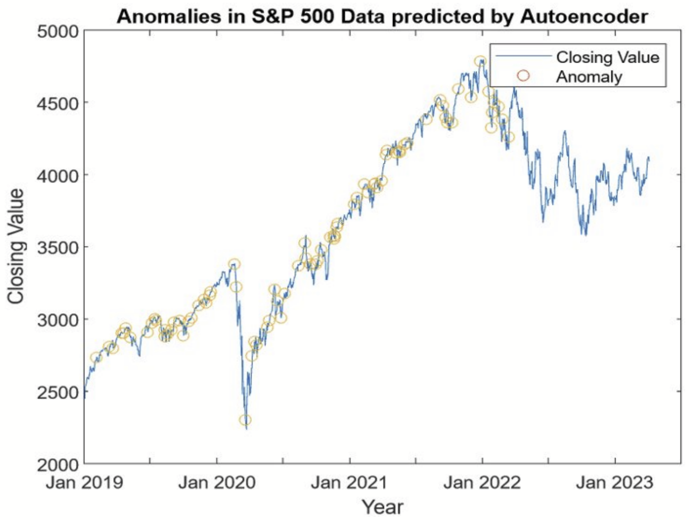

Figure 10: Anomalies predicted on stock market data by the autoencoder

4.2 Isolation Forests

The following ROC plots and AUC values have been obtained by testing different number of trees and rounds of splitting on the bank dataset. Whereas, figure 16 shows the performance of Isolation Forests on stock market test data.

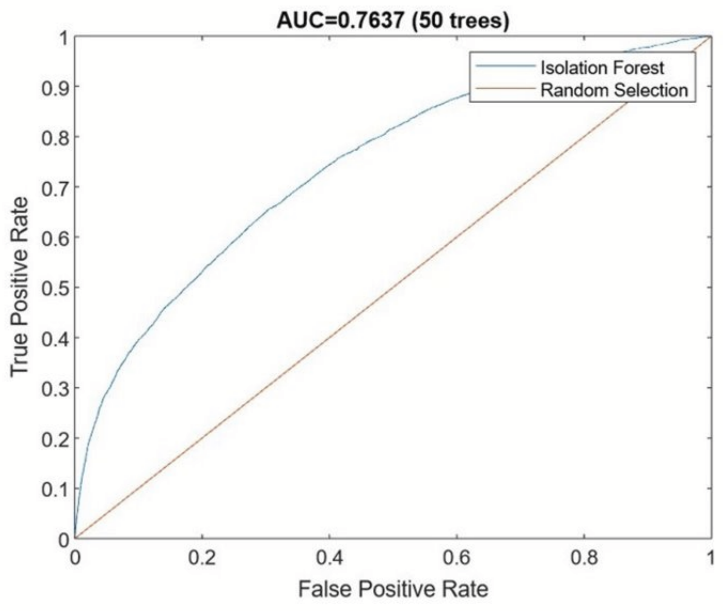

Figure 11: Best AUC using Isolation Forest

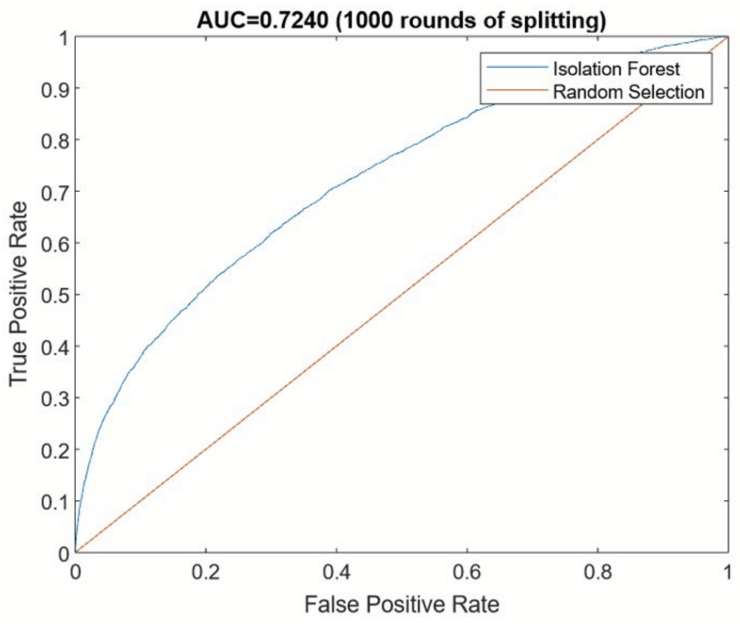

Figure 12: AUC for 1000 splitting rounds

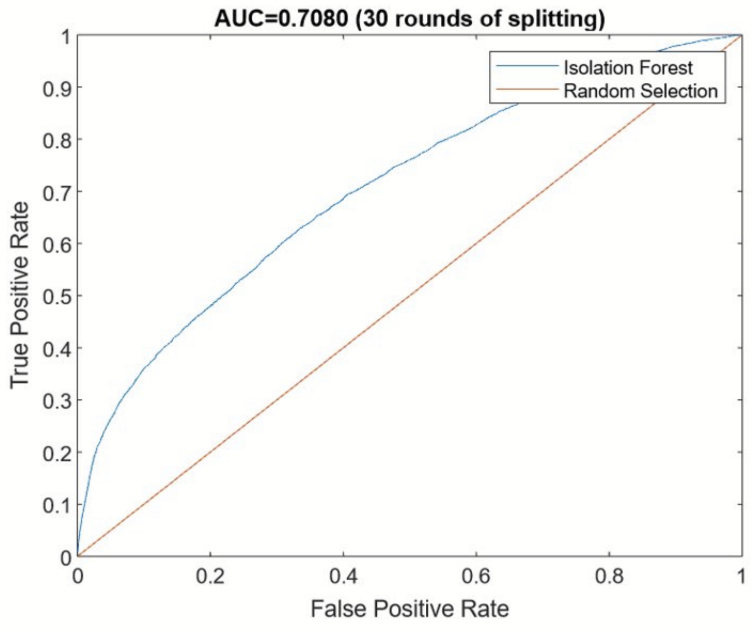

Figure 13: AUC for 30 splitting rounds

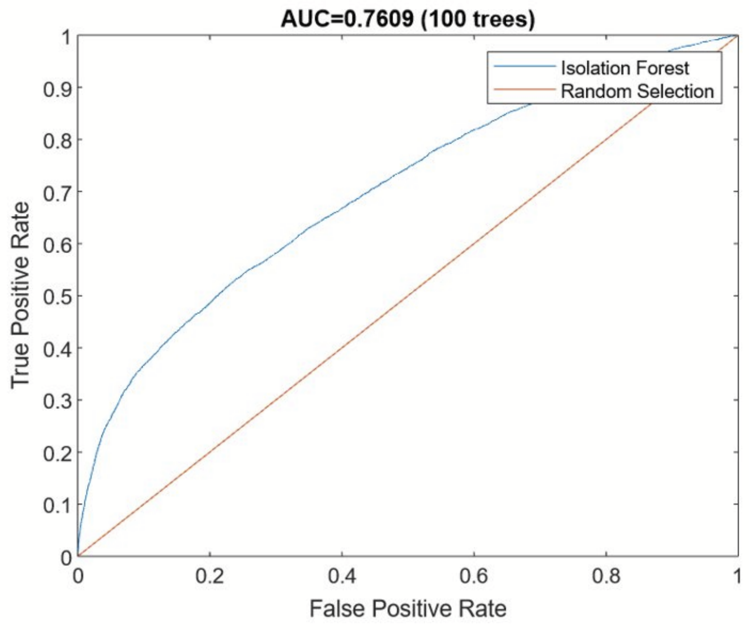

Figure 14: AUC for 100 trees

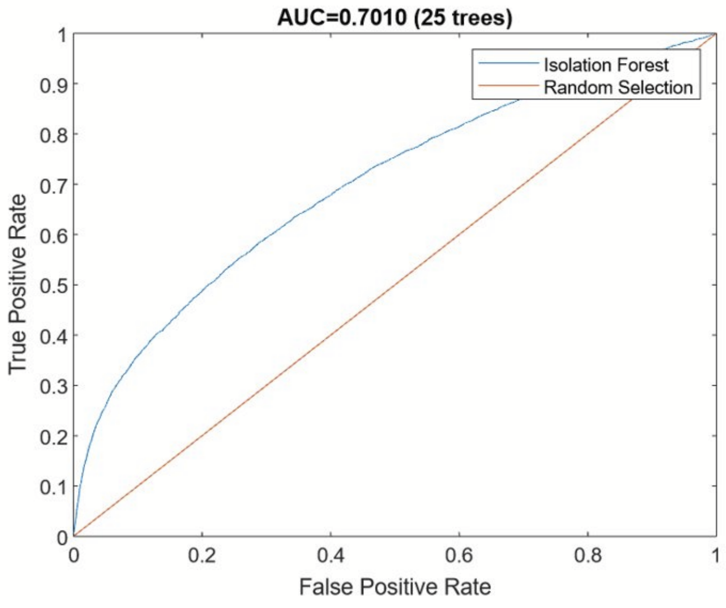

Figure 15: AUC for 25 trees

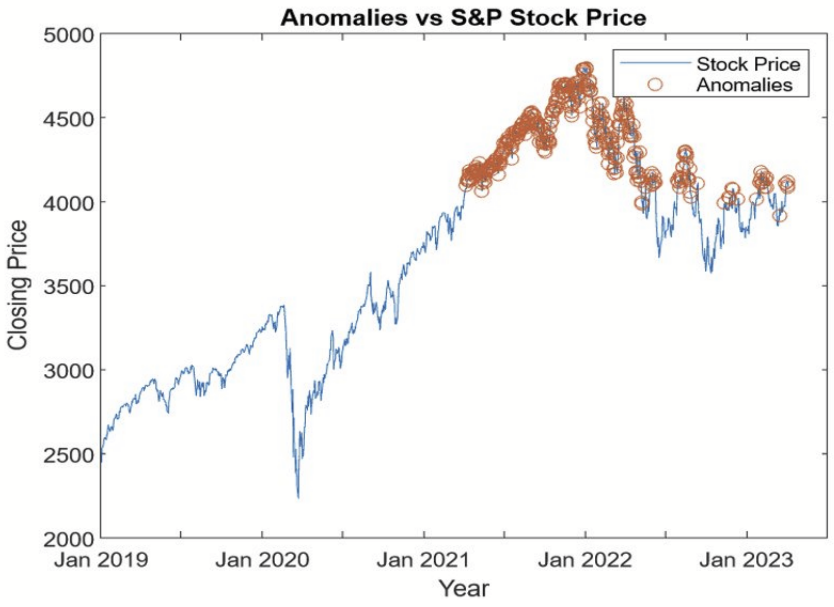

Figure 16: Anomalies predicted on stock market data by Isolation Forest

4.3 OCSVM

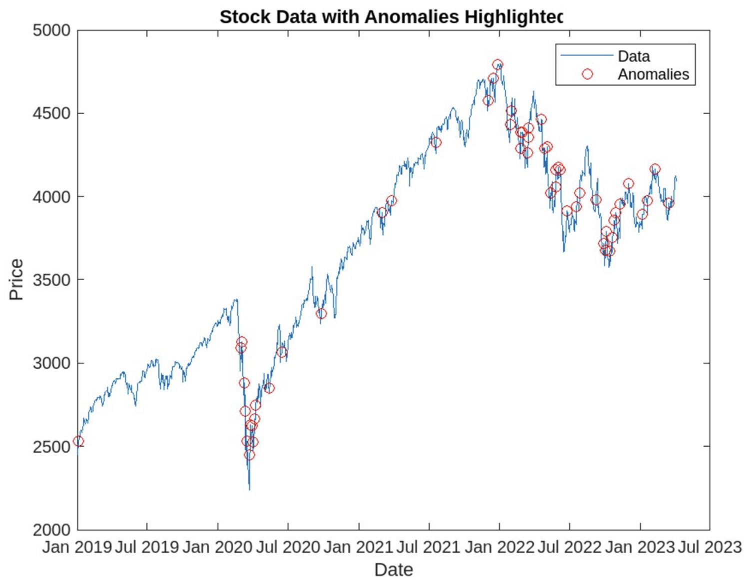

Figure 17: OCSVM Stock Price With Anomalies Highlighted (2019 - present)

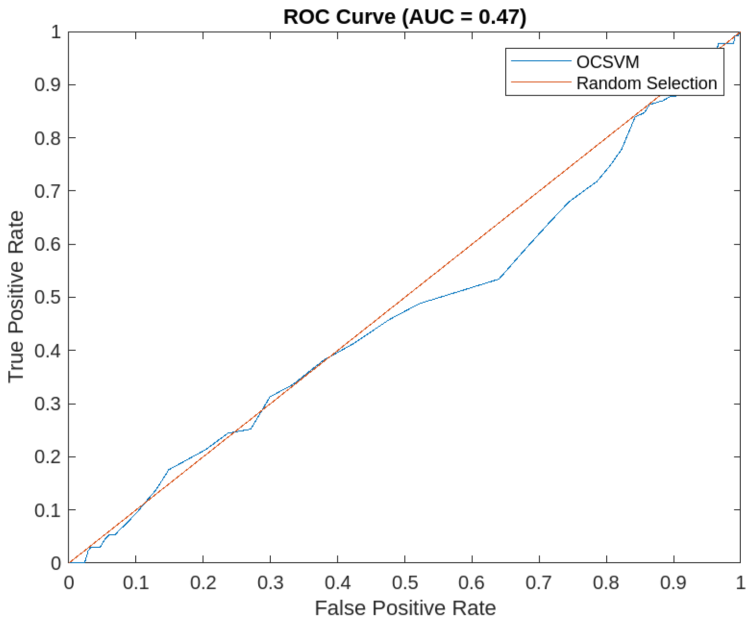

Figure 18: OCSVM ROC Curve

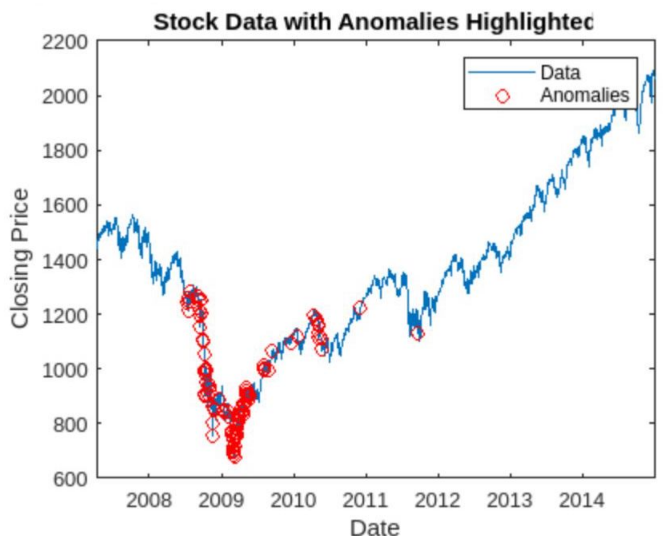

Figure 19: OCSVM Stock Price With Anomalies Highlighted (2007 - 2015)

4.4 K-Nearest Neighbour (K-NN)

The following graphs for KNN used the supervised bank data set.

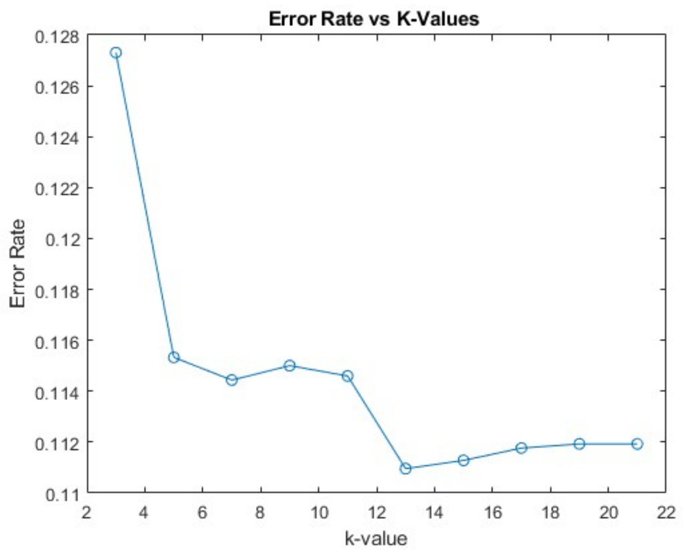

Figure 20: Error Rate vs K-values, optimal k =13

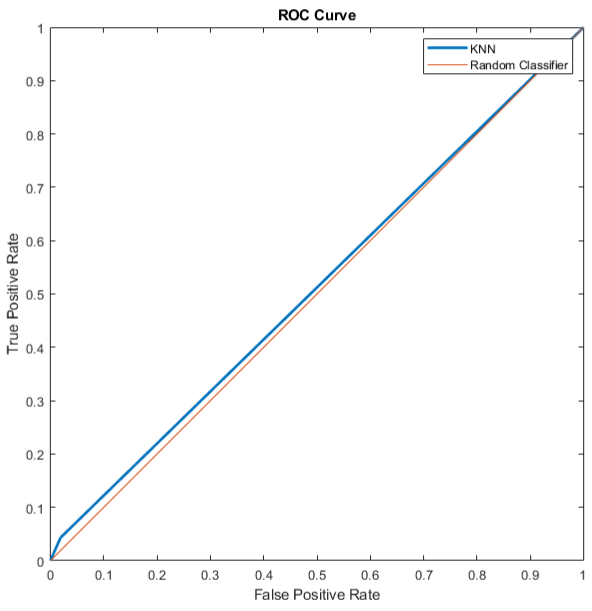

Figure 21: ROC for KNN

4.5 Unsupervised KNN

Figure 23 shows the closing stock price values for each day in the test set which ranges from January 2019 to April 2013. Synthetic abels were added to each point classified as an anomaly, and supervised K-NN was performed on the labelled version of the stock market data set. Figure 23 shows the corresponding ROC curve with an AUC value of 0.70.

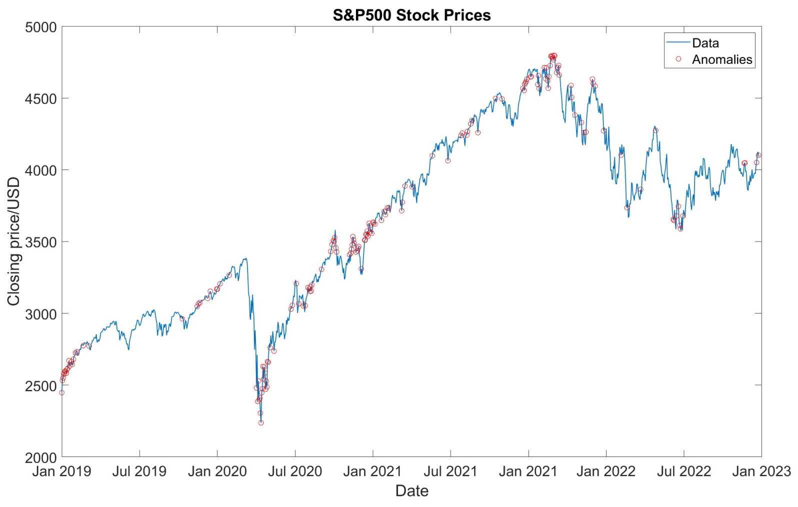

Figure 22: K-NN for stock market dataset

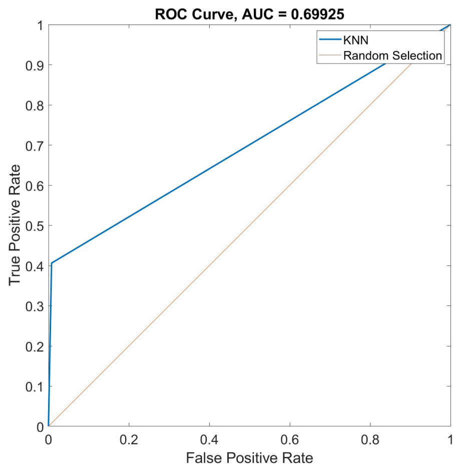

Figure 23: ROC curve for synthetically labelled S&P500 dataset

5 Analysis

5.1 Autoencoders

Autoencoders give us a low AUC with our parameters, but will take $O(n^k)$ time to train depending on the number of hyperparameters k we choose to train. Therefore, they are presented as a proof of concept rather than a fully trained autoencoder. Our autoencoder also failed to detect the sudden stock market drop due to the Russian invasion of Ukraine, but we cannot pinpoint the exact cause of this without further training.

5.2 Isolation Forest

Isolation Forest algorithms are fast and require low memory overhead. They have better performance than our autoencoder. Changing the number of split rounds affects the final AUC more than changing the number of trees. However, isolation forests do not seem to work so well on unlabeled data without prior information. They did not detect drops in the stock market due to COVID, even after repeated testing.

5.3 OCSVM

5.3.1 Anomaly Detection

The implementation of One Class SVM shown in Figure 17 uses S&P 500 index data that is split into three distinct portions for training, validation, and testing purposes. The training set consists of data from 2007 to 2016, the validation set spans from 2016 to 2019, and the testing set covers 2019 to the present. Relevant features, such as the difference between opening and closing prices or stock trade volume, are extracted from the data. However, the specific implementation of One Class SVM in Figure 17 only utilized closing prices as the primary feature. The Radial Basis Function (RBF) kernel was employed in the model to capture complex relationships in the data. Upon analyzing the detected anomalies in terms of their dates and time periods, it was observed that the One Class SVM model successfully identified notable sudden stock drops. These drops were attributed to significant events, including the COVID-19 pandemic, interest rate hikes by the Federal Reserve, and declines due to United States sanctions.

5.3.2 ROC Curve

Figure 18 presents the ROC curve for the OCSVM implementation. The curve was constructed by augmenting the original dataset with a new column of synthetic labels, which indicated whether a given date was an anomaly or not. These labels were derived from the detected anomalies shown in Figure 17, which clearly indicates which dates are associated with anomalies and which are not. The AUC for the ROC curve was calculated to be 0.47, indicating a performance that is worse than random chance, which would yield an AUC of 0.5. When compared to algorithms such as Isolation Forest, Autoencoders, and KNN, the performance of OCSVM appears to be objectively worse. However, it is important to note that One Class SVM is primarily designed for use with unlabeled data, and therefore, the ROC curve may not be an entirely accurate indicator of the model's accuracy or usability in this context.

5.3.3 Alternate Feature Usage

Figure 19 shows the results of an alternative One Class SVM implementation, which uses all columns of stock data, excluding the date, instead of solely relying on the closing prices. This approach includes a more comprehensive set of features and was tested on the stock market data from 2007 to 2015 in an attempt to detect the 2007-2008 Global Financial Crisis. Compared to the previous implementation that used only closing prices and nothing else, this new approach performed significantly better, with much fewer false positives and accurately detecting the beginning and ending dates of the stock market crash. These results show the importance of experimenting with different feature sets when employing anomaly detection algorithms. As evident in Figure 19, identifying and selecting the appropriate features can greatly improve the performance of the model.

5.4 Unsupervised K-NN

The plots for both One Class SVM and Unsupervised K-NN are fairly similar, they both are able to detect major drops due to COVID-19 which caused a drop of 65% during March 2020, and the decrease in prices due to the war tension initiated in 2022. Similarly anomalies in surges during when the prices increased due to low interest rates in January 2021 due to inflation and low interest rates has also been detected.

The similarity between OCSVM and Unsupervised K-NN is due to the fact that both use Euclidean distances, other types of distances can be experimented to see how anomaly scores would change.

The AUC value for Unsupervised K-NN was greater than the AUC value achieved using One Class SVM. Overall the density based unsupervised K-NN performed better than the other algorithms tested on stock market data.

5.5 K-NN

The optimal k-value is selected based on the minimal error rate which is k=13 as shown in figure 20. The optimal k-value is than plugged into the K-NN algorithm, which classifies the data points as anomaly or regular points. The ROC curve for K-NN is shown in figure 21, which indicates that there is no relationship and k-NN is a random classier based on the straight line produced. When analyzing the data points this makes sense because the data is imbalanced, there are more regular points than anomaly points. This indicates that k-NN is not a suited algorithm for anomaly detection. K-NN classifies points based on the nearest neighbors, since there are more regular points than anomaly points K-NN will misclassify anomaly points as regular points.

Alternative methods to prevent imbalance data is to under-sample the dataset by randomly reducing the data points in the regular class to match the size of the anomaly class. Another method is over-sampling, which balances the data by increasing the size of the rare data points. These two methods are not ideal because under-sampling may remove important data points. While over-sampling would increase the occurrence of anomaly detection. The ideal method would be to add more weights to the anomaly points over regular points. For example, if the k-value is three and there are two regular points and one anomaly point. The anomaly point should have three times the weight of one regular point. Adding more weights to anomaly points could produce a better ROC curve for K-NN classification.

6 Conclusion

In conclusion, this study evaluated the performance of anomaly detection techniques using Autoencoders, Isolation Forests, K-Nearest Neighbors (KNN), Unsupervised KNN, and One-Class Support

Vector Machines (OCSVMs). The experimental results demonstrated that all five techniques were capable of detecting anomalies, but each method exhibited its own strengths and weaknesses. We conclude that Autoencoders have the potential to be exceptionally powerful if fully trained; however, the computational power and time required for training must be considered. Isolation Forests serve as a good starting point for quickly identifying the majority of anomalies in a dataset, but additional analysis is necessary to achieve a more comprehensive result.

Utilizing One Class SVM (OCSVM) on the S&P 500 stock data successfully identified notable sudden stock drops, including those attributed to significant events such as the COVID-19 pandemic, interest rate hikes by the Federal Reserve, and declines due to United States sanctions. The ROC curve for the OCSVM implementation, however, indicated a performance worse than random chance, when compared to algorithms like Isolation Forest, Autoencoders, and KNN. It is essential to note that One Class SVM is primarily designed for use with unlabeled data, so the ROC curve may not be a completely accurate indicator of the model's accuracy or usability. An alternative One Class SVM implementation using all columns of stock data, excluding the date, showcased significantly better performance in detecting the 2007-2008 Global Financial Crisis.

K-NN is not recommended algorithm to use with 'unbalanced' data, since majority of the classifications would be considered regular points over anomaly points. To use K-NN for anomaly detection, anomaly points should have more weight than regular points. Adding more weights to the anomaly points could produce better predictions for K-NN. Other variations of K-NN such as Unsupervised KNN would be better suited for anomaly detection. Unsupervised KNN detects abnormalities within the data, which is favorable in detecting anomalies.

The comparative analysis of these techniques provides valuable insights into their applicability and usefulness in different settings and contexts. The results of this study demonstrate that no single technique is optimal for all situations, and the choice of technique depends on the specific characteristics of the dataset and the requirements of the application.

Future research in anomaly detection should focus on developing hybrid approaches that combine the strengths of multiple techniques and can handle the limitations of individual techniques. Additionally, the evaluation and interpretation of detected anomalies remain an open problem, and further research is needed to develop effective methods for anomaly interpretation and visualization.

Overall, this study contributes to the ongoing research in anomaly detection and provides a comprehensive evaluation of four commonly used techniques. The insights gained from this study can be useful for researchers and practitioners in various fields, including finance, healthcare, and cybersecurity, where detecting anomalies is critical for identifying potential risks and problems.

7 Work Distrubution

Name	Contribution
Kshitij Duraphe	History of Anomaly Detection, Collecting Bank data, Implementation/Analysis of Isolation Forests and Autoencoders
Kashyap Panda	One-Class SVM Methodology, Experiments/Results, Analysis, Conclusion
Priscila Rubio	K Nearest Neighbors, Optimal k-value, Unsupervised KNN, Experiments/Results, Analysis, Conclusion
Anjena Daswani	Initial Project Formulation, Collection of Stock Market Data, Abstract and Introduction, Implementation/Analysis for Unsupervised K-NN

References

[1] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. ACM Comput. Surv. 41, 3, Article 15 (July 2009), 58 pages. https://doi.org/10.1145/1541880.1541882

[2] arXiv:2003.05991 [cs.LG]

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Learning internal representations by error propagation. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations. MIT Press, Cambridge, MA, USA, 318–362.

[4] F. T. Liu, K. M. Ting and Z. -H. Zhou, "Isolation Forest," 2008 Eighth IEEE International Conference on Data Mining, Pisa, Italy, 2008, pp. 413-422, doi: 10.1109/ICDM.2008.17.

[5] Francesco Orabona, EC503 'Learning From Data', Lecture 21, Spring 2023, 'Backpropagation', Boston University

[6] Krikamol Muandet and Bernhard Schölkopf. 2013. One-class support measure machines for group anomaly detection. In Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence (UAI'13). AUAI Press, Arlington, Virginia, USA, 449–458.

[7]Kramer, O. (2011). Unsupervised K-Nearest Neighbor Regression. [online] arXiv.org. https://doi.org/10.48550/arXiv.1107.3600