Semi-Supervised Multiclass Classification with Novelty Detection Using Support Vector Machines and Linear Discriminant Analysis

by

Ryan Dove

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
December 14, 2024

Keywords: Novelty Detection, Support Vector Machines, Linear Discriminant Analysis

Approved by

Roy J. Hartfield Jr., Chair, Walt and Virginia Woltosz Professor of Aerospace Engineering
Mark Carpenter, Co-Chair, Professor of Mathematics and Statistics
Davide Guzzetti, Member, Assistant Professor of Aerospace Engineering

Abstract

Semi-supervised multi-class classification with novelty detection consists of one tool that can accurately define what class an instance belongs to from a set group of known classes while simultaneously detecting instances that do not belong to any of the classes. This field of machine learning has been studied for a decade or more, with many different algorithms and solutions tested. In this thesis, an algorithm built using One-Class Support Vector Machines (OCSVM) and Linear Discriminant Analysis (LDA) will be explored and demonstrated on data from simulated missile trajectories. Not only is this algorithm novel to the aerospace industry, but also to the computer science and machine learning industries as well. Its strengths and limitations will be explored, as well as techniques to increase accuracy and ideas for future work and other uses for this tool.

The data used is produced by the Auburn University Solid Rocket Code (AUSRC), a validated code which simulates the flight of a missile given its design parameters. The output data from the AUSRC is preprocessed using common techniques such as standardization, to eliminate the influence of units of measurement, and principal component analysis (PCA) as a method of data reduction. The data is then fed into an iterative OCSVM, during which necessary data is recorded to extract any outlier points. An objective threshold is used to differentiate between instances that are outliers of a known clean class and instances that are true novelties from an unknown class. The instances deemed 'true' novelties by the threshold are added to the training data to fit the LDA to. The data is then run through the LDA, which provides a higher classification accuracy than the OCSVM, cleaning up inter-class misclassifications as well as tracking the outliers back to their known class. Since the LDA was trained on the 'true' novelty class data, it acts as a novelty detector by extracting any other points the OCSVM missed.

This method works accurately ($> 90\%$ overall accuracy) on simulated missile trajectories across 3 different data sets and robustness tests. The data set features used are comprised of performance parameters, trajectory data, or derived features. The tests comprise of reduced

time frames and random missing data. The OCSVM provides high accuracy novelty detection while the LDA provides high accuracy classification, exploiting the strengths of each machine learning algorithm. Further work should be done to improve the hybrid method and to investigate ways to eliminate the LDA's underlying assumption of the data set to have a multivariate Gaussian distribution.

Acknowledgments

I would first like to thank my advisors Dr. Roy Hartfield and Dr. Mark Carpenter. Thank you to Dr. Hartfield for believing in and recruiting me to Auburn which was instrumental into me receiving my Master's degree. His expertise in the aerospace field has greatly benefited me and expanded my knowledge base. Thank you to Dr. Carpenter for your advice and guidance throughout my tenure as well. His instruction on statistical methods allowed me to develop the OCSVM-LDA hybrid method and greatly improved the project laid out in this thesis. I am greatly indebted to the both of you for your mentorship and helping to advance my career through a graduate degree. I would like to thank my family for supporting me throughout my studies and for their constant encouragement these last few years. I would like to thank my friends for making my experience at Auburn enjoyable and memorable. I am truly grateful to have friends, family and advisors who challenge me and push me to reach my goals.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

α	SVM Soft Constraint Parameter
γ	Linear Support Vector Machine Margin
γ_{RBF}	RBF Kernel Learning Rate Parameter
λ_1	PCA Eigenvalue
$\mathbf{\Sigma}$	LDA Covariance Matrix
μ	LDA Class Mean
π	LDA Prior Probability of a Class
$\mathbf{S}$	PCA Covariance Matrix
$\mathbf{u}_1$	PCA Projection Direction Vector
$\vec{\mathbf{S}}_B$	LDA Between-Class Scatter Matrix
$\vec{\mathbf{S}}_W$	LDA Within-Class Scatter Matrix
ξ	SVM Slack Parameter
b	Y-Axis Intercept Variable for Linear Hyperplane
C	SVM Error Rate Parameter
D	Matrix of distances to each point from the Hyperplane
m_k	LDA Mean Projection

s_k^2 LDA Variance

x' Input Vectors for RBF Kernel

y_i Sign on SVM Class $(-1, +1)$

z_n LDA Data Projection

w Normal Vector from Separating Hyperplane

x Matrix of Coordinate Points for Data Sets

AUSRC Auburn University Solid Rocket Code

BURNTIME Burn Time

DBODY Body Diameter

DOF Degree of Freedom

FINENESS fineness ratio

LDA Linear Discriminant Analysis

MAXPC Maximum Chamber Pressure

MAXPE Maximum Exit Pressure

MAXTHRUST Maximum Thrust

OCSVM One-Class Support Vector Machine

PCA Principal Components Analysis

RBF Radial Basis Function

SVM Support Vector Machine

THROAT Throat Diameter

TIME Time From Detection

VZMAX Maximum Velocity in Z Direction

Chapter 1

Introduction

Multi-class classification with novelty detection is an ongoing issue, with many researchers testing and proposing methods to effectively accomplish this feat. Novelty detection involves detecting instances that deviate from the norm of the clean data available to train on, and there are many machine learning algorithms that can accurately perform this [17]. Novelty detection is found in many industries and real world uses, including bioinformatics [19], seizure analysis [20], fraud identification [21], data stream monitoring [22], inspection robotics [23], and planetary exploration [24] to name a few. Extending this principle to multi-class classification with novelty detection increases the difficulty of this problem, because a machine learning algorithm must not only learn the feature space of the known classes but also reserve the probability of determining a point to not belong to any of the known classes. This is difficult because many statistical learning algorithms use probability functions that 'force' all points into a class, classifying the point purely based off of which class it scores highest for.

A popular state-of-the-art method for tackling this problem is using Support Vector Machines (SVM). Many papers have been published demonstrating methods on how to use support vector methods or how to make them more accurate [10] [25] [26] [27] [28]. The SVM is traditionally a binary classification algorithm between two classes, but the One-Class Support Vector Machine (OCSVM) only trains on one class and uses hyperparameters and a kernel to define a hyperplane that returns a binary classification as either belonging to the trained on class or not. The OCSVM can be implemented iteratively to learn multiple hyperplanes, shown to be more accurate than lumping all known classes together into a large group [11] [18], with the advantage of the opportunity for classification.

The OCSVM has its disadvantages, including suffering from the 'curse of dimensionality' [3] where too many features causes the algorithm to become inaccurate due to no built in feature weight function. Zhu et al. [28] attempted to recify this by adding their own. Others have tested their own methods that resemble some principles from OCSVMs, such as non-linear transformations [11] [29], large margin distribution [30], and a neural network that only defines trained on feature space [12].

The data sets used in this thesis are comprised of simulated missile trajectories modeled by the Auburn University Solid Rocket Code (AUSRC). It is FORTRAN tool that calculates the mass properties, aerodynamics, propulsion, 6-DOF flight, guidance and autopilot for a missile given its design inputs to model the trajectory. Aerospace vehicles such as missiles can be safely assumed to have a multivariate Gaussian class distribution due to their flight windows and performance limitations. This allows the use of Linear Discriminant Analysis (LDA) on the data, which requires this distribution.

The iterative OCSVM analysis is ran first and returns its results, consisting of predicted classification labels as well as points deemed novelties. These novelty points contain false positives, or points that are outliers of a clean class, not a true novelty from an unknown class. An objective threshold is used to split the two. The points that score below the threshold are deemed 'true' novelties and can be added to the training data. The LDA can then take the results or new incoming data and increase the overall classification accuracy, increase novelty detection accuracy, and reclassify false positives back to their clean class.

Chapter 2

Data Generation

2.1 Auburn University Solid Rocket Code (AUSRC)

The Auburn University Solid Rocket Code is a tool for developing the trajectories of solid propellant missile systems [1]. This FORTRAN tool contains routines to develop mass properties, propellant performance characteristics, missile aerodynamics, potentially guidance, and a resulting simulated flight trajectory. The inputs for the missile design parameters are stated as a range of values, set as a median with a noise range, and a chosen random sampling routine is used to run various scenarios, returning the data for the successful flights while discarding the failed ones. The user can set the number of successful flights required to be returned before terminating the simulations. One should refer to Cervantes's doctoral dissertation [1] for more information on the AUSRC including details for each subroutine that calculates the mass properties, aerodynamics, propulsion, 6-DOF flight, guidance and autopilot, as well as the sampling routines used in the Auburn code and tools. The solid propellant rockets are loosely sized by the performance of the liquid propellant SCUD-B missile, a tactical ballistic missile developed by the Soviet Union.

2.2 Box and Whisker Plots

Box and whisker plots are used throughout the paper to visualize the data sets. Figure 2.1 shows a diagram of how they are formatted. The interquartile range is encapulated by the box, representing the middle 50% of the data, where the center line is the median and the edges of

the box represent the upper and lower quartile. The whiskers extend from each end of the box and extend out to an extreme of the data. Each whisker also represents 25% of the data.

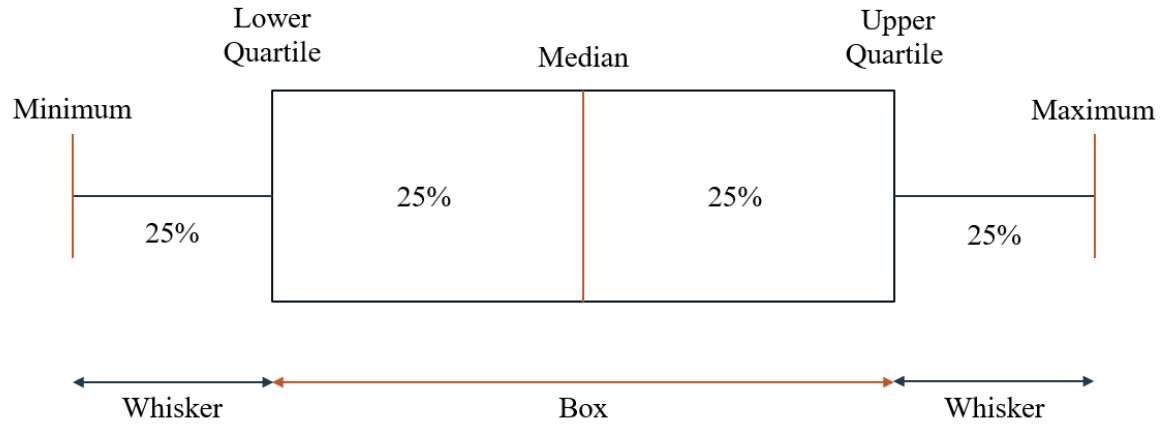


Fig. 2.1: Box and Whisker Plot Explanation

The box and whisker plots show the range of the inputs, where the AUSRC samples a point from, as well as the range of the outputs in the data set following the missile flyout simulations. The box and whisker plot visualization allows the reader to better understand the data set as well as gain valuable insight into the relationships between the variance of inputs on the affect of outputs.

2.3 18 Class Data Set

The '18 Class Data Set' is used as a proof of concept for all advances in the hybrid method because of its relative ease of classification. The data set developed for this initial demonstration effort consists of 18 different classes of ballistic missiles, differentiated by the inputs DBODY (body diameter), THROAT (throat diameter), and FINENESS (fineness ratio). A total of $45,000$ runs were generated, or $2,500$ runs per class. Table 2.1 shows the 3 discriminating input values for each of the 18 classes. There are 3 different values for DBODY, 3 different values for FINENESS, and 2 different values for THROAT.

Table 2.1: 18 Class Data Set Mean Value for Discriminating Inputs

Class	Body Diameter (DBODY)	Fineness Ratio (FINENESS)	Throat Diameter (THROAT)
1	0.844	8.91878	0.14264
2	0.884	8.91878	0.14264
3	0.924	8.91878	0.14264
4	0.844	9.11878	0.14264
5	0.884	9.11878	0.14264
6	0.924	9.11878	0.14264
7	0.844	9.31878	0.14264
8	0.884	9.31878	0.14264
9	0.924	9.31878	0.14264
10	0.844	8.91878	0.18264
11	0.884	8.91878	0.18264
12	0.924	8.91878	0.18264
13	0.844	9.11878	0.18264
14	0.884	9.11878	0.18264
15	0.924	9.11878	0.18264
16	0.844	9.31878	0.18264
17	0.884	9.31878	0.18264
18	0.924	9.31878	0.18264

DBODY can be seen in Fig. 2.2 for each class with an added noise of $+/-0.002$ meters. FINENESS can be seen in Fig. 2.3 for each class with an added noise of $+/-0.01$. THROAT can be seen in Fig. 2.4 for each class with an added noise of $+/-0.006$ meters. The noise is added as a part of the random sampling subroutine in the AUSRC. The subroutine picks a value within the bounds of the variable and simulates its flight. If the variables are not compatible, the run is terminated and new variables within the bounds are chosen. This routine continues until the program performs the set number of required successful runs, in this case $2,500$ per class.

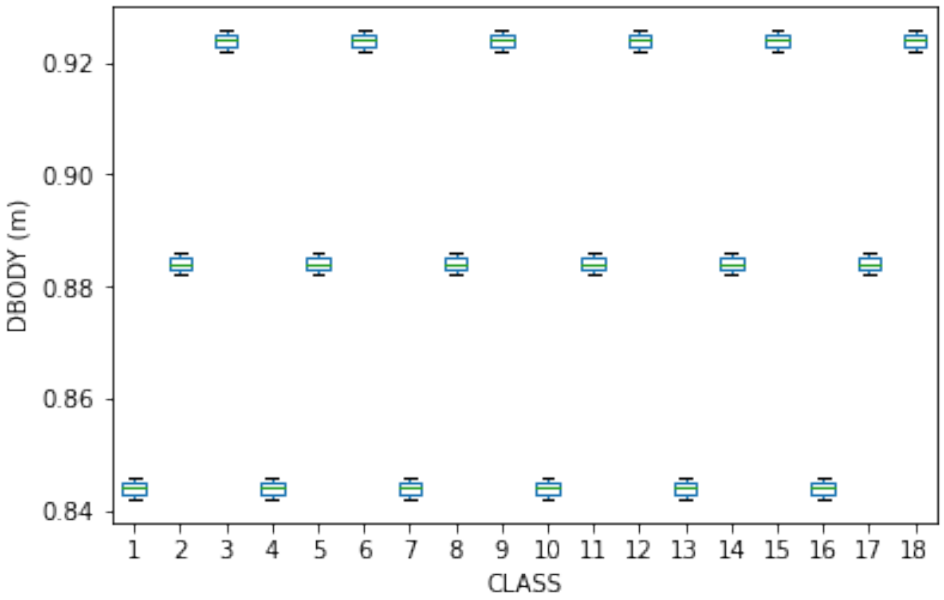


Fig. 2.2: DBODY Box Plots for Each Class (18 Class)

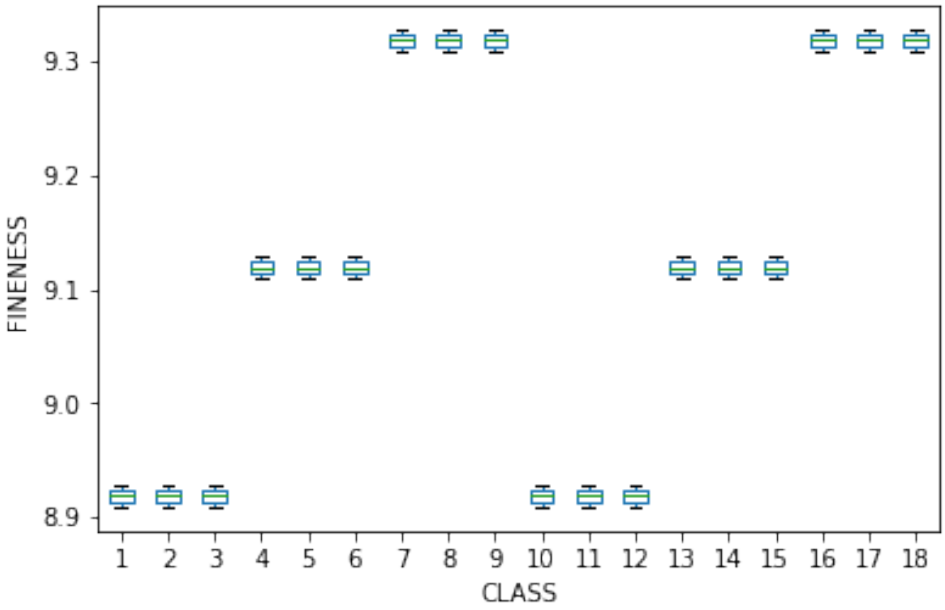


Fig. 2.3: FINENESS Box Plots for Each Class (18 Class)

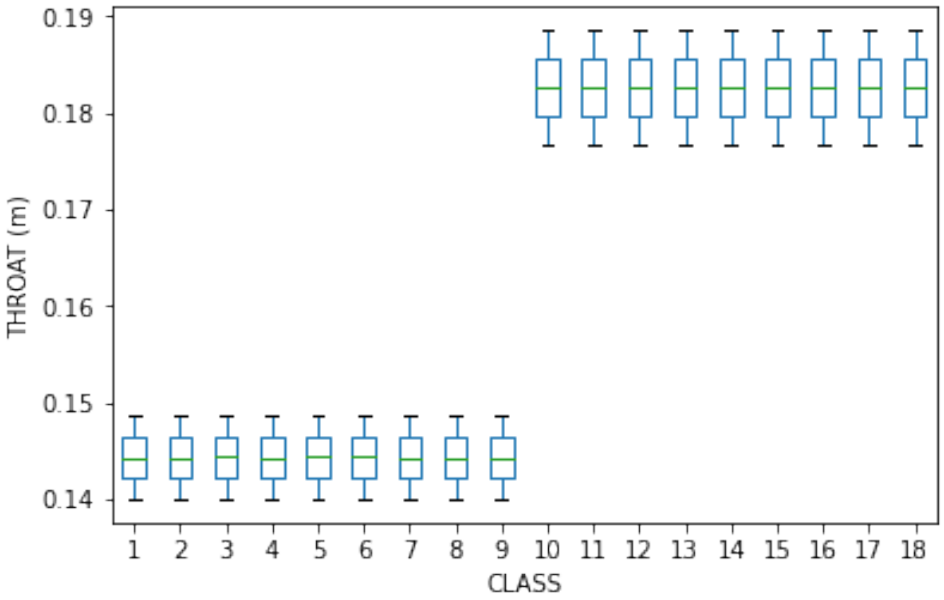


Fig. 2.4: THROAT Box Plots for Each Class (18 Class)

The outputs used from this data set as 'features' are BURNTIME (Burn Time), MAXTHRUST (Maximum Thrust), MAXPC (Maximum Chamber Pressure), and MAXPE . While these performance parameters are not readily available from an adverse missile, this data set still provides valuable insight to how the OCSVM iterative algorithm functions (OCSVM Iterative Method Alone, Section 5.1) and how preprocessing the data increases accuracy (Data Handling, Section 4.1).

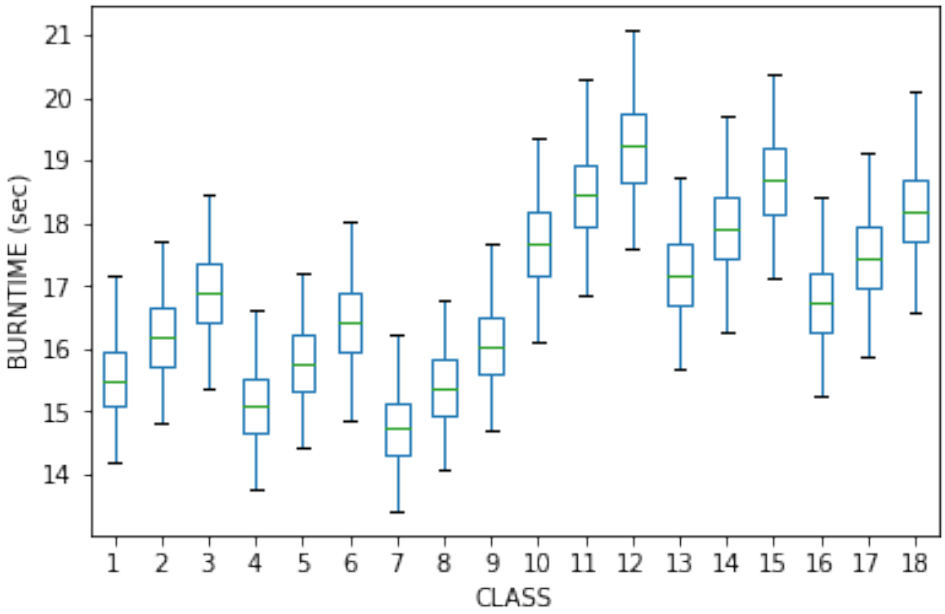


Fig. 2.5: BURNTIME Box Plots for Each Class (18 Class)

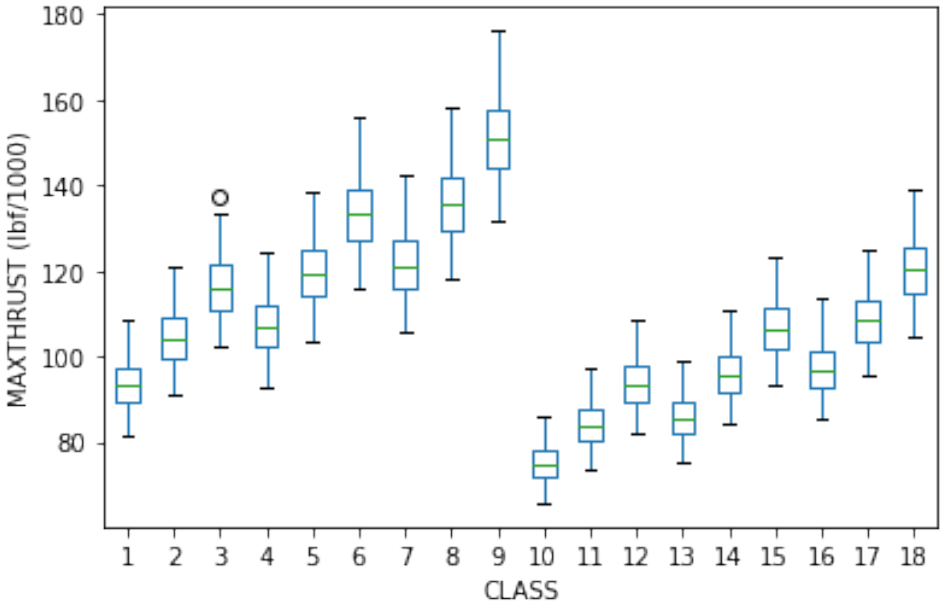


Fig. 2.6: MAXTHRUST Box Plots for Each Class (18 Class)

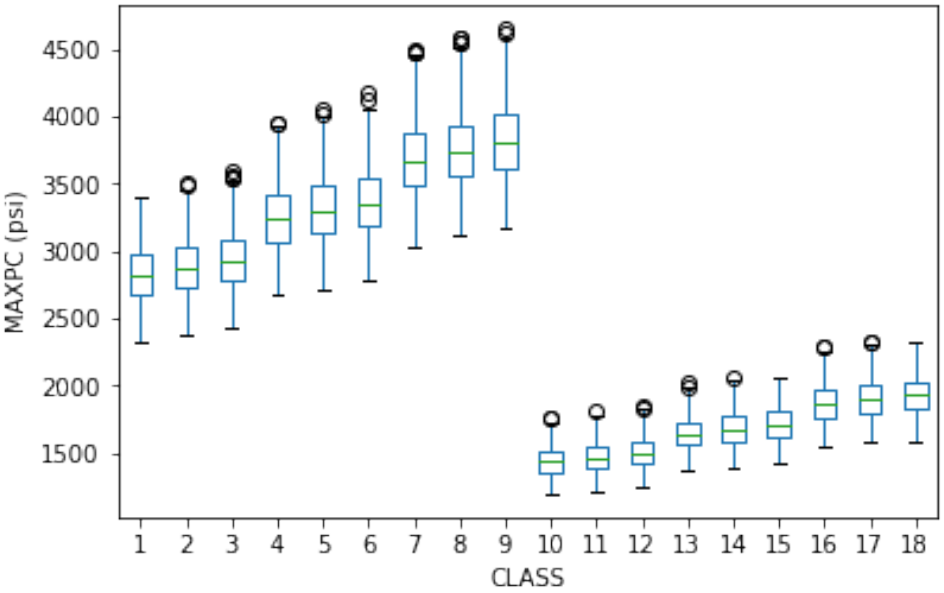


Fig. 2.7: MAXPC Box Plots for Each Class (18 Class)

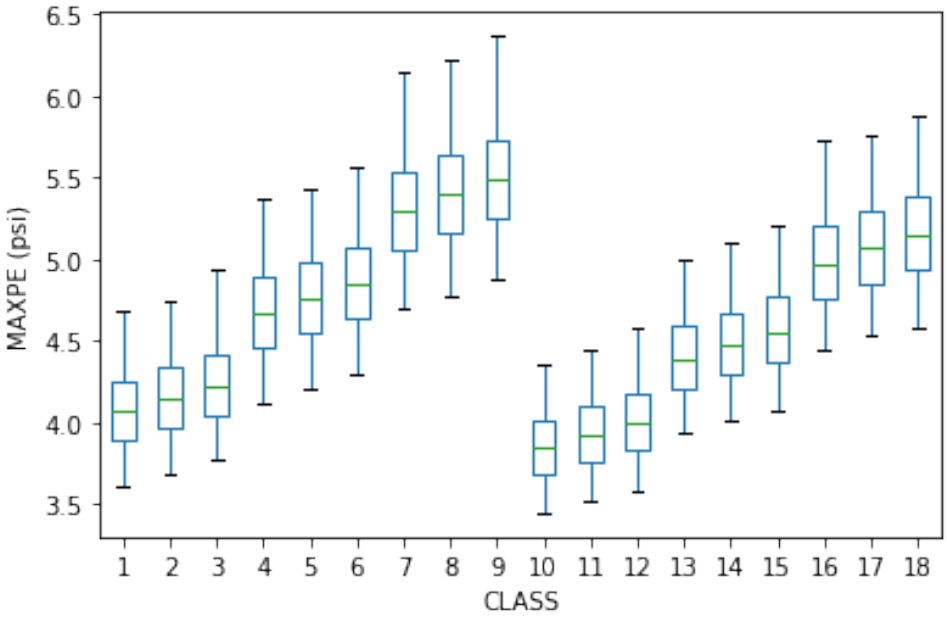


Fig. 2.8: MAXPE Box Plots for Each Class (18 Class)

The inputs can be used to analyze how they affect the outputs. Increasing THROAT leads to an increase in BURNTIME, while decreasing MAXTHRUST, MAXPC, and MAXPE. Increasing FINENESS leads to a decrease in BURNTIME, while increasing MAXTHRUST, MAXPC, and MAXPE. Increasing DBODY increases all four outputs. These results are to be expected and can be shown through aerospace performance equations.

2.4 12 Class Data Set

For robustness, a second data set consisting of 12 different classes of missiles was developed, again primarily differentiated by the inputs DBODY, THROAT, and FINENESS. A total of $36,000$ runs were generated, or $3,000$ per class. Table 2.2 shows the 3 discriminating input values for each of the 12 classes.

Table 2.2: 12 Class Data Set Mean Value for Discriminating Inputs

Class	Body Diameter (DBODY)	Fineness Ratio (FINENESS)	Throat Diameter (THROAT)
1	0.70	14.3337	0.250
2	0.70	15.7671	0.250
3	0.70	14.3337	0.325
4	0.70	15.7671	0.325
5	0.75	14.3337	0.250
6	0.75	15.7671	0.250
7	0.75	14.3337	0.325
8	0.75	15.7671	0.325
9	0.80	14.3337	0.250
10	0.80	15.7671	0.250
11	0.80	14.3337	0.325
12	0.80	15.7671	0.325

There are 3 different values for DBODY, with an added noise of $+/-0.0075$ meters. FINENESS has 2 different values and an added noise of $+/-0.1434$. THROAT has 3 different values and an added noise of $+/-0.0029$ meters. The box plots below (Figures 2.9, 2.10, 2.11) help visualize these inputs with their respective noise.

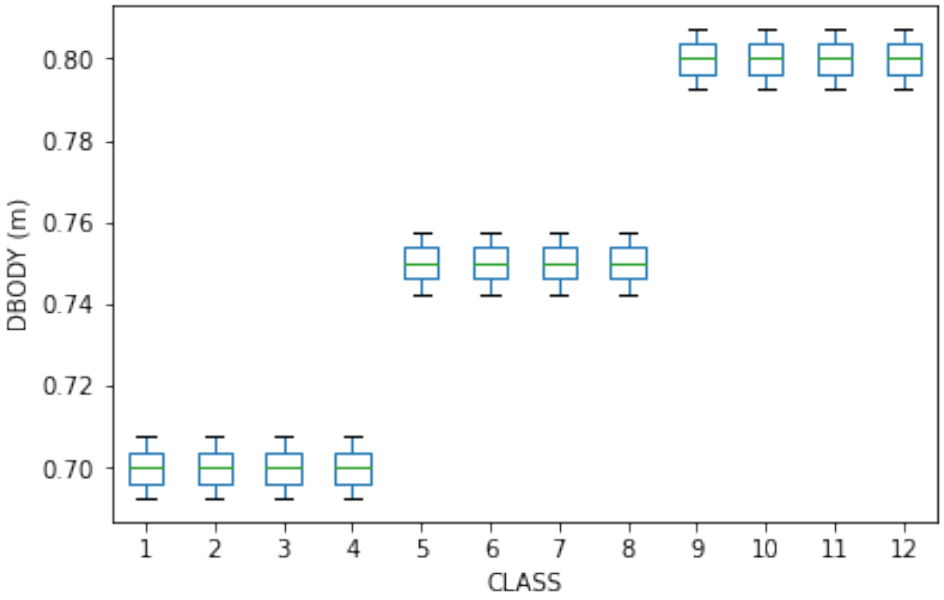


Fig. 2.9: DBODY Box Plots for Each Class (12 Class)

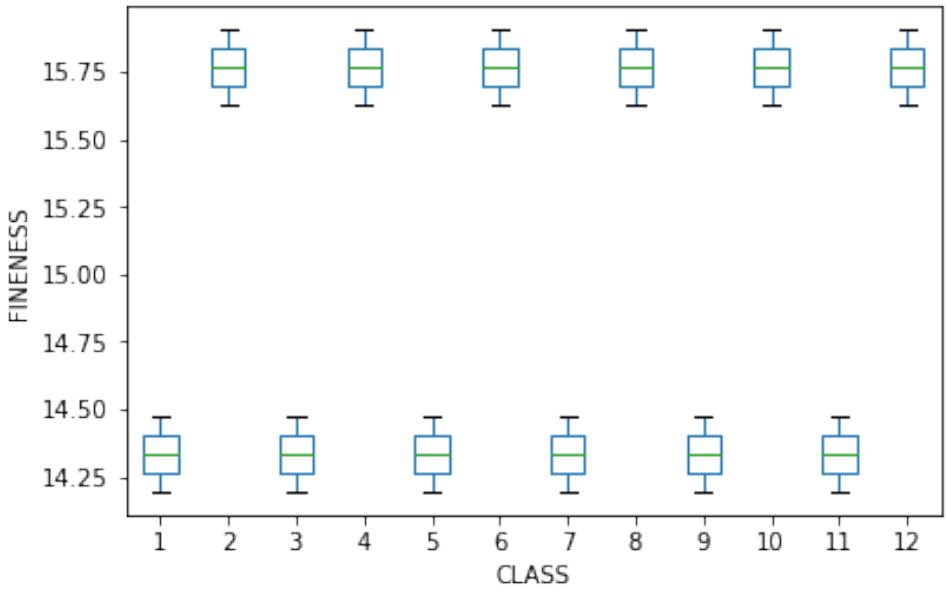


Fig. 2.10: FINENESS Box Plots for Each Class (12 Class)

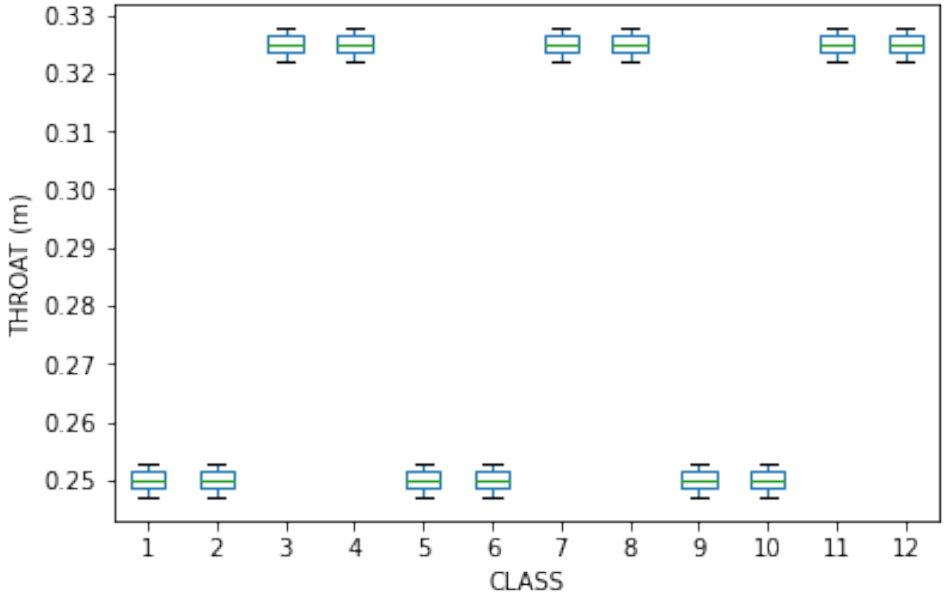


Fig. 2.11: THROAT Box Plots for Each Class (12 Class)

The outputs used as features differ in this data set compared to the '18 Class Data Set'. This data set uses pure trajectory data consisting of position and velocity data in the X and Z directions. This data is recorded once every 0.4 seconds for a total of 30 seconds. Using this time step, other useful features can be calculated and used, such as acceleration or curve fits on a position versus time plot. Using pure trajectory data versus derived features is a topic of

interest and will be demonstrated and compared using this data set in the '12 Class Data Set' results section (Section 5.2.1).

Figure 2.12 shows a randomly selected trajectory from each of the 12 classes. There is a lot of overlap between classes early on but diverge later in their respective flights. This single plot is not indicative of each class as a whole but just provides a visual for how each one might fly.

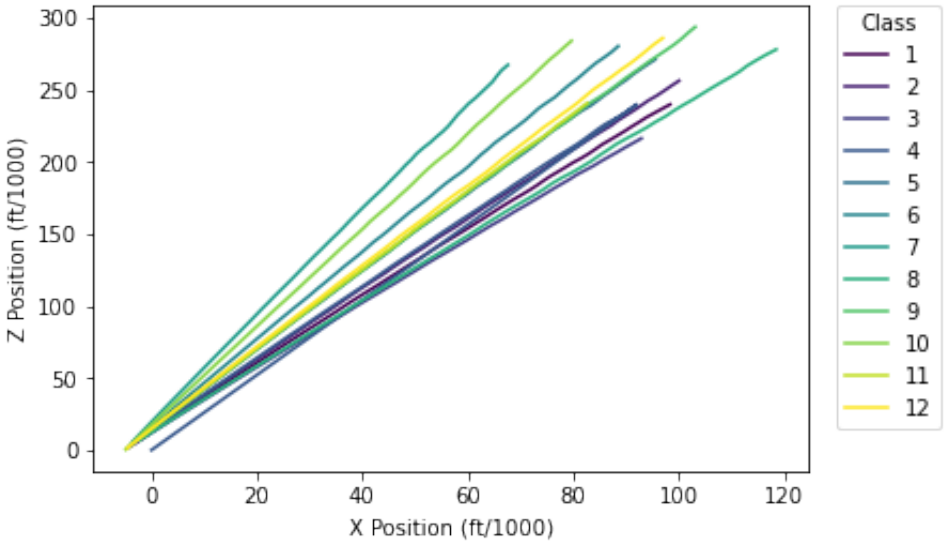


Fig. 2.12: Sample Trajectory from Each Class (12 Class)

Figures 2.13 and 2.14 give a better representation of how each class is performing. The maximum position and velocity for a sample of 150 runs per class were found and plotted against each other to better visualize the overlap between classes. This overlap stress tests the developed hybrid method and is the reason this data set is used to test robustness. A time study (Section 5.2.1) and a missing data study (Section 5.2.1) were used to further test the hybrid method on this data set by dropping time steps across the trajectory to determine where the limits for the algorithm are as well as to gain insight on the what portion of the trajectory is most discriminating.

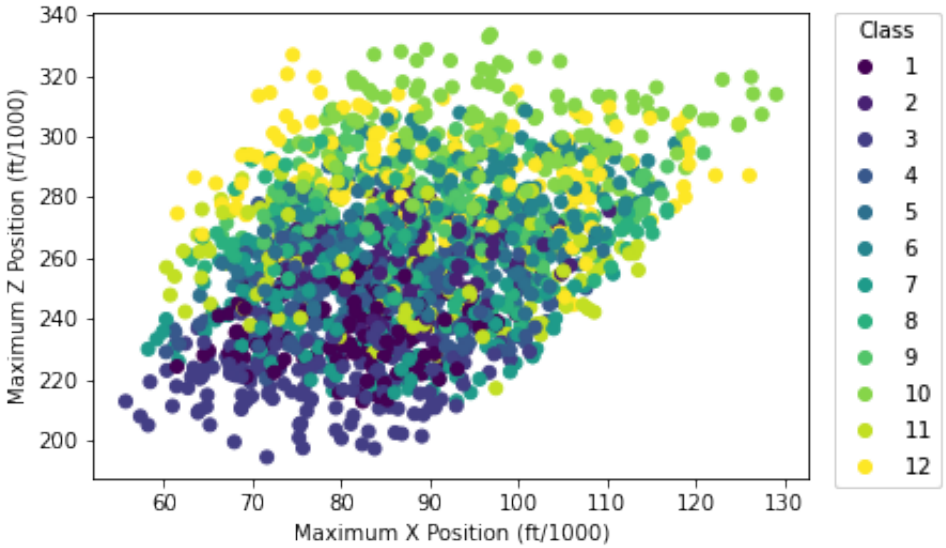


Fig. 2.13: 150 Run Sample Maximum Positions for Each Class (12 Class)

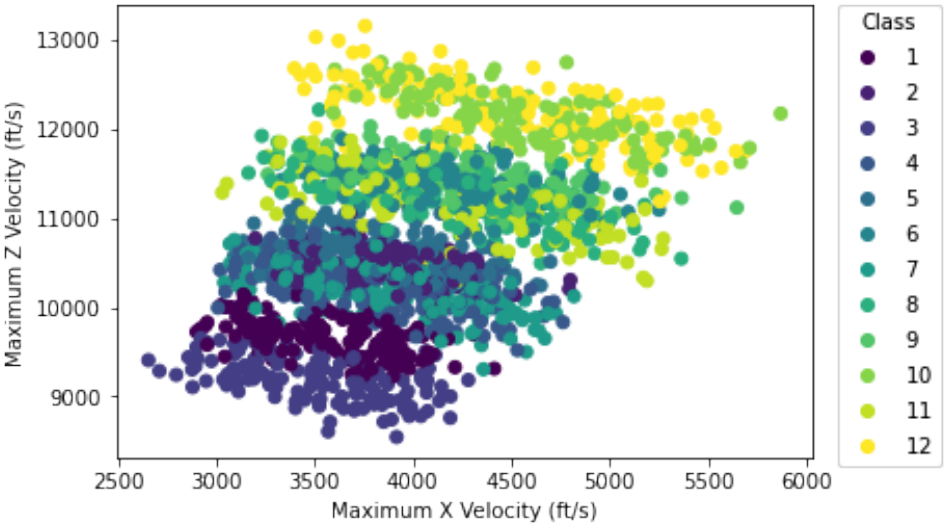


Fig. 2.14: 150 Run Sample of Maximum Velocities for Each Class (12 Class)

2.5 Ballistic/Maneuverable Data Set

This data set consists of 2 different classes of missiles, one built as ballistic missile and one built as a maneuverable missile with adjustable fins. The two classes of missiles are identical except for that the maneuverable missiles have actuators to manipulate the fins. These actuators add weight, which is the primary discriminating input between these two classes. A total of $2,000$ runs were generated, or $1,000$ per class.

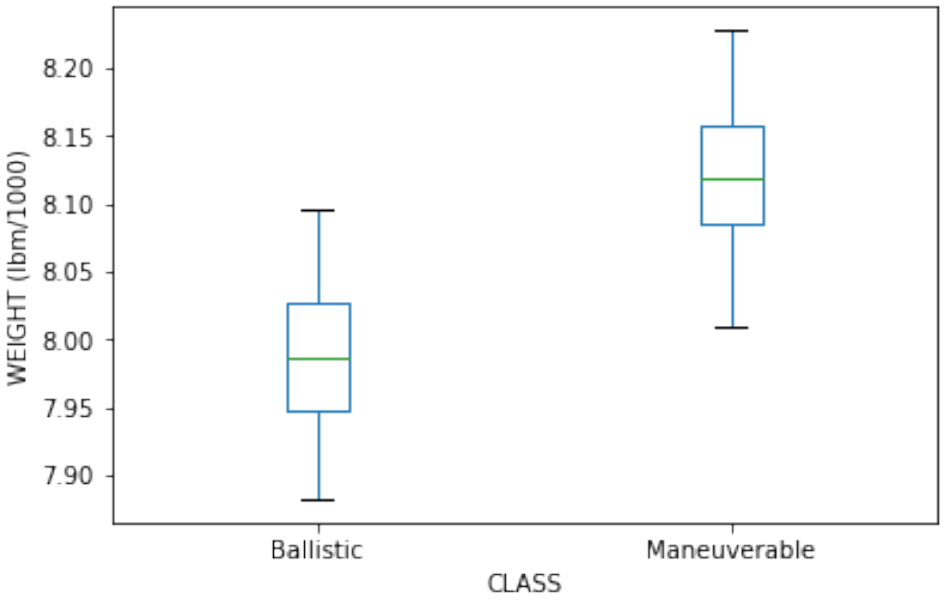


Fig. 2.15: WEIGHT Box Plots for Each Class (B/M)

While each class produces the same amount of thrust, this small weight discrepancy leads to different accelerations and velocities, even early in flight, primarily in the Z direction. The outputs used as features are position and velocity data in 2 dimensions (X,Z), recorded once every 0.4 seconds until the missile completes its flight and hits the ground, which occurs roughly between $450 - 475$ seconds. Figure 2.16 below shows the maximum velocity in the vertical direction (VZMAX) for each class.

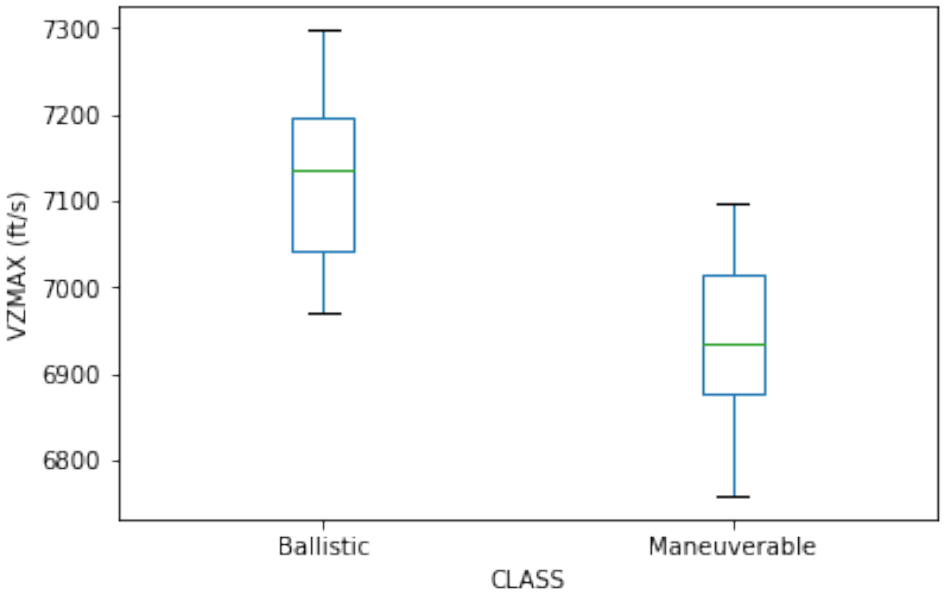


Fig. 2.16: VZMAX Box Plots for Each Class (B/M)

Figure 2.17 below shows a sample trajectory for a ballistic and a maneuverable missile through the first 9.6 seconds of flight, amounting to 24 total observation points (1 observation every 0.4 seconds). Note that the missiles have a steep launch angle and are flying almost vertically.

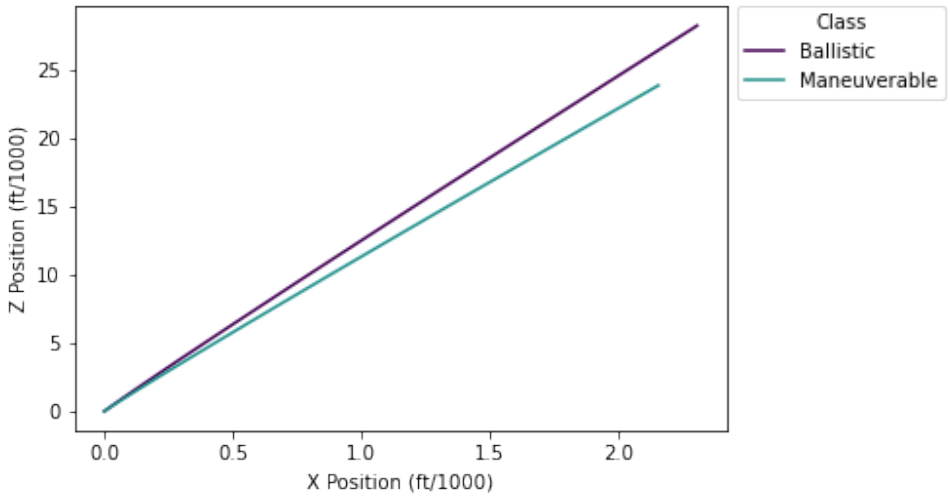


Fig. 2.17: Sample Trajectory for Each Class (B/M)

The extra weight in the maneuverable missile detracts from its climbing ability, as shown in its VZMAX and the respective heights in their sample flights. Even early on the height difference is noticeable to the eye. The goal for this data set is to identify the maneuverable missile as early as possible, the results of which are shown in the 'Ballistic/Maneuverable Data Set' results (Section 5.2.2). These flights are easily distinguishable to the naked eye, but when the time scale is cut down to just a few seconds it becomes more ambiguous, also noting that this is a sample flyout and does not represent the entire class. The flight envelope for the missiles overlap, they are not completely distinct.

Chapter 3

Machine Learning Algorithms

3.1 One-Class Support Vector Machine

The Support Vector Machine (SVM) is a machine learning algorithm that finds the maximum margin hyperplane that separates two data sets. The margin is the minimum distance from the hyperplane to the nearest point in either class. The SVM is a binary classification algorithm, defining each point to one of the two input classes. The One-Class SVM (OCSVM) is one of the many different SVM models built from the original. The OCSVM is still a binary classification algorithm, but instead of requiring two data sets to define a separating hyperplane, it only requires one 'clean' (known) class and no adversarial data. The binary classification then becomes either belonging to the trained-on class or not. This is possible by setting hyperparameters which are tuned according to the intricacies of the data set and the needs of the user, and by setting a kernel to define the shape of the hyperplane. This attribute makes the OCSVM a popular algorithm for novelty detection and is considered one of the state-of-the-art methods.

3.1.1 SVM Theory

The SVM is built as an extension of the Perceptron, one of the earliest machine learning algorithms developed. The Perceptron is a machine learning algorithm that will find a hyperplane that separates two data sets if this separation is possible. This algorithm stops once it finds a separating hyperplane and if the data sets are linearly separable, there are infinite separating hyperplanes, shown in Fig. 3.1. It should be noted that figures and the notation throughout the linear SVM portion are modeled after a lecture from a machine learning course at Cornell

University [2]. This lecture provides a good understanding of how SVMs work and the theory behind them.

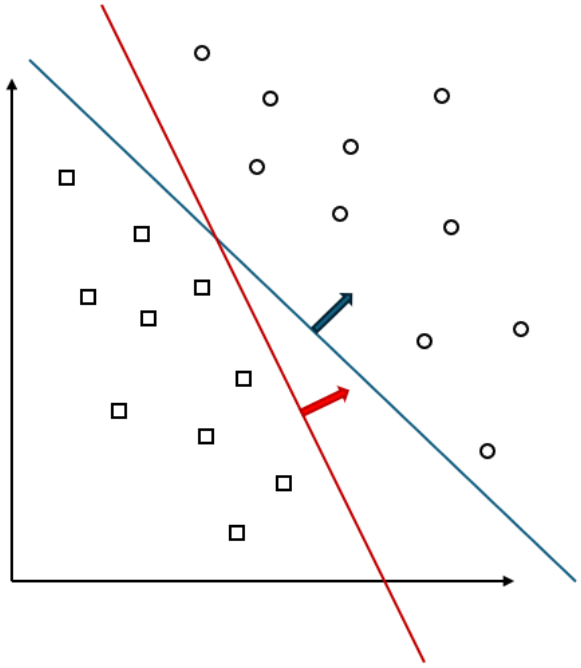


Fig. 3.1: Example Hyperplanes that Separate the Same Data Sets

The linear SVM extends off this principle by introducing a margin, γ, defined as the distance from the hyperplane to the nearest point from either class, and a hyperplane normal vector, $\mathbf{w}$. By using an optimization routine, this margin can be maximized to provide the most robust and accurate separating hyperplane, shown in Fig. 3.2. Using the largest margin makes the decision boundary less suceptible to noise and perturbations in the data [5]. To perform this, a linear classifier is used of the form $h(\mathbf{x}) = sign(\mathbf{w}^T\mathbf{x} + b)$ with the aforementioned binary classification defined with labels $\{+1, -1\}$.

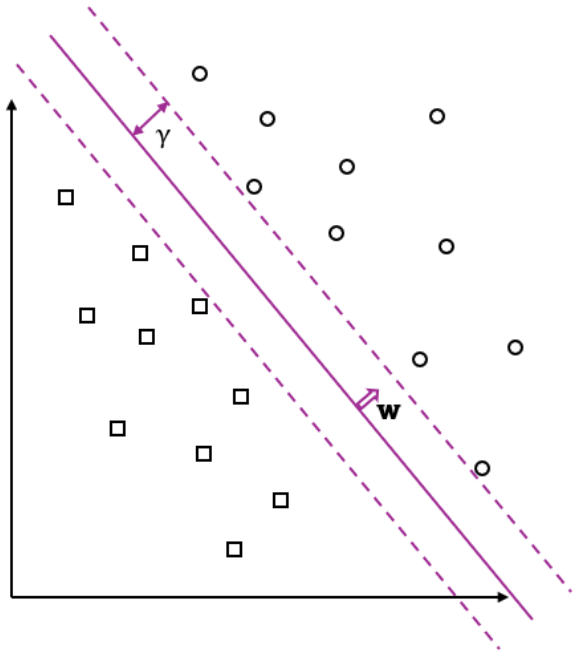


Fig. 3.2: Maximum Margin Separating Hyperplane

The hyperplane, H, is defined by $\mathbf{x}$ and b, representing a set of points such that $H = \{\mathbf{x}|\mathbf{w}^T\mathbf{x} + b = 0\}$. The distance then of a point (x) to the hyperplane (H) can be defined by Eq. 3.1. Figure 3.3 shows the distance (**d**), the sample point (x), the projected point (x_p), the hyperplane (H), and the hyperplane normal vector (**w**) on an example plot for visualization.

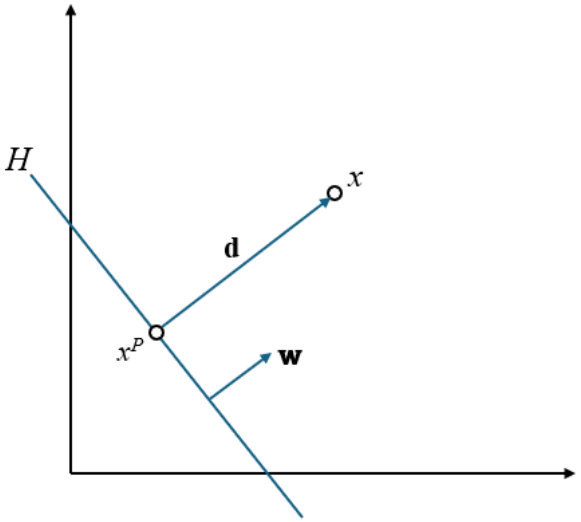


Fig. 3.3: SVM Distance Example

$$\|\mathbf{d}\|_2 = \sqrt{\mathbf{d}^T\mathbf{d}} = \frac{|\mathbf{w}^T\mathbf{x} + b|}{\sqrt{\mathbf{w}^T\mathbf{w}}} = \frac{|\mathbf{w}^T\mathbf{x} + b|}{\|\mathbf{w}\|_2} \tag{3.1}$$

It follows then that the margin for this hyperplane is the shortest distance to a point from both classes. The margin (γ) of the hyperplane (H) with respect to the matrix of distances (D) is defined as Eq. 3.2.

$$\gamma(\mathbf{w}, b) = min_{\mathbf{x}\in D}\frac{|\mathbf{w}^T\mathbf{x} + b|}{\|\mathbf{w}\|_2} \tag{3.2}$$

As stated before, for the most robust decision boundary the hyperplane with the largest separating margin is found and used. This is done such that there are no negative values across the entirety of both class instances, indicating that the hyperplane is a separating hyperplane and all the points are on the correct side. The problem then becomes an optimization routine to find the maximum, shown in Eq. 3.3.

$$max_{w,b}\frac{1}{\|\mathbf{w}\|_2}min_{\mathbf{x}_i\in D}|\mathbf{w}^T\mathbf{x}_i + b| \text{ such that } \forall i\ y_i(\mathbf{w}^T x_i + b) \geq 0 \tag{3.3}$$

Through rescaling, a simpler formation can be shown for the optimization problem. This formation, shown in Eq. 3.4, is a quadratic optimization problem that has one unique solution given that a separating hyperplane exists.

$$min_{w,b}\mathbf{w}^T\mathbf{w} \text{ such that } \forall i\ y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \tag{3.4}$$

From this optimization routine the resultant $\mathbf{w}$ and b for the maximum margin separating hyperplane will be defined. The points which are closest to this hyperplane are called the

support vectors and they define the hyperplane as well as the margin. The hyperplane acts independently of points that are not support vectors [4].

In cases where the data sets cannot be separated, soft constraints can be used. This allows points to be 'sacrificed' and will fall on the wrong side of the hyperplane to maximize the margin and provide higher overall accuracy. This can be seen below in Fig. 3.4 where there is one green point on the incorrect side of the hyperplane, as well as a red point inside the margin. This hyperplane, with a soft constraint error factor (ν) of 0.1, was made using Ref. [6]. The slack parameters ξ and C can be introduced as a constant in the optimization routine, updated and shown in Eq. 3.5. The parameter ξ is the minimized variable according to C which sets the error rate, or the fraction of points which can be in error.

$$min_{w,b}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i \;\text{ such that }\; \forall i\; y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \;\; \forall i\; \xi_i \geq 0 \tag{3.5}$$

This allows the SVM to find the optimal separating line that provides the highest classification accuracy by minimizing the slack variable (ξ) contribution. The routine will allow points to invade the separating margin or even points to be on the incorrect side if the line leads to a simpler solution.

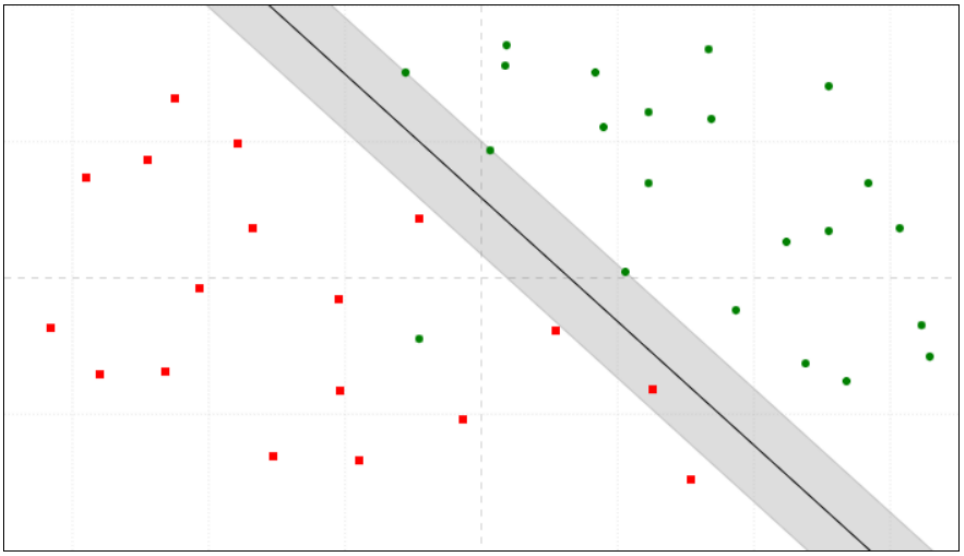


Fig. 3.4: Linear SVM with Soft Constraints

Using a kernel, the data can be non-linearly transformed before fitting a linear SVM to the data. In the original feature space this makes the boundary non-linear and more robust to difficult datasets. The non-linear transformation used is the Radial Basis Function (RBF) kernel. The linear solution function takes the form of $h(x) = sign(\mathbf{w}^T\mathbf{x} + b)$ while the RBF kernel solution function takes the form of Eq. 3.6, pulled from Ref. [3]. This kernel function also includes α, a value resulting from introducing soft constraints. The function depends on a matrix of data points (x), a matrix of input vectors (x'), and the kernel learning rate coefficient (γ_{RBF}).

$$h(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i K(x, x')_{RBF} + b \quad \text{where} \quad K(x, x')_{RBF} = exp(-\gamma_{RBF}\|x - x'\|^2) \tag{3.6}$$

Figure 3.5 shows a data set that is not linearly separable. The result is a very inaccurate linear SVM hyperplane. Figure 3.6 shows the same data set but using an RBF kernel to define the hyperplane, resulting in a perfectly accurate non-linear hyperplane. In these figures, the instances that have blue circles are the support vectors. These figures were made using Ref. [6].

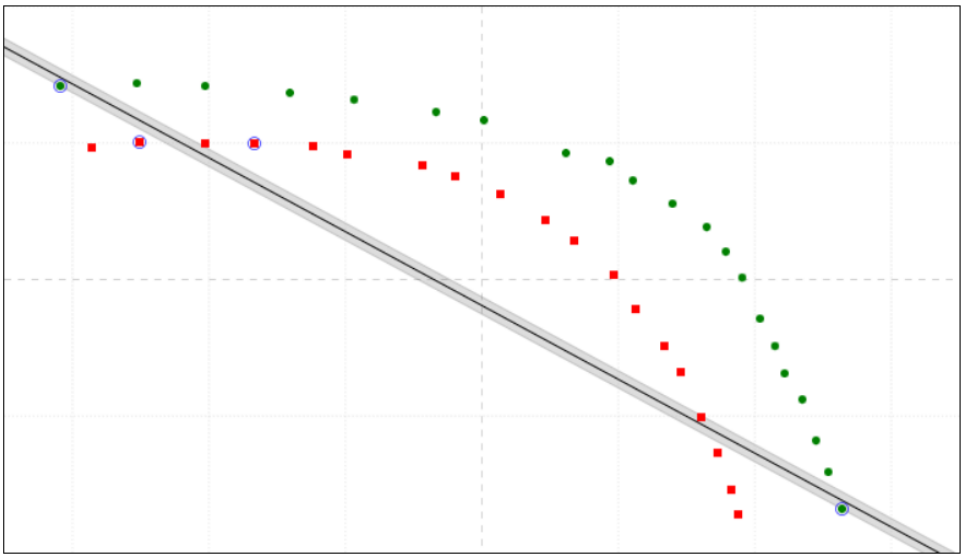


Fig. 3.5: Linear SVM Example

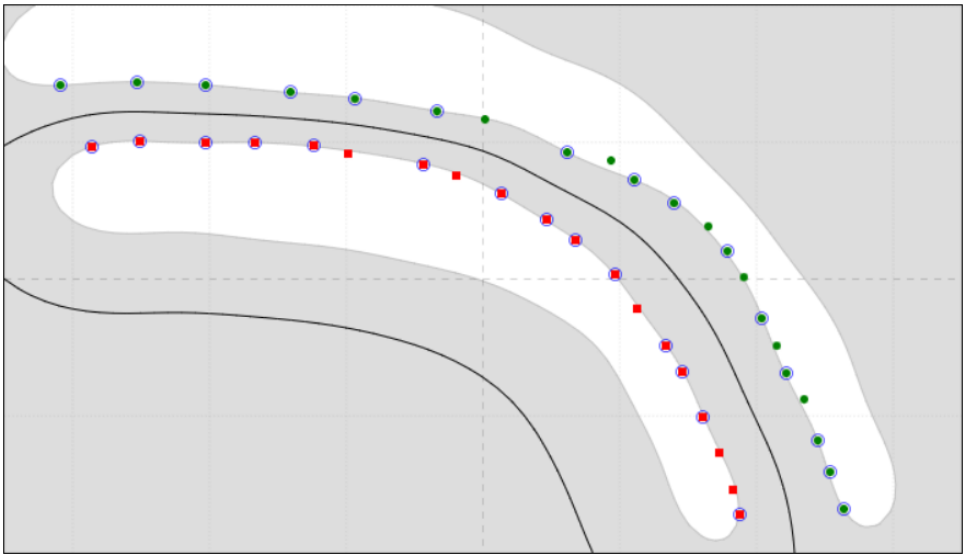


Fig. 3.6: RBF SVM Example

For a more in-depth explanation of SVMs, one should refer to Refs. [2], [3], [4], and [5]. These lectures and textbooks provide comprehensive explanations and examples of SVMs. The theory above is meant to give a general understanding of how a binary SVM works. A One-Class SVM works much in the same way, except that it only needs one class to train on, with the length of the support vectors and slack parameters determined by the input hyperparameters and the shape of the hyperplane determined by the input kernel.

3.1.2 ”OneClassSVM” Algorithm

The algorithm ”OneClassSVM”, Ref. [7], from the *scikit-learn* python library is utilized in this paper. The definitions and explanation below are pulled from Ref. [7] as well. Hence the name, it is an unsupervised OCSVM algorithm implemented by inputting your hyperparameters and a kernel, then fitting it to a clean class. New data can then be run through, and the OCSVM will judge each instance and give it a score that determines if it is an inlier or outlier. All aspects are explained in depth below throughout this section.

Kernel

This function defines the kernel type used in the analysis. As stated above, shown in Eq. 3.6, the RBF kernel is used.

Radial Basis Function: $K(x, x') = exp(-\gamma_{RBF}\|x - x'\|^2)$

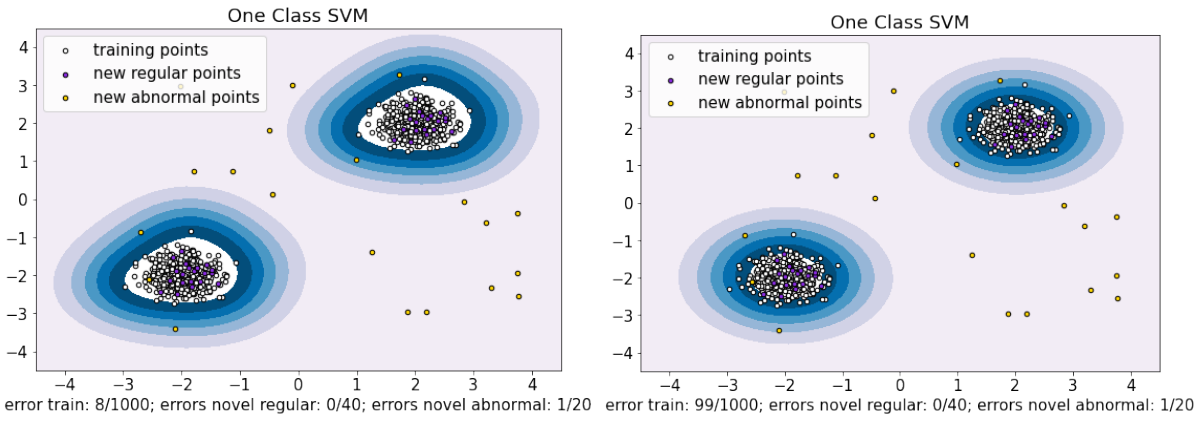
This kernel is used for complex data sets and is known for being robust and returns a smooth but intricate hyperplane. The RBF kernel depends only on the radial distance from the center [4] of the data set to form the hyperplane.

Hyperparameters

While "OneClassSVM" has several hyperparameters, only three are explicitly defined for this research while the rest were left to their default settings. These three are *nu*, *gamma*, and *tol*. Proper hyperparameter tuning is essential for an accurate analysis and is the difference between a successful run and a failure. The definitions below were pulled from the "OneClassSVM" page on the *scikit-learn* website [7].

1. nu: An upper bound on the fraction of training errors and a lower bound on the fraction of support vectors. Should be in the interval $(0, 1]$.

This hyperparameter is found in the SVM theory (Section 3.1.1) equations as C.

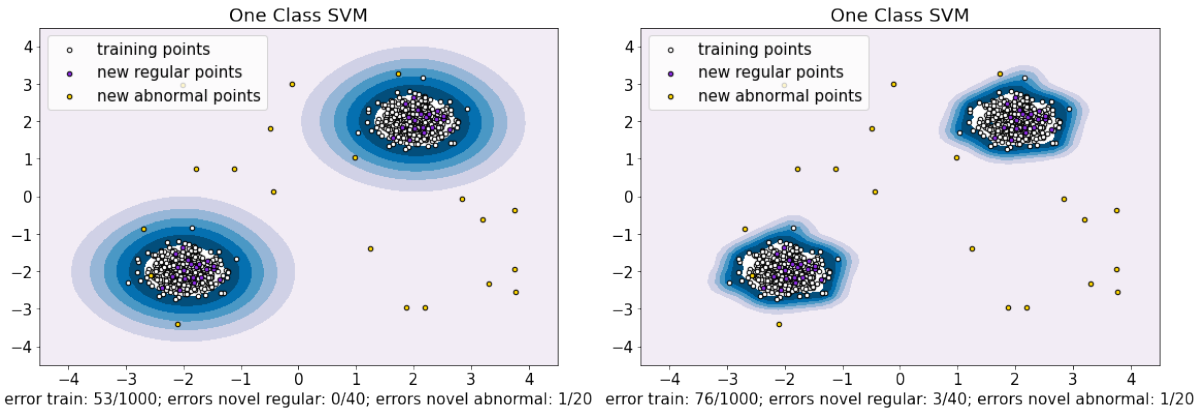


(a) nu = 0.01 (b) nu = 0.1

Fig. 3.7: Effect of 'nu' on OCSVM Boundary

Figure 3.7 shows how 'nu' affects the OCSVM boundary. A lower value of 'nu' produces a larger boundary with a shallow gradient. Remember that the value of 'nu' is the upper bound of training errors, that can be shown by the 'error train' fraction in the bottom left. In this case, changing the 'nu' hyperparameter does not change the error rate in regular or abnormal points, but is very important and has drastic effects in more difficult scenarios.

2. gamma: Kernel coefficient for 'rbf'.

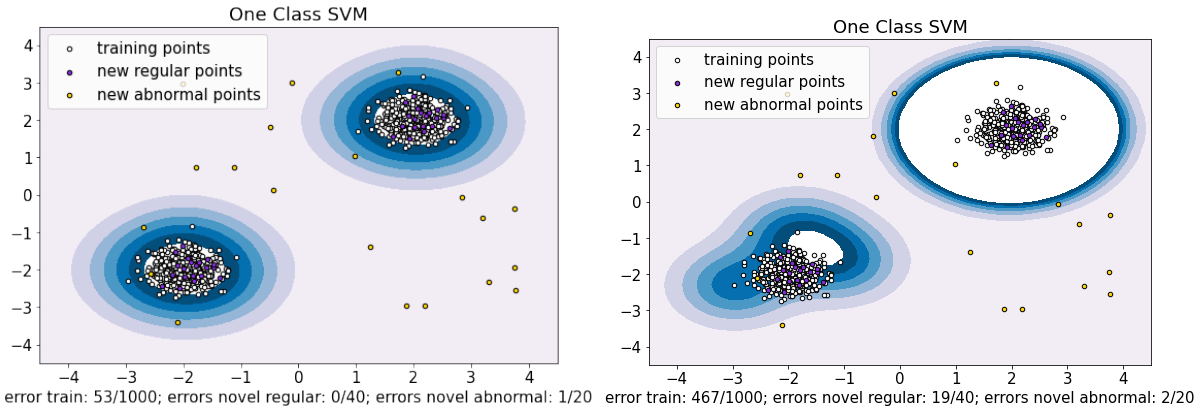


(a) gamma = 1 (b) gamma = 10

Fig. 3.8: Effect of 'gamma' on OCSVM Boundary

Figure 3.8 shows how 'gamma' affects the OCSVM boundary. The hyperparameter 'gamma' is shown in the SVM theory (Section 3.1.1) as γ_{RBF} and is the learning rate parameter for the RBF kernel. If it is too high, the algorithm doesn't properly converge on a solution, as shown when 'gamma' equals 10 and a rough boundary is returned. For data sets with many features, 'gamma' needs to be small to produce accurate results. Take note of the increase in error rates in train and regular points when 'gamma' is increased between the two runs.

3. tol: Tolerance for stopping criterion.



(a) tol = $1e^{-3}$ (b) tol = 50

Fig. 3.9: Effect of 'tol' on OCSVM Boundary

Figure 3.9 shows how the OCSVM boundary changes as 'tol' changes. Since this data set is a simple 2D example, the 'tol' increase had to be exaggerated to give a good representation of its influence. If the convergence tolerance is too small, overfitting will occur, while if the tolerance is too large, training errors are increased because the algorithm's optimization routine stops too early.

Properly tuning these three hyperparameters is essential to creating a smooth, accurate decision boundary, especially in difficult data sets with many features.

Score Samples

The "OneClassSVM" algorithm has a feature called *score_samples* which is a built-in raw scoring function of the samples. It gives a score to each instance and that score is used in determining if a point is an inlier or an outlier. For example, Fig. 3.10 shows an OCSVM scenario where 'nu' = 0.05, 'gamma' = 1, and 'tol' = $1e^{-3}$. Table 3.1 shows the resultant stats for the score samples given to the train (error train), test (errors novel regular), and outliers (errors novel abnormal) data points.

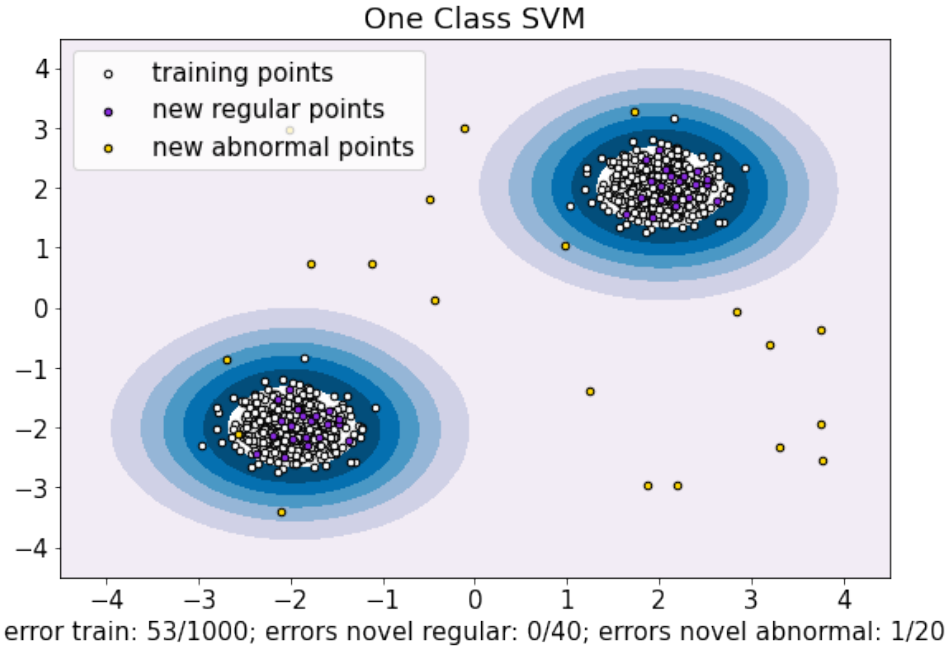


Fig. 3.10: Score Sample SVM Example

Table 3.1: Score Sample Stats for Each Data Set

	Train	Test	Outliers
Count	1000	40	20
Mean	12.15	12.19	1.73
std	0.794	0.595	3.09
min	7.27	11.06	$\sim$0.00
max	12.95	12.94	11.30

Checking the error rates in Fig. 3.10 supports what we can observe from the score samples from the data sets. The training points received scores ranging from 7.27-12.95, which defines the inlier score range. The range for the test points was 11.06-12.94, and the OCSVM had zero errors in identifying these points, because their scores all fell inside of the inlier score range. The outlier points had a score range from 0.00-11.30. The OCSVM had one error in identifying outlier points, making the mistake on the point that fell inside the range for the training points and received a score of 11.30, well inside of the inlier range.

The score samples prove important in identifying novelties and in multi-class OCSVM classification. How they are used is laid out in depth later in the OCSVM portion of the Developed Methods chapter (Section 4.2).

3.2 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a generalization of Fisher's Linear Discriminant and finds linear combinations of features that characterize and separate two or more classes. It is a fully supervised method, meaning it can utilize class labels to maximize variance between classes while also minimizing variance within each class. It also acts as a feature reduction method by learning the most discriminating axes between classes and dropping the ones that are negligible. After transforming the data based on these most discriminating axes, the LDA then defines hyperplanes that separate the various classes.

3.2.1 LDA Theory

The LDA theory and nomenclature in this section is taken from the *Elements of Statistical Learning* textbook [3], used for the feature space explanation and equations, and the *Probabilistic Machine Learning* textbook [5], used for the feature reduction explanation and equations.

LDA is used in a case when it is assumed that the classes all have a common covariance matrix ($\mathbf{\Sigma}_k = \mathbf{\Sigma}\ \forall k$) and the classes conditional densities are multivariate Gaussians. When dealing with aerospace vehicles this is a safe assumption, as each one should have a multivariate Gaussian distribution due to its flight window and other limitations in performance. The log-ratio between two classes (*k* and *l*), shown in Eq. 3.7, is linear in *x*, meaning that the decision boundary is also linear in *x*. Equation 3.7 can be thought of as the difference of distances to each class's centroid. If the feature space is divided into N regions, each classified to different classes, these regions will be separated by linear hyperplanes and each region will represent the class that has the highest probability for that space based on the log-posterior of the fit model.

$$log\frac{Pr(G=k|X=x)}{Pr(G=l|X=x)} = log\frac{\pi_k}{\pi_l} - \frac{1}{2}(\mu_k+\mu_l)^T\mathbf{\Sigma}^{-1}(\mu_k-\mu_l) + x^T\mathbf{\Sigma}^{-1}(\mu_k-\mu_l) \tag{3.7}$$

From Eq. 3.7, the linear discriminant functions, shown in Eq. 3.8, are the description of the decision rule, where $G(x) = argmax_k\delta_k(x)$.

$$\delta_k(x) = x^T\mathbf{\Sigma}^{-1}\mu_k - \frac{1}{2}\mu_k^T\mathbf{\Sigma}^{-1}\mu_k + log\pi_k \tag{3.8}$$

The training data is used to estimate the parameters for the Gaussian distribution for each class.

$$\vec{\pi}_k = \frac{N_k}{N}, \text{ where } N_k \text{ is the number of class-k observations} \tag{3.9}$$

$$\vec{\mu}_k = \sum_{g_i=k} \frac{x_i}{N_k} \tag{3.10}$$

$$\vec{\mathbf{\Sigma}} = \frac{\sum_{k=1}^{K} \sum_{g_i=k} (x_i - \vec{\mu}_k)(x_i - \vec{\mu}_k)^T}{N - K} \tag{3.11}$$

The LDA is implemented by projecting the data with respect to the most discriminatory axes. Instances can be classified to the closest class centroid in this transformed space. To find these axes, let $m_k = \mathbf{w}^T \mu_k$ be the projection of each class's mean (μ_k) onto the line ($\mathbf{w}$) and let $z_n = \mathbf{w}^T \mathbf{x}_n$ be the projection of the data ($\mathbf{x}_n$) onto the line ($\mathbf{w}$). The variance of these points is defined by Eq. 3.12.

$$s_k^2 = \sum_{y_n=k} (z_n - m_k)^2 \tag{3.12}$$

As stated before, the LDA is fully supervised, meaning it can utilize class labels. Using this information the goal then becomes to maximize Eq. 3.13, by maximizing the distance between the class means [$(m_2 - m_1)^2$] and minimizing the variances of each class [$(s_1^2 + s_2^2)$].

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} \tag{3.13}$$

Extending this principle to several classes is very similar. The goal of the LDA is to reduce the number of dimensions in the features ($\mathbf{x} \in \mathbb{R}^D$) by transforming the data to the resulting

low-dimension features ($\mathbf{z} \in \mathbb{R}^K$). This is done using a linear projection matrix ($\mathbf{z} = \mathbf{W}\mathbf{x}$) where $\mathbf{W}$ is a KxD matrix.

Two scatter matrices can be defined as Eqs. 3.14 and 3.15, where $\vec{\mathbf{S}}_B$ is the between-class scatter matrix, and $\vec{\mathbf{S}}_W$ is the within-class scatter matrix. The variable $\mathbf{m}_c$ is the mean for class 'c', while $\mathbf{m}$ ($\mathbf{m} = \frac{1}{N}\sum_{c=1}^{C} N_c\mathbf{m}_c$) is the overall mean.

$$\vec{\mathbf{S}}_B = \sum_{c=1}^{C} N_c(\mathbf{m}_c - \mathbf{m})(\mathbf{m}_c - \mathbf{m})^T \tag{3.14}$$

$$\vec{\mathbf{S}}_W = \sum_{c=1}^{C} \sum_{y_n=c} (\mathbf{z}_n - \mathbf{m}_c)(\mathbf{z}_n - \mathbf{m}_c)^T \tag{3.15}$$

Equation 3.13 can be rewritten in terms of $\mathbf{W}$ by inserting Eqs. 3.14 and 3.15 in. The goal can now be defined as maximizing Eq. 3.16.

$$J(\mathbf{W}) = \frac{|\mathbf{W}^T\mathbf{S}_B\mathbf{W}|}{|\mathbf{W}^T\mathbf{S}_W\mathbf{W}|} \tag{3.16}$$

The solution of Eq. 3.16 can be shown as Eq. 3.17, where $\mathbf{U}$ are the K leading eigenvectors of $\mathbf{S}_W^{-\frac{1}{2}}\mathbf{S}_B\mathbf{S}_W^{-\frac{1}{2}}$.

$$\mathbf{W} = \mathbf{S}_W^{-\frac{1}{2}}\mathbf{U} \tag{3.17}$$

3.2.2 LinearDiscriminantAnalysis Algorithm

The algorithm "LinearDiscriminantAnalysis", Ref. [8], from the *scikit-learn* python library is utilized in this paper. The algorithm is a is a fully supervised LDA which provides high accuracy classification. Since the LDA uses a log-posterior probability function do determine

class, it classifies points based on what class they score highest for. This means this algorithm cannot be used alone for novelty detection because each point will be 'forced' into a class. The solution to this issue is described in depth later in the Developed Methods chapter (Section 4). In short, the OCSVM is used as a novelty detector and the LDA is used for increased classification accuracy, exploiting the strengths of both individual algorithms.

Components

The number of components can be defined in the algorithm. The components are linear combinations of features, the most discriminatory of which will be used to project the data into a new, low-dimensional feature space. The definition for the hyperparameter that controls this is defined below, taken from Ref. [8].

n_components: Number of components ($\leq\ min(n\_classes-1, n\_features)$) for dimensionality reduction

Figure 3.11 is a two component LDA projection of the commonly used Iris data set. The source code for this example came from the *scikit-learn* website (Ref. [13]).

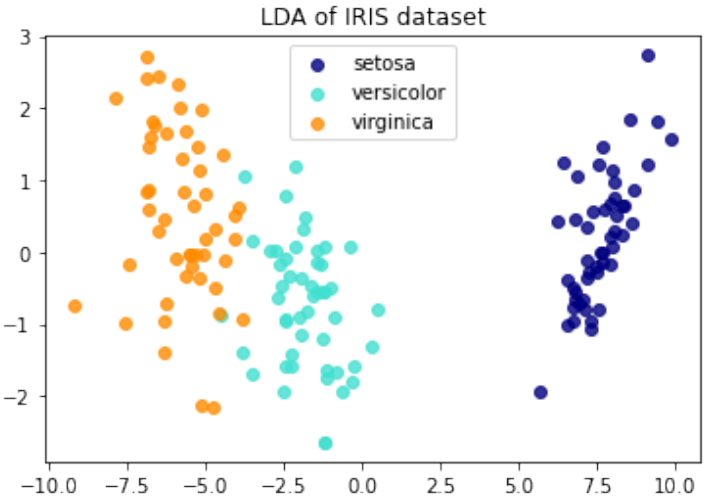


Fig. 3.11: LDA Projection of Iris Data Set

3.3 Principal Components Analysis

Principal Components Analysis (PCA) is a popular method for feature reduction. The PCA finds the most discriminating projections or linear combinations of the data, called principal

components, that provide the maximum variance for the data set. The number of principal components needed to properly capture the variance of the data set can be defined and used to transform the data set, reducing the number of features down to this number. While the PCA is very similar to an LDA, it is unsupervised so it can only maximize variance through the data set as a whole, it cannot use class labels to maximize variance between classes or minimize variance within a class.

How the PCA is applied is laid out in detail in the Developed Methods chapter (Section 4). The OCSVM is subject to the 'curse of dimensionality' [3], where noisy or negligible features lead to high run times and inaccuracies. Using a PCA to preprocess the data increases the accuracy and effectiveness of the OCSVM [14].

3.3.1 PCA Theory

The PCA theory and nomenclature in this section was pulled from Ref. [4], *Pattern Recognition and Machine Learning* by Christopher M. Bishop. For a data set with N observations, the goal of a PCA is to project the data set into D dimensions where $D < N$ while maximizing the variability in the projection. The mean of the original data is given by Eq. 3.18 and the mean of the projected data is $\mathbf{u}_1^T\overline{\mathbf{x}}$, where $\mathbf{u}_1$ is a vector with D dimensions that defines the direction of the projection. The variance of the projected data is given by Eq. 3.19.

$$\overline{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n \tag{3.18}$$

$$\frac{1}{N}\sum_{n=1}^{N}\{\mathbf{u}_1^T\mathbf{x}_n - \mathbf{u}_1^T\overline{\mathbf{x}}\}^2 = \mathbf{x}_1^T\mathbf{S}\mathbf{u}_1 \tag{3.19}$$

Where **S** above is the covariance matrix for the data, given by Eq. 3.20.

$$\mathbf{S} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \overline{\mathbf{x}})(\mathbf{x}_n - \overline{\mathbf{x}})^T \tag{3.20}$$

To maximize the variance with respect to $\mathbf{u}_1$, a constraint is introduced in the form of the normalization condition $\mathbf{u}_1^T\mathbf{u}_1 = 1$. A Lagrange multiplier (λ_1) is introduced to enforce this condition. Through further analysis, Bishop (Ref. [4]) shows that $\mathbf{u}_1$ must be an eigenvector of $\mathbf{S}$. This then gives the variance as shown by Eq. 3.21.

$$\mathbf{u}_1^T\mathbf{S}\mathbf{u}_1 = \lambda_1 \tag{3.21}$$

The eigenvector ($\mathbf{u}_1$) with the largest eigenvalue (λ_1) will have the maximum variance. This eigenvector is called the first principal component. The principal components are arranged in order of eigenvalues, allowing the original data set to be projected in multiple dimensions in the directions of maximum variance.

3.3.2 "PCA" Algorithm

The algorithm "PCA", Ref. [9], from the *scikit-learn* python library is utilized in this paper. The algorithm is an unsupervised PCA used as a feature reduction method and to increase accuracy by maximizing the variability in the data. This becomes useful when processing data to feed into the OCSVM, since it does not include weights in the process and suffers from the 'curse of dimensionality' [3], and is shown to increase OCSVM accuracy [14]. This is especially important to missile classification due to having redundant, highly correlated features in the trajectory. An example can be seen in the X position data, where x_1 is very highly correlated to x_2, x_2 is very highly correlated to x_3, and so on.

Components

Similar to the implementation of the LDA algorithm, the components are the only variable explicitly defined by the user. The definition of this variable is shown below, pulled from [9].

n_components: Number of components to keep. If $n\_components$ is not set all components are kept.

The user can tune the number of principal components kept according to the data set. Relatively few dimensions can account for $> 95\%$ of the total variability in the data set. After the PCA is run on the data set, the $transform$ function takes the original data set and projects it according to the principal components.

Figure 3.12 shows a two component PCA projection of the commonly used Iris data set. The first two principal components account for 97.76% of the explained variance in the data set. The source code to produce this plot was pulled from the *scikit-learn* website [13].

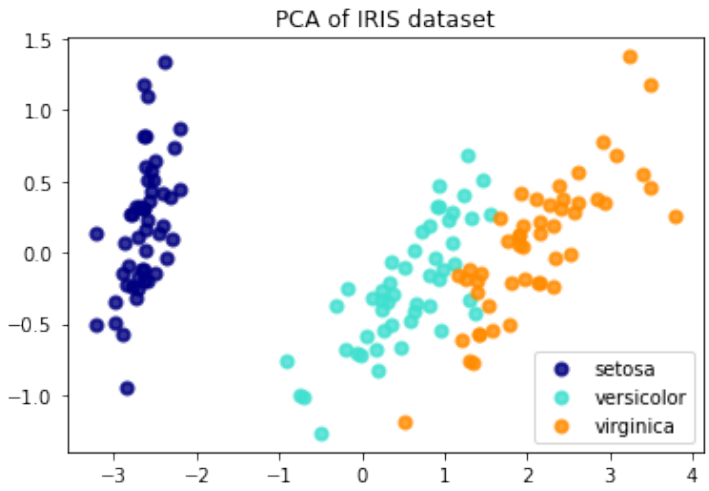
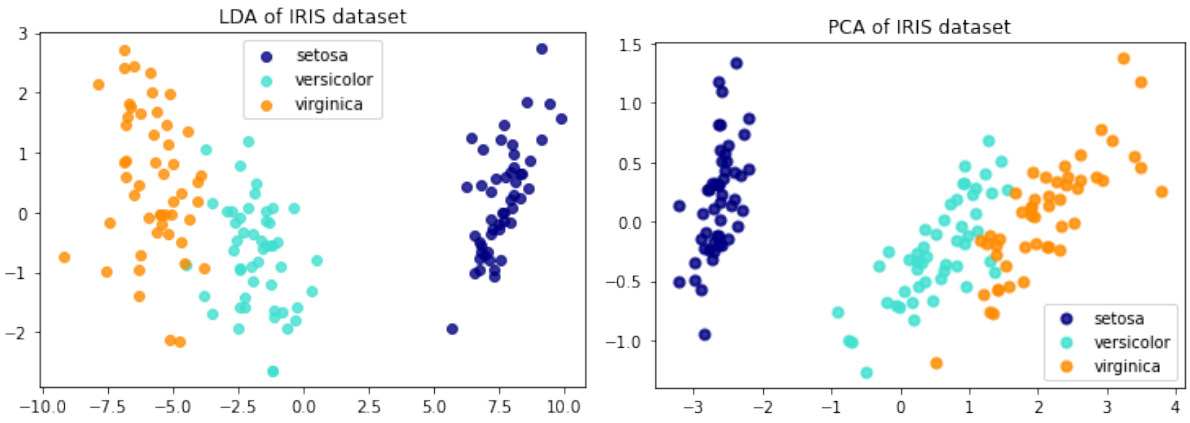


Fig. 3.12: PCA Projection of Iris Data Set

Figures 3.11 and 3.12 can be compared to each other to show how the 2D projections differ between LDA and PCA on the same data set. Figure 3.13 below shows that comparison and many similarities between the two can be observed.



(a) Two Component LDA

(b) Two Component PCA

Fig. 3.13: Comparison of LDA and PCA on Iris Data Set

Chapter 4

Developed Hybrid Methods and Python Code Process

This section will describe in detail how the whole developed method works, as well as the reasoning behind each step. The code was written in Python and is run using JupyterLab, a web-based development, editing and application environment for notebooks, code and data [15].

4.1 Data Handling

4.1.1 Reading in Data

The output data from the AUSRC is read into the JupyterLab workspace and is manipulated so that the features are represented by columns, and individual flyouts are represented by rows. It is important to exclude any features that could not be readily measured or are not of importance.

For example, the '12 Class Data Set' is modeled in two dimensions (X and Z), but still returns Y dimension data that has a mean of zero but involves some noise. This Y position and velocity is dropped before being inserted into the algorithm. The data set also includes TIME, a set of features representing time from launch, which is manipulated to show time from detection since time from launch should not be assumed to be available. This is done by setting the initial observation TIME to zero and subtracting the initial observation TIME from the rest of the observations.

4.1.2 Features

Now that the data is properly manipulated to represent an adverse missile scenario, the feature type can be selected. The features can be left as is, or 'derived' features can be calculated. Three scenarios will be shown in the Results chapter (Section 5):

1. Performance Variables: Performance parameters such as thrust, exit pressure, chamber pressure and burn time are the features. These parameters may not be readily available in a real life scenario.

2. Trajectory Data: Time from detection, position and velocity are the features. They are recorded at regular intervals over a select period in multiple dimensions. These are meant to emulate incoming radar data.

3. Derived Features: To attempt to emulate performance variables, 'derived' features are calculated from the trajectory data. This method was only applied to the '12 Class Data Set'. The 'derived' features include determining the average and maximum position, velocity and acceleration in both X and Z dimensions. Acceleration is calculated using the change in velocity across the known time between observations (0.4 seconds). A two-degree polynomial curve fit is applied to the trajectory, on three scenarios, $X\ vs\ Z$, $X\ vs\ TIME$, and $Z\ vs\ TIME$. The coefficients from these polynomials are used as the features. This method gives 21 total features.

4.1.3 Train-Test Split

The data set is now ready to split into a train, test, and validation group. This is done using the *scikit-learn* algorithm $train\_test\_split$ [16]. The algorithm splits the data into two groups according to the user's desired size, stratified by class label, so each class is equally represented in the each set. The first split is typically a 90% training to 10% test from the original data set. To make a validation set, the test set is split in half, again by class label. The result is 90% training, 5% test, and 5% validation sets, where the percentages are of the original data set size. The training set is used to train the OCSVM and LDA, the test set is used on the OCSVM, and

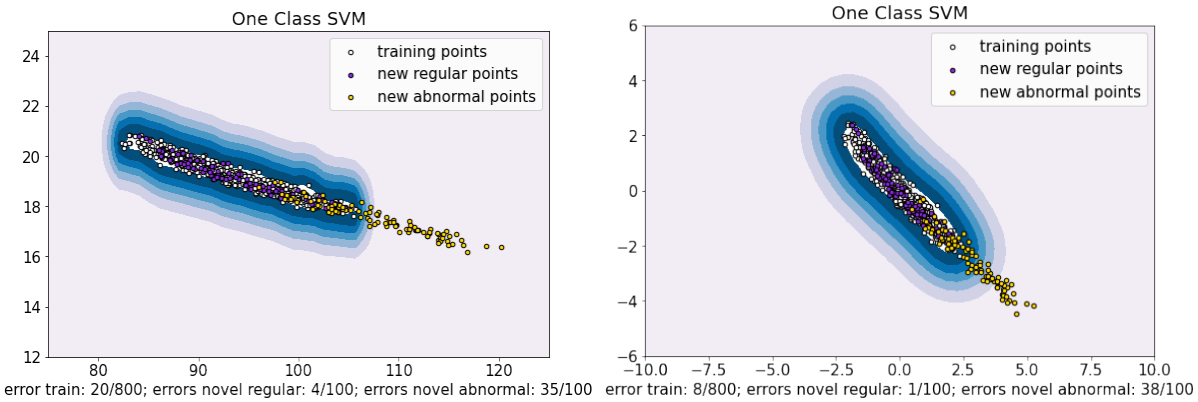
the validation set is used on the LDA. To create a novelty class, the novelty class labels is input into the code and that is dropped from the training data set.

4.1.4 Standardization

All three data sets are then standardized according to each feature across all classes. Standardization results in a data set with a mean of 0 and a standard deviation of 1. The mean (μ) and standard deviation (σ) of each feature is calculated, then each is transformed according to Eq. 4.1.

$$x_{standardized} = \frac{x_{original} - \mu}{\sigma} \tag{4.1}$$

Standardizing the data set eliminates the impact of units, leading to higher accuracy analysis. Figure 4.1 shows a sample OCSVM scenario where the white points represent the training points of the clean class, the purple points represent the test points of the clean class, and the yellow points represent test points from a novel class. Note the transformation of units and how the standardization reduces the training error by 60% and the test error by 75%. The novel error slightly (8%) increases in this scenario. Overall error is considerably reduced.



(a) Original Data

(b) Standardized Data

Fig. 4.1: OCSVM Comparison Standardized vs Original Data

4.1.5 Feature Reduction

The standardized data sets are then run through the PCA. The number of components set are determined on a case-by-case basis. In general, the minimum number of components needed to represent $> 95\%$ of the explained variance is used. Figure 4.2 below shows the first three principal components plotted against each other from the '12 Class Data Set' (Section 2.4). The first three principal components account for 91% of the explained variance. The explained variance percentage can be seen on the axis labels for each component.

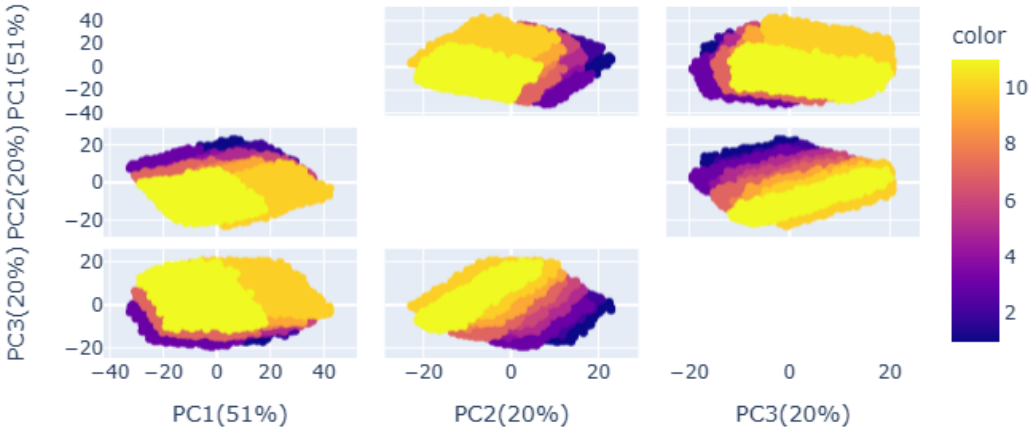


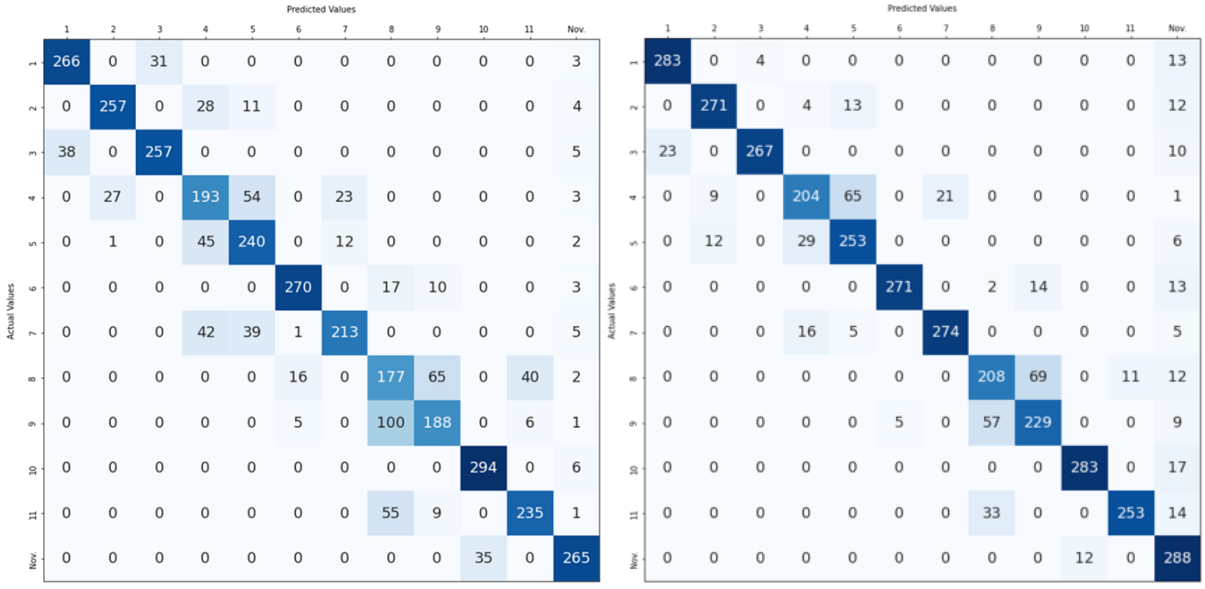
Fig. 4.2: PCA Projection of 12 Class Data Set

In this example, there were 304 original features, reduced to 50 principal components by the PCA, which explained 99.76% of the explained variance.

Confusion matrices are a easy way to visualize the accuracy of an algorithm. They show the algorithm's predicted classification labels across the horizontal axis and the actual classification labels across the vertical axis. These matrices will be used extensively throughout the Results chapter (Section 5).

Figure 4.3 shows a sample run of the OCSVM on the '12 Class Data Set'. Class '12' is the novelty class, labelled as 'Nov.'. Confusion matrix (a) is the OCSVM run on the original data, resulting in a 79.3% overall accuracy. The PCA was then run on the data set with no other

changes, and returned confusion matrix (b), with an 85.6% overall accuracy, or a 6.3% overall increase for this scenario.



(a) Before PCA (79.3% Accuracy)

(b) After PCA (85.6% Accuracy)

Fig. 4.3: Confusion Matrix Comparison of PCA Utilization

The PCA not only increased accuracy but also shifted where the error occurs. Before the PCA, there was more inter-class misclassification, which is defined as a clean class point that was incorrectly classified as a different clean class. After the PCA, more false positives occurred, which are defined as a clean class point classified as a novelty point. False positives are much easier to deal with in post-run analysis and are as equally important as novelties to extract and identify, because they represent outliers for their respective clean class. For the current scenario, they represent a missile flyout that is pushing the boundaries of what is commonly known about the class, indicating modifications to the missile class.

4.2 One-Class Support Vector Machine

Once all data handling and preprocessing is complete, the OCSVM is fit to the data. Since the OCSVM is a one-vs-all algorithm, a multi-class classifier with novelty detection is built by iteratively training and running one clean class at a time. Not only is this more accurate than lumping all the clean classes into one single system [11], but gives the ability to classify

clean points and classify false positives. Since the iterative OCSVM creates boundaries around each clean class, it only defines the feature space it learns through training, unlike some other traditional machine learning algorithms such as LDA where the entire feature space is defined. Figure 4.4 shows a traditional feature space, note how both novelty classes would be incorrectly classified.

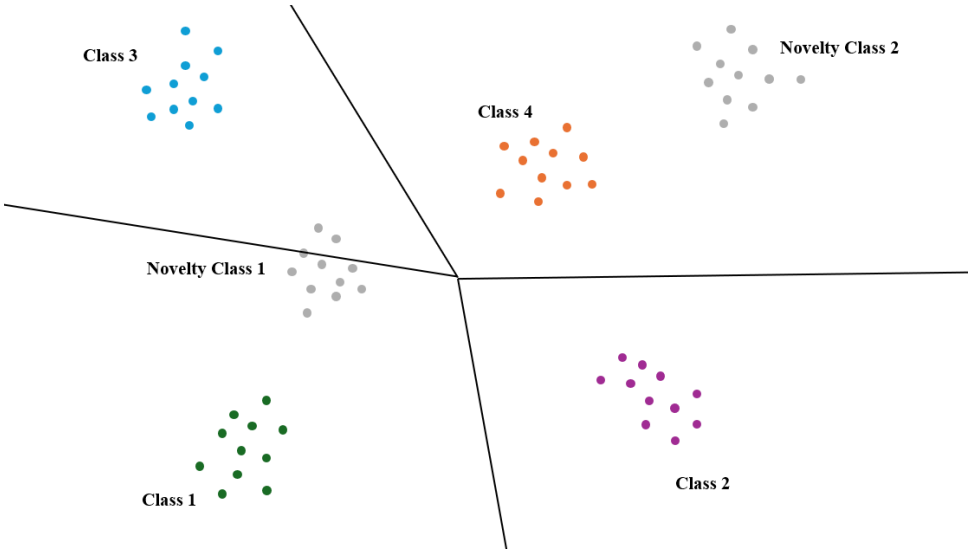


Fig. 4.4: Traditional Feature Space

Figure 4.5 shows how an iterative OCSVM feature space looks. Each OCSVM iteration defines the feature space for the clean class that it was trained on and nothing else. In this scenario, both novelty classes are extracted and not forced into a clean class. Figs. 4.4 and 4.5 were modeled after Ref. [12].

The iterative OCSVM does not store the hyperplanes for all classes though. It should be elaborated that each class is run one at a time, the OCSVM is fit to the class, the hyperplane is defined and each flyout from the test set is judged and scored. All data but the classification and sample scores are discarded before proceeding to the next class. This process is further explained below in this section.

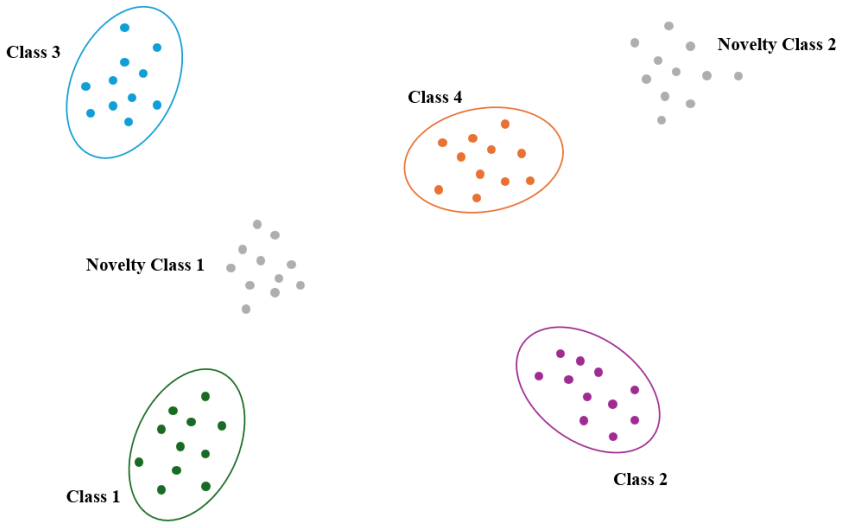


Fig. 4.5: OCSVM Feature Space

OCSVM hyperparameters are tuned according to the data set and scenario being run, taking into account the number of flyouts and the number of features being used. Proper tuning for high classification accuracy without overfitting the data results in the most accurate novelty detection. Reference [10] attempted to build a multi-class classifier with novelty detection using SVMs but struggled to produce accurate results. One reason for this was that their variable 'reject' was a set percentage of points that were to be classified as novelty points. For instance, if the 'reject' is 10%, in every run only 90% of the points are kept. When tuning the OCSVM hyperparameters, an error rate on the training data is set (Section 3.1.2), but nothing is defined as to how many points to classify as novelties. This allows the algorithm to be more objective in its classification process, not forcing any points out.

Throughout the iterative OCSVM, the score samples and class prediction are kept in a matrix, constantly updating if better predictions are found. As described in Section 3.1.2, a score is given to each missile flyout, receiving a higher score the more it resembles the clean class the OCSVM was trained on. The OCSVM will classify some instances as inliers, belonging to the trained-on class, and the rest as outliers, or not belonging to the trained-on class. Since multiple OCSVM iterations can claim points as an inlier and the inlier score range between OCSVM iterations are different, a relative score similar to a percentile is used. This relative

score is defined in Eq. 4.2. The missile flyout is classified as the class that it scored the highest relative score for.

$$Relative\ Score = \frac{Score\ Sample - Minimum\ Inlier\ Score}{Maximum\ Inlier\ Score - Minimum\ Inlier\ Score} \tag{4.2}$$

Table 3.1 shows an example of a missile flyout. In this example, Class '1', '2', and '3' all claimed the flyout as an inlier. The flyout scored highest for Class '2' with a 2.38, but is in the bottom $\sim 17\%$ (Relative Score: 0.172) of flyouts claimed as inliers. Compare this to Class '1', where the flyout only scored a 1.47, but is in the top $\sim 30\%$ (Relative Score: 0.698) of flyouts claimed. This flyout then looks much more like Class '1' than Class '2' and would therefore be classified as Class '1'.

Table 4.1: Sample Score Example for Single Flyout

	Class 1 OCSVM	Class 2 OCSVM	Class 3 OCSVM
Score	1.47	2.38	0.47
Max Inlier Score	1.69	4.54	3.02
Min Inlier Score	0.96	1.93	0.40
Relative Score	0.698	0.172	0.02

This process repeats itself and continuously updates predictions as each OCSVM iteration is ran. If a point is never claimed by any of the known classes, it is deemed a novelty point. The novelty points still receive sample scores from each class's OCSVM, and the highest raw score the point receives is stored in the same matrix although it was never an inlier.

4.3 Novelty Class Extraction

After the iterative OCSVM has ran, a matrix is returned of its class predictions and the maximum relative score each claimed flyout received along with the maximum raw score each outlier received. The flyouts that the OCSVM classified as novelties may contain clean class points that are outliers for their respective classes, called false positives. An objective threshold, defined by Eq. 4.3, equal to 95% of the minimum inlier score for the class that the missile flyout scored highest for, is used to divide these flyouts.

$$Objective\ Threshold = 0.95(Class\ Minimum\ Inlier\ Score) \tag{4.3}$$

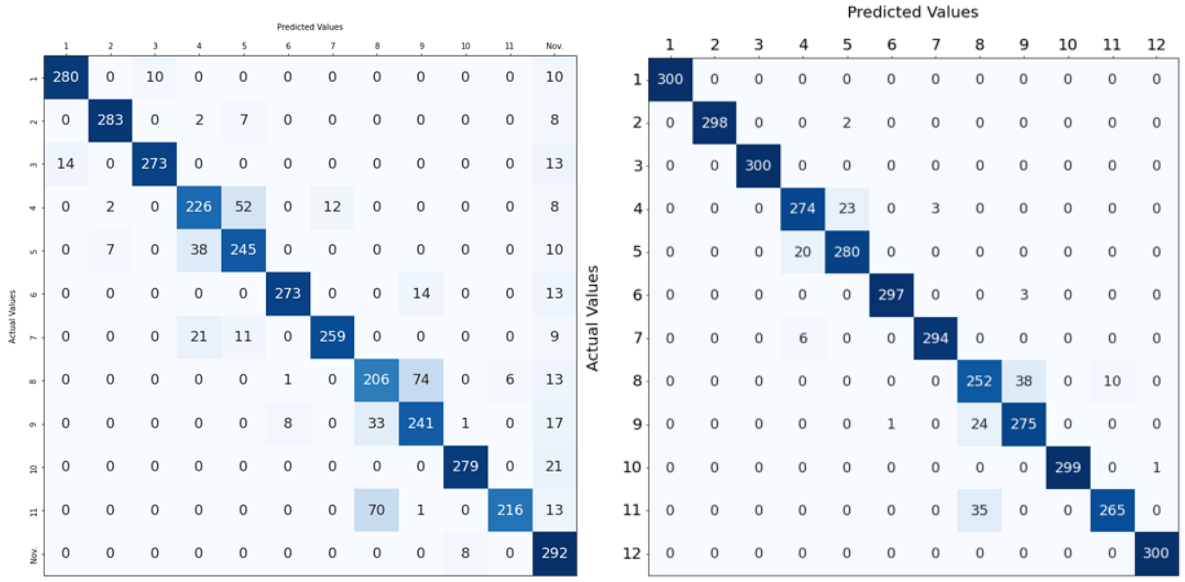
This objective threshold is used to distinguish the clean class outliers from the true novelty points. In general, if the missile flyout scores higher than this threshold, that means it looks very similar to but scored just outside of what was considered an inlier. These points then can be classified and are important to detect and study as well, because they can represent a modification or performance increase for a known missile class. If a missile flyout scores below this threshold, it is considered a 'true' novelty, and generally looks nothing like what is currently known. The points that score below the threshold are grouped together, considered 'true' novelties, and are labeled as a new class.

4.4 Linear Discriminant Analysis

Once enough 'true' novelty points are found and grouped into a class, they can be added to the training data. In this way, the LDA works as a novelty detector by giving the 'true' novelty points a class label, allowing the LDA to fit a hyperplane to the feature space for the novelty class. It is shown by the 'Ballistic/Maneuverable Data Set' Results (Section 5.2.2) how few points are required for the LDA to accurately characterize a novelty class.

The LDA then takes the clean class training data and the 'true' novelty class extracted by the iterative OCSVM and is fit to them. The LDA is implemented in one of two ways.

The first is to use the LDA to clean up the analysis on the incoming test data done by the iterative OCSVM. Since the LDA provides higher accuracy classification than the iterative OCSVM, the LDA can classify the false positives back to their clean classes, decrease the number of inter-class misclassifications and extract any more novelty points that the OCSVM missed. Figure 4.6 shows sample scenario results for this cleanup implementation method.



(a) Iterative OCSVM Test Results (85.3% Accuracy) (b) LDA Cleanup Test Results (95.3% Accuracy)

Fig. 4.6: Confusion Matrix Comparison Before/After LDA Cleanup

In this sample run, the iterative OCSVM returns an 85.3% $(3073/3600)$ overall accuracy, with a 97.3% $(292/300)$ novelty extraction rate, 392 inter-class misclassifications, and 135 false positives. After filtering the entire predicted novelty class (rightmost column) through the threshold and adding the 'true' novelty class to the training data, the same data set was run through the LDA. The LDA cleaned up the analysis and returned a 95.3% $(3434/3600)$ overall accuracy, with a 100% $(300/300)$ novelty extraction rate, 165 inter-class misclassifications and 1 false positive. The LDA not only greatly increased accuracy $(+10\%)$, but by cross referencing the results from both algorithms, the points that were false positives in the OCSVM results were reclassified to their clean classes and can now also be further analyzed.

The second method is to apply the LDA to the validation data set. The validation was set aside to be used as a stress test to determine how accurate the LDA would be on new incoming data and to observe where and when it becomes inaccurate or breaks down. Most of the results are done using this method across a variety of scenarios.

Chapter 5

Results

5.1 OCSVM Iterative Method Alone

This section shows the results of using the iterative OCSVM by itself, in multiple scenarios without the aid of the LDA or a PCA. The iterative OCSVM shows promise for future work by itself and paired with other machine learning algorithms.

5.1.1 Single Novelty Class

This scenario shows the ability of the iterative OCSVM to accurately classify known classes and extract a single novelty class.

The '18 Class Data Set' (Section 2.3) is used for this analysis. The run uses 4 performance parameters as features: BURNTIME (Time of burn), MAXTHRUST (Maximum thrust), MAXPC (Maximum chamber pressure), and MAXPE (Maximum exit pressure). It should not be assumed that these performance features can be observed in real scenarios. This data set was used as a proof-of-concept before adding to the developed algorithm. These features are fed directly to the iterative OCSVM, a PCA is not run for this scenario. The OCSVM hyperparameters are: $nu = 0.01$, $gamma = 0.5$, and $tol = 1e^{-3}$. Class '18' was used as the novelty class, the training data set had 2000 runs per class and the testing data set had 250 runs per class. Figure 5.1 shows the results of this run.

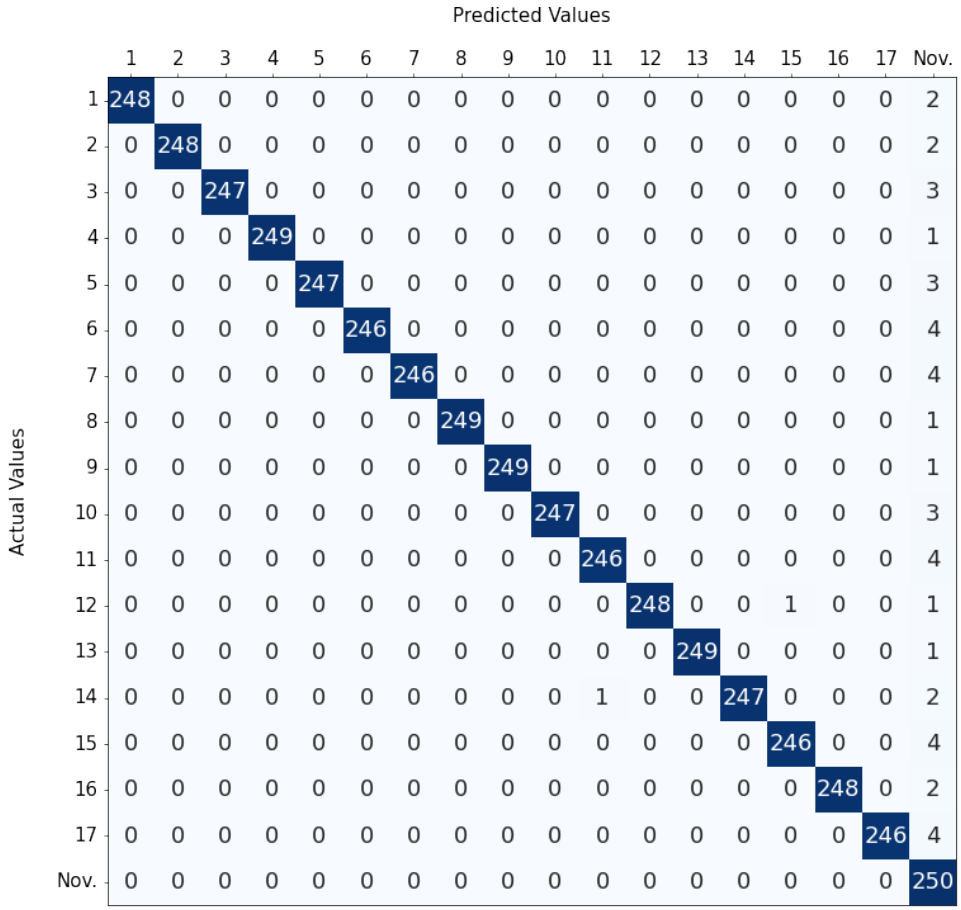
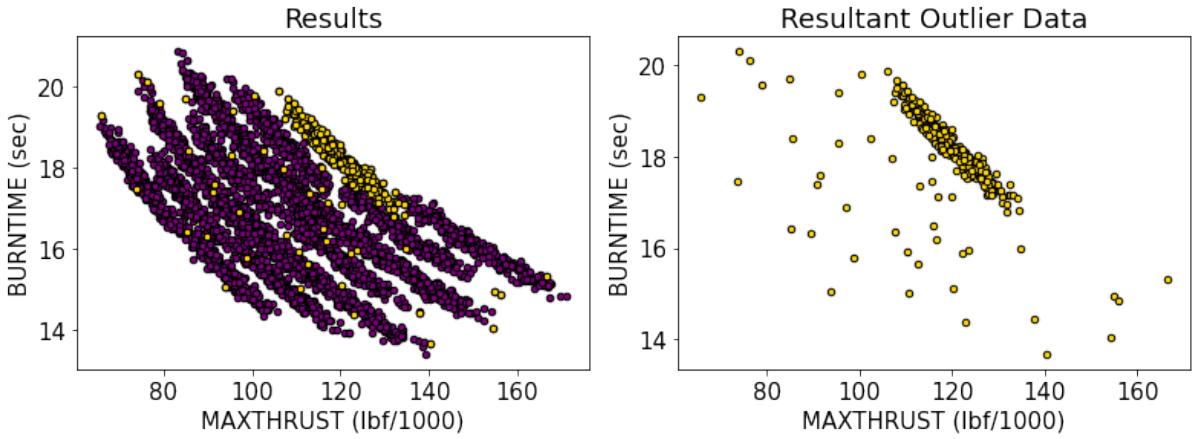


Fig. 5.1: Single Novelty Class Iterative OCSVM Confusion Matrix

This confusion matrix represents an overall 99% accuracy $(4456/4500)$ with 2 inter-class misclassifications and 42 false positives. Figure 5.2 shows a BURNTIME vs MAXTHRUST scatter plot with the points deemed novelties in yellow. Subplot (a) shows all points plotted to give a visual of how the classes overlap in this feature space. Class '18' is the dense cluster of yellow points, selected as the novelty class since it is pushing the boundaries of what is currently known for missile performance. Subplot (b) shows how the outlier data looks alone. When using many test points, the novelty class is clearly observable and can be extracted using a clustering machine learning algorithm or by using the objective threshold described in Section 4.3. After the threshold is applied and the novelty points are separated from the outliers, the raw sample scores can be used to classify the outlier points back to their respective clean classes.



(a) Iterative OCSVM Results

(b) Outlier Data

Fig. 5.2: BURNTIME vs MAXTHRUST

5.1.2 Multiple Novelty Classes

This scenario shows the ability of the iterative OCSVM to accurately classify known classes while extracting multiple novelty classes. This is useful in cases where there may be many forms of adversaries, in this case multiple missile classes. This analysis proves that the OCSVM is not affected by adversarial data and is only trained on clean class data.

The '18 Class Data Set' (Section 2.3) is used in this analysis. The run uses 4 performance parameters as features: BURNTIME (Time of burn), MAXTHRUST (Maximum thrust), MAXPC (Maximum chamber pressure), and MAXPE (Maximum exit pressure). These features are fed directly to the iterative OCSVM, a PCA is not run for this scenario. The OCSVM hyperparameters are: $nu = 0.01$, $gamma = 0.5$, and $tol = 1e^{-3}$. Class '17' and Class '18' were used as novelty classes, the training data set had 2000 runs per class and the testing data set had 250 runs per class. Figure 5.3 shows the results of this run.

The confusion matrix represents an overall 98.6% accuracy $(4440/4500)$ with 0 inter-class misclassifications and 60 false positives. As a reminder, the novelty class is made up of two classes, totaling 500 points. Figure 5.4 shows that Class '17' is from the middle of the class distribution, as opposed to the edge like Class '18', when viewing the BURNTIME vs MAXTHRUST feature space. This highlights the ability of the iterative OCSVM to provide high accuracy novelty extraction despite the class location.

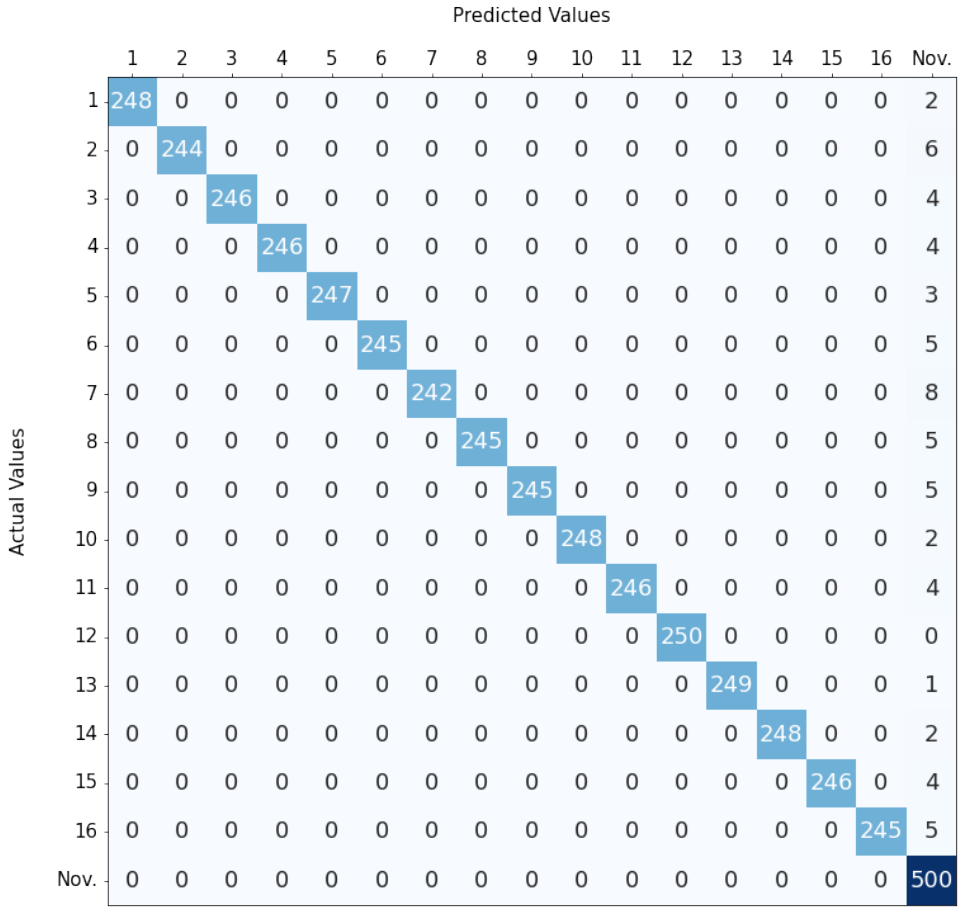
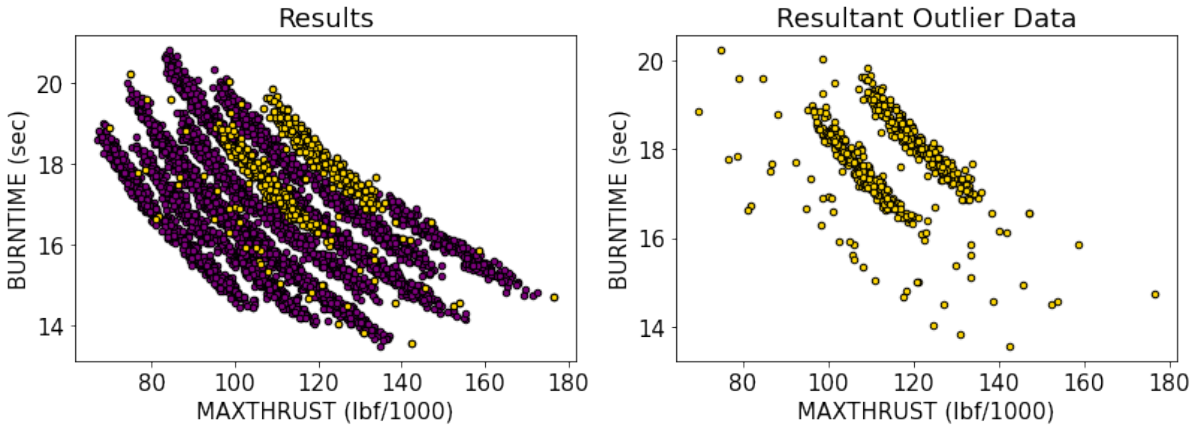


Fig. 5.3: Multiple Novelty Classes Iterative OCSVM Confusion Matrix

Figure 5.4 shows the resulting scatter plot from this scenario. When many test points, once again clear classes are observed. Pairing a clustering machine learning algorithm to these results to define and extract multiple novelty classes should be pursued in future work.



(a) Iterative OCSVM Results

(b) Outlier Data

Fig. 5.4: BURNTIME vs MAXTHRUST

5.2 OCSVM - LDA

This section shows the entire developed OCSVM-LDA hybrid method. It is applied to two different data sets, the '12 Class Data Set' (Section 2.4) and the 'Ballistic/Maneuverable Data Set' (Section 2.5).

5.2.1 12 Class Data Set

Three tests were done on the '12 Class Data Set'. It was first utilized to test the accuracy when using trajectory data and derived features (Section 4.1.2). As will be shown later in this section, the OCSVM-LDA is accurate on both data types, with each type having advantages and disadvantages. A time study was conducted next, where the OCSVM-LDA had only portions of the full trajectory. This was done to emulate the event that the missile was detected late or that the time frame is limited in which the algorithm must accurately classify the incoming missile. Lastly a missing data study was conducted, where time steps throughout the trajectory are dropped at random to emulate spotty radar observances. These tests and scenarios are explained further in detail later throughout this section.

Trajectory Data versus Derived Features

After the OCSVM-LDA concept was proven on trajectory data, it was then tested on derived features to determine which data type was more accurate and robust. The output data, or trajectory data, from the '12 Class Data Set' consists of position and velocity data from the missile in the X and Z coordinate directions. An observation is made every 0.4 seconds for 30 seconds. The derived features are calculated from the trajectory data, used to attempt to emulate performance parameters for each missile.

A. Trajectory Data

For the trajectory data the full 30 seconds of observations are used, resulting in 300 initial features. There are 75 features each from X position ('xs'), Z position ('zs'), X velocity ('vxs') and Z velocity ('vzs'). The data is split three ways into a training set with 2700 flyouts per class, a test set with 150 flyouts per class, and a validation set with 150 flyouts per class. Class '12' is

set as the novelty class. A PCA is run on the data, reducing the number of dimensions from 300 features down to 48 principal components. The 48 principal components account for 99.76% of the explained variance for the data set. The iterative OCSVM is ran on the data, the results of which are shown in the confusion matrix below, Fig. 5.5. The OCSVM's hyperparameters were set at: $nu = 0.01$, $gamma = 0.009$, and $tol = 1e^{-3}$. The confusion matrix represents an overall 86.5% (1557/1800) accuracy with 52 false positives, 5 false negatives and 186 inter-class misclassifications.

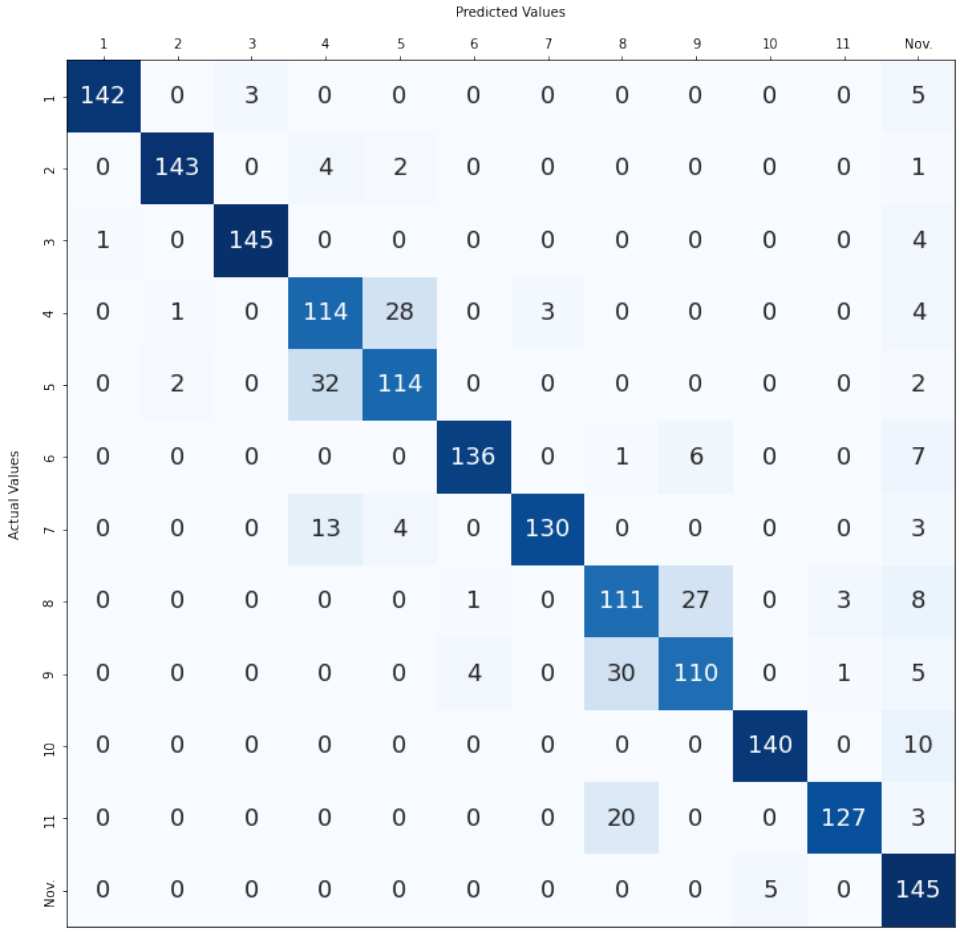


Fig. 5.5: Trajectory Data Iterative OCSVM Confusion Matrix

The OCSVM had a 96.7% (145/150) novelty detection rate and 197 points classified as novelties. Out of the 197 points run through the objective threshold, 148 points scored below the threshold and were deemed 'true' novelties. Take note that there are more points deemed 'true' novelties than were extracted from the novelty class (145). The threshold is not perfect and the newly built 'true' novelty class will contain clean class points. The LDA proves to be

robust and is able to accurately classify the novelty class in light of these adverse points within the 'true' novelty class.

The 'true' novelty class is added to the training data and the LDA is fit to the now 12 known classes. The validation data set is fed into the LDA. Figure 5.6 shows the confusion matrix from the LDA analysis on the validation data set. It shows an overall 96.0% $(1728/1800)$ accuracy with 0 false positives and 72 inter-class misclassifications. That represents an increase in overall classification of 9.5% from the OCSVM analysis.

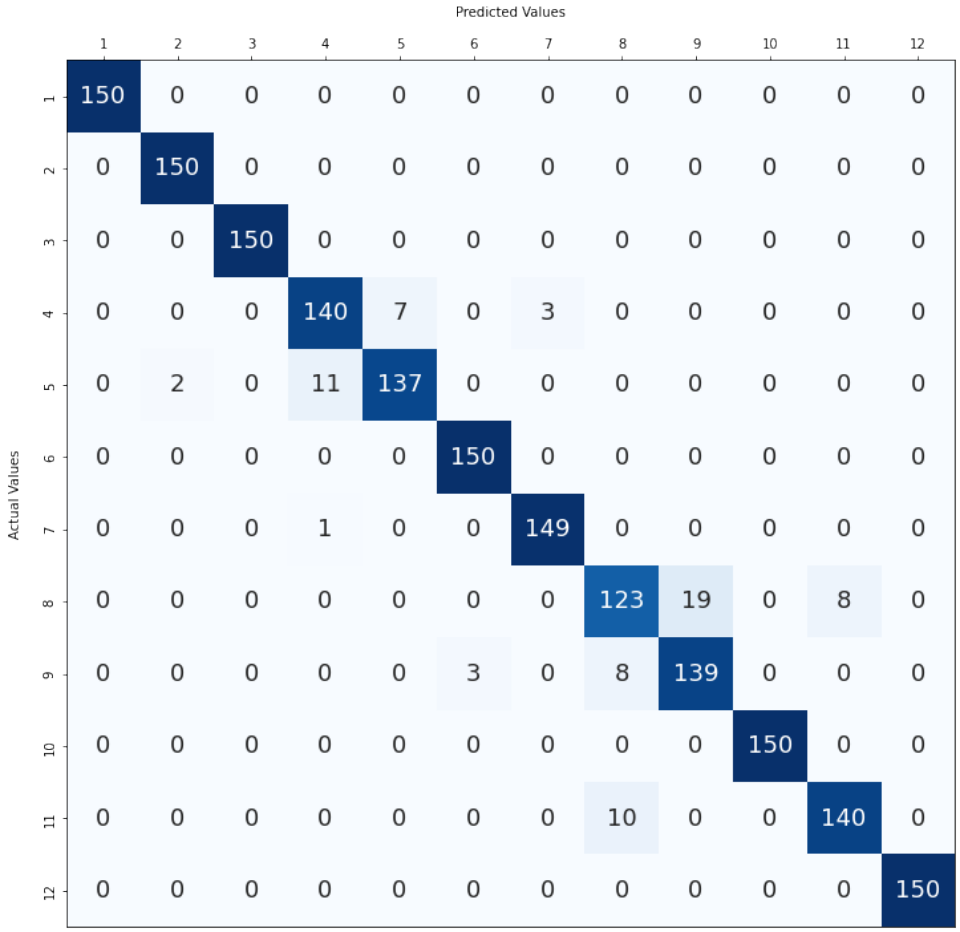


Fig. 5.6: Trajectory Data LDA Confusion Matrix

B. Derived Features

The derived features consist of 21 total features calculated from the full 30 seconds of trajectory data. These include 6 features from the maximum position, velocity and acceleration in both coordinate directions (Max X,Z,VX,VZ,AX,AZ). Another 6 features come from the average position, velocity, and acceleration in both coordinate directions (Avg. X,Z,VX,VZ,AX,AZ).

Lastly, 9 features come from the coefficients of two degree polynomial curve fits on: $X\ vs\ Z$, $X\ vs\ TIME$, and $Z\ vs\ TIME$. TIME is time from detection. Position and velocity data are given from the AUSRC, but acceleration is calculated using the known observation time step.

The data is split three ways into a training set with 2700 flyouts per class, a test set with 150 flyouts per class, and a validation set with 150 flyouts per class. Class '12' is set as the novelty class. A PCA is ran on the data, reducing the number of dimensions from 21 features down to 15 principal components. The 15 principal components account for $> 99\%$ of the explained variance for the data set. The iterative OCSVM is ran on the data, the results of which are shown in the confusion matrix below, Fig. 5.7. The OCSVM's hyperparameters were set at: $nu = 0.01$, $gamma = 0.175$, $tol = 1e^{-3}$. The confusion matrix represents an overall 86.1% $(1551/1800)$ accuracy with 65 false positives, 3 false negatives and 181 inter-class misclassifications.

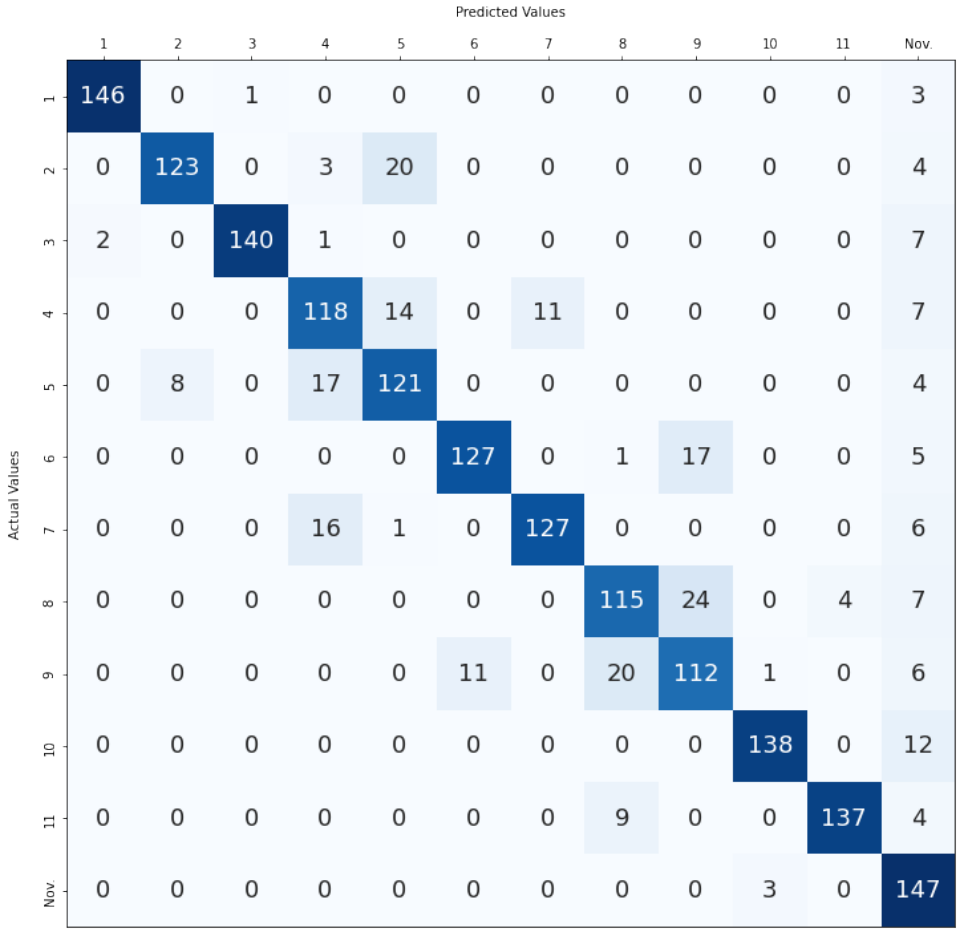


Fig. 5.7: Derived Features OCSVM Confusion Matrix

The OCSVM had a 98.0% $(147/150)$ novelty detection rate and 212 points total classified as novelties. Out of the 212 points run through the objective threshold, only 181 scored below the threshold and were deemed 'true' novelties. The 'true' novelty class is added to the training data and the LDA is fit to the now 12 known classes. Figure 5.8 shows the confusion matrix from the LDA analysis on the validation data set. It shows an overall 95.5% $(1719/1800)$ accuracy with 0 false positives and 81 inter-class misclassifications. This represents an increase in overall accuracy of 9.4% from the OCSVM analysis.

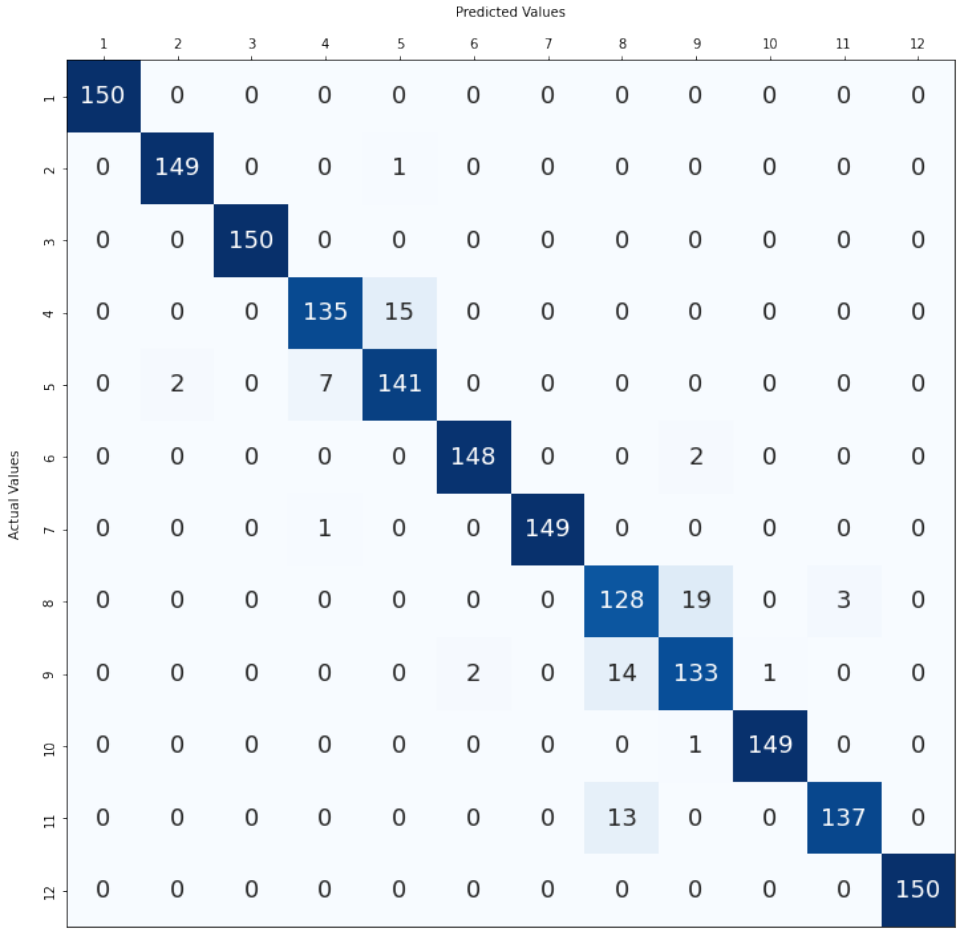


Fig. 5.8: Derived Features LDA Confusion Matrix

Time Study

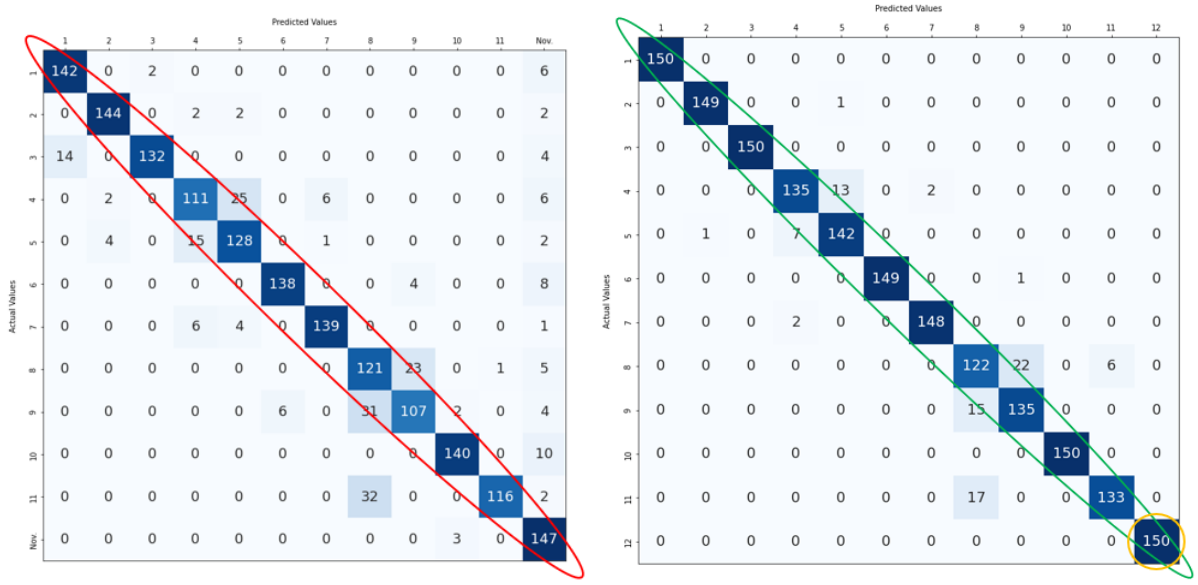
After demonstrating that the OCSVM-LDA hybrid method worked accurately on the full 30 seconds from the '12 Class Data Set', a time study was done. Time was cut from the beginning and end of the 30 second time frame to emulate late detection of the missile or limited time

frames in which the algorithm must accurately classify the incoming missile. This also tests the robustness of the algorithm as well as determines the portion of the missile trajectory most critical to classification and novelty extraction. For each time frame the observations are steady, remaining at the original time step of 0.4 seconds for the duration of the selected frame.

Traditionally these problems are done by training the algorithm on the full data set and then dropping data from the test set until the results become inaccurate. In this method, the incoming data is observed and then the same data is pulled from the training data and used to train the algorithm. For example, if the missile data that is observed is from 5 through 10 seconds after initial detection, missile data from 5 through 10 seconds would be used to train the algorithm and classify the missile.

The results are shown in table form, as there are too many runs to concisely show all of the confusion matrices. In the table, 'Total Time' shows the total time observed and 'T.F.D. Range' shows the time from detection range. The accuracies shown are the circled numbers in Fig. 5.9 divided by the total number of instances for a perfect run, given the test size. The 'SVM Total Acc.' is SVM overall accuracy shown by Fig. 5.9 (a) circled in red, divided by 1800, 'LDA Nov. Acc.' is LDA novelty class accuracy shown in Fig. 5.9 (b) circled in yellow, divided by 150, and 'LDA Total Acc.' is overall accuracy shown in Fig. 5.9 (b) circled in green, divided by 1800.

For example Fig. 5.9 (a) gives a 'SVM Total Acc.' of 86.9% $(1565/1800)$, Fig. 5.9 (b) shows a 'LDA Nov. Acc.' of 100% $(150/150)$ and a 'LDA Total Acc.' of 95.1% $(1713/1800)$.



(a) Sample OCSVM Results

(b) Sample LDA Results

Fig. 5.9: Accuracy Definitions

This study was run on 6 different time frames, with the full 30 second trajectory shown at the top of each table. The results for both the trajectory data and the derived features are shown. The OCSVM hyperparameter 'gamma' is shown next to the 'OCSVM Total Acc.'. For the trajectory data, since the number of initial features fluctuates with the time frame, the number of principal components are set to 16% of the total number of features. This percentage consistently allows the PCA to characterize the data through explained variance on the '12 Class Data Set'. The derived features number of principal components stays constant at 15.

Table 5.1: Time Study OCSVM Total Accuracy Results

Time Scale		OCSVM Total Acc. (gamma)	
Total Time	T.F.D Range	Trajectory Data	Derived Features
30 sec	0-30 sec	86.6% (0.009)	87.0% (0.175)
22 sec	0-22 sec	88.0% (0.009)	86.7% (0.175)
14 sec	0-14 sec	78.8% (0.009)	90.1% (0.175)
10 sec	0-10 sec	76.1% (0.04)	79.4% (0.175)
10 sec	4-14 sec	88.7% (0.04)	83.9% (0.175)
8 sec	6-14 sec	83.0% (0.04)	81.4% (0.175)

Table 5.2: Time Study LDA Novelty Accuracy Results

Time Scale		LDA Nov. Acc.	
Total Time	T.F.D Range	Trajectory Data	Derived Features
30 sec	0-30 sec	100.0%	100.0%
22 sec	0-22 sec	100.0%	100.0%
14 sec	0-14 sec	98.7%	98.7%
10 sec	0-10 sec	4.7%	13.3%
10 sec	4-14 sec	98.7%	97.3%
8 sec	6-14 sec	86.0%	81.3%

Table 5.3: Time Study LDA Total Accuracy Results

Time Scale		LDA Total Acc.	
Total Time	T.F.D Range	Trajectory Data	Derived Features
30 sec	0-30 sec	95.0%	96.0%
22 sec	0-22 sec	95.7%	95.9%
14 sec	0-14 sec	95.1%	95.8%
10 sec	0-10 sec	66.4%	69.4%
10 sec	4-14 sec	93.5%	92.2%
8 sec	6-14 sec	80.2%	83.0%

Table 5.3 shows how the overall accuracy changes across various time frames. It shows that using the 'T.F.D Range' 0-10 sec and 6-14 sec, the algorithm breaks down and becomes inaccurate. This can be backtracked to the iterative OCSVM providing the LDA with bad 'true' novelty information. As shown in Table 5.1 the OCSVM dips to its lowest overall accuracy at this time frame, where most of the decline in accuracy comes from losses in extracting the novelty class as opposed to inter-class errors. In terms of overall accuracy, 'T.F.D Range' of 4-14 seconds is the most critical section of the flyout for these missile classes. In the 4-14 second 'T.F.D Range', the algorithm provides $> 92\%$ accuracy and $< 4\%$ off the peak accuracy across both data types. As stated before, when accounting for the small fluctuations in accuracy ($< 2\%$) between runs due to random sampling when splitting, the differences in accuracies between the trajectory data and derived features are negligible. The largest difference can be seen in Table 5.1 'T.F.D Range' 0-14 seconds, where the OCSVM Total Acc. differs by 11.3%. This contrast does not show up in Table 5.3 though, where the LDA Total Acc. only differs by 0.7%.

Missing Data Study

The missing data study was the final test conducted on the '12 Class Data Set'. This study was performed by dropping time steps throughout the trajectory at random, meant to emulate spotty radar observations of an incoming missile trajectory. A variable called 'fraction' is set, representing the fraction of the total time steps removed from the set. For example, when 'fraction' is set to 20%, only 80% of the data is kept.

The training method worked well in the time study, so the same method was employed here. All of the time steps that are missing from the incoming data are also removed from the training data. This allows the algorithm to compare the two flyouts according to the same points along their individual trajectories.

The results are in table form below, as there are too many runs to show all of them concisely in confusion matrices. The accuracies shown are the same as used in the time study, and are explained by Fig. 5.9.

The missing data study was done across 10 different 'fraction' values, starting at 10% and working all the way up to 95%. The number of principal components for the trajectory data varied according to the fraction dropped from the data set, and again was set equal to 16% of the total original features. The number of principal components kept for the derived features stays constant at 15.

Table 5.4: Missing Data Study OCSVM Total Accuracy Results

Missing Data Fraction	OCSVM Total Acc. (gamma)	
	Trajectory Data	Derived Features
10%	86.2% (0.009)	82.4% (0.175)
20%	84.4% (0.009)	79.2% (0.175)
30%	83.6% (0.009)	81.7% (0.175)
40%	82.9% (0.009)	83.9% (0.175)
50%	89.1% (0.02)	81.9% (0.175)
60%	86.5% (0.03)	81.6% (0.175)
70%	84.5% (0.03)	83.3% (0.175)
80%	87.9% (0.04)	83.3% (0.175)
90%	89.3% (0.15)	86.1% (0.175)
95%	88.7% (0.25)	88.2% (0.175)

Table 5.5: Missing Data Study LDA Novelty Accuracy Results

Missing Data Fraction	LDA Nov. Acc.	
	Trajectory Data	Derived Features
10%	100.0%	100.0%
20%	99.3%	96.0%
30%	100.0%	94.7%
40%	99.3%	100.0%
50%	100.0%	97.3%
60%	100.0%	98.0%
70%	100.0%	98.7%
80%	100.0%	95.3%
90%	99.3%	98.7%
95%	100.0%	99.3%

Table 5.6: Missing Data Study LDA Total Accuracy Results

Missing Data Fraction	LDA Total Acc.	
	Trajectory Data	Derived Features
10%	94.9%	94.7%
20%	94.9%	95.1%
30%	95.9%	95.8%
40%	95.7%	95.5%
50%	95.4%	95.2%
60%	94.8%	95.1%
70%	93.9%	93.8%
80%	94.0%	94.5%
90%	90.3%	94.2%
95%	93.6%	94.2%

The results of the missing data study are unexpected and differ from the time study. All three accuracies recorded fluctuate negligibly across all 'fraction' value. Even when only using 5% of the data the 'LDA Total Acc.' is $> 90\%$. The random sampling from the train/test/validation splits and dropped time steps causes a higher rate of accuracy fluctuation when using little data. A run has the possibility of the incoming data to be points early on or very late in the trajectory, making the run very inaccurate. If the random points are spread sufficiently far enough apart to represent the entire trajectory, the analysis is accurate. Again, between the trajectory data and the derived features, the difference in accuracy is negligible.

5.2.2 Ballistic/Maneuverable Data Set

The 'Ballistic/Maneuverable Data Set' (Section 2.5) was the last data set applied to the OCSVM-LDA hybrid method. This data set is comprised of two classes, one ballistic and one maneuverable. The goal of this data set is to determine how quickly the algorithm can identify if an adverse missile is maneuverable. For the run, 4 seconds of trajectory data were used across the time frame 0.8-4.8 seconds after detection. Again, the trajectory data is recorded every 0.4 seconds. The data was split three ways into a training set with 800 flyouts per class, a test set with 100 flyouts per class and a validation set with 100 flyouts per class. The maneuverable class is set as the novelty class. The PCA transforms the 66 total features down to 30 principal components that account for $> 99.9\%$ of the explained variance in the data set.

Figure 5.10 shows the OCSVM confusion matrix. The OCSVM hyperparameters were set at: $nu = 0.01$, $gamma = 0.009$, and $tol = 1e^{-3}$. The confusion matrix shows an overall 51.5% accuracy with 25 false positives and 72 false negatives. Out of the 53 points deemed novelties, 33 scored under the objective threshold and were deemed 'true' novelties.

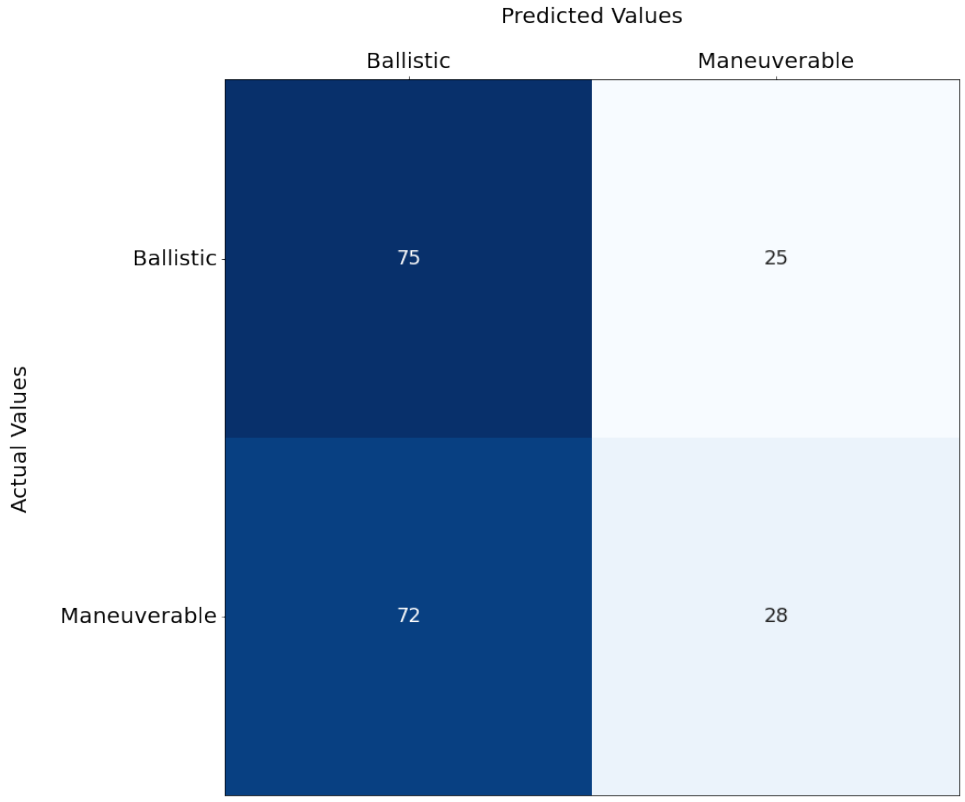


Fig. 5.10: OCSVM Confusion Matrix

Even with a low accuracy OCSVM analysis, the LDA is robust enough to accurately characterize the maneuverable group on few data points. Figure 5.11 shows the LDA resulting confusion matrix with an overall 98.5% accuracy with 1 false positive and 2 false negatives.

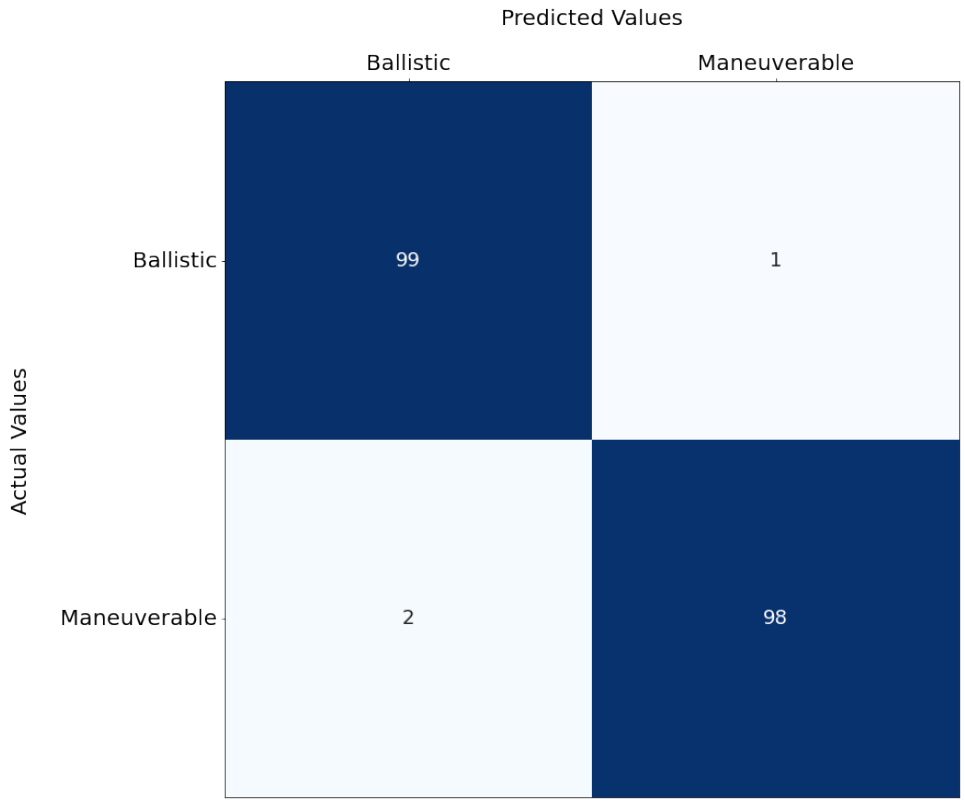


Fig. 5.11: LDA Confusion Matrix

Chapter 6

Conclusions and Future Work

The OCSVM-LDA hybrid method worked accurately and was robust on AUSRC missile data, providing high accuracy multi-class classification and novelty detection. The '18 Class Data Set' provided a relatively simple data set for proof-of-concept. The OCSVM alone is able to extract multiple novelty classes, for which a clustering machine learning algorithm should be added to identify groupings and extract the classes.

Results from the trajectory data and derived features on the '12 Class Data Set' showed negligible overall accuracy differences. The difference seen is in running each data type through the algorithm. The disadvantage of derived features is that they add a significant computational load to the process, inflating the run time to 133 seconds when running the full 30 second trajectory, while the trajectory data run on the same scenario only took 18 seconds. The disadvantage to trajectory data is that through the time study and missing data study the number of total features changed, necessitating hyperparameter tuning and principal component number changes between runs. The derived features stay constant at 21, allowing the hyperparameters and principal component number to stay constant as well. An automatic hyperparameter tuner should be added to eliminate the manual user tuning. Overall, the baseline accuracy on the '12 Class Data Set' for the LDA when it is fully supervised, or allowed to train on all 12 clean classes, is between $94 - 96\%$. The OCSVM-LDA is able to reach this baseline accuracy when only training on 11 clean classes, even reaching this accuracy at points in the time study and missing data study when the full trajectory was not available.

Results from the 'Ballistic/Maneuverable Data Set' showed how the iterative OCSVM can return inaccurate results but the LDA is robust enough to still accurately characterize and classify the novelty class on few points.

Through various data sets, tests and scenarios, the OCSVM-LDA hybrid method proved to be a robust and accurate multi-class classification and novelty detection method. It provides an in depth analysis of the incoming data points, not only classifying points and detecting true novelty points, but also detecting and reclassifying clean class outliers. The OCSVM-LDA hybrid method is both novel not only to the aerospace industry but also to the machine learning industry. It has worked accurately on aerospace vehicle data, but this includes the underlying assumption for the LDA that the class data has a multivariate Gaussian distribution. An option for the hybrid method could be to replace the LDA with a neural network, eliminating any assumptions for the incoming data. Further work should be done to continue improving this method as well as to compare it to the other developed algorithms for multi-class classification with novelty detection on data sets from different backgrounds.

References

[1] Cervantes N. (2022). “Engineering and Statistical Analysis of Solid and Liquid Missile Systems,” [Doctoral Dissertation, Auburn University]

[2] Weinberger, K. (2017). *Linear SVM. Machine Learning*. Retrieved 2024, from https://www.cs.cornell.edu/courses/cs4780/2017sp/lectures/lecturenote09.html

[3] Hastie, T., Tibshirani, R., and Friedman, J., *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Second Edition, Springer Science & Business Media, 2009

[4] Bishop, C. M., *Pattern Recognition and Machine Learning*, Springer Science & Business Media, 2006

[5] Murphy, K. P., *Probabilistic Machine Learning: An Introduction*, The MIT Press, 2022

[6] Greitemann, J., ”Interactive demo of Support Vector Machines (SVM)”, 2018. https://greitemann.dev/svm-demo

[7] Scikit-Learn, ”OneClassSVM”, 2024. https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html

[8] Scikit-Learn, ”LinearDiscriminantAnalysis”, 2024. https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.LinearDiscriminantAnalysis.html

[9] Scikit-Learn, ”PCA”, 2024. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[10] Tax, D.M.J., Duin, R.P.W., "Growing a multi-class classifier with a reject option," *Pattern Recognition Letters*, Vol. 29, 2008. doi:10.1016/j.patrec.2008.03.010

[11] Bodesheim, P., Freytag, A., Rodner, E., Kemmler, M., Denzler, J., "Kernel Null Space Methods for Novelty Detection," *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[12] Chan, F.T.S., Wang, Z.X., Patnaik, S., Tiwari, M.K., Wang, X.P., Ruan, J.H., "Ensemble-learning based neural networks for novelty detection in multi-class systems," *Applied Soft Computing Journal*, Vol. 93, 2020. https://doi.org/10.1016/j.asoc.2020.106396

[13] Scikit-Learn, "Comparison of LDA and PCA 2D projection of Iris dataset," 2024. https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_vs_lda.html #sphx-glr-auto-examples-decomposition-plot-pca-vs-lda-py

[14] Lian, H., "On feature selection with principal component analysis for one-class SVM," *Pattern Recognition Letters*, Vol. 33, 2012. doi:10.1016/j.patrec.2012.01.019

[15] Jupyter, "JupyterLab Documentation," 2024. https://jupyterlab.readthedocs.io/en/latest/

[16] Scikit-Learn, "train_test_split", 2024. https://scikit-learn.org/stable/modules /generated/sklearn.model_selection.train_test_split.html

[17] Pimentel, M.A.F., Clifton, D.A., Clifton, L., Tarassenko, L., "A review of novelty detection," *Signal Processing*, Vol. 99, 2014. http://dx.doi.org/10.1016/j.sigpro.2013.12.026

[18] Hsu, C., Lin, C., "A Comparison of Methods for Multiclass Support Vector Machines," *IEEE Transactions on Neural Networks*, Vol. 13, No. 2, March 2002.

[19] Spinosa, E.J., Carvalho, A., "Support vector machines for novel class detection in Bioinformatics," *Genetics and Molecular Research*, Vol. 4, September 2005.

[20] Gardner, A.B., Krieger, A.M., Vachtsevanos, G., Litt, B., "One-Class Novelty Detection for Seizure Analysis from Intracranial EEG," *Journal of Machine Learning Research*, Vol. 7, 2006.

[21] Domingues, R. (2019). ”Probabilistic Modeling for Novelty Detection with Applications to Fraud Identification,” [Doctoral Dissertation, Sorbonne University]

[22] Spinosa, E.J., Carvalho, A., Gama, J., ”Novelty detection with application to data streams,” *Intelligent Data Analysis*, Vol. 13, No. 3, 2009. doi: 10.3233/IDA-2009-0373

[23] Marsland, S. (2001). ”On-Line Novelty Detection Through Self-Organization, With Application To Inspection Robotics,” [Doctoral Thesis, University of Manchester]

[24] Kerner, H.R., Wellington, D.F., Wagstaff, K.L., Bell, J.F, Kwan, C., Amor, H.B., ”Novelty Detection for Multispectral Images with Application to Planetary Exploration,” *AAAI Conference on Artificial Intelligence*, Vol. 33, No. 1, 2019. https://doi.org/10.1609/aaai.v33i01.33019484

[25] Scholkopf, B., Williamson, R., Smola, A., Shawe-Taylor, J., Platt, J., ”Support Vector Method for Novelty Detection,” *Advances in Neural Information Processing Systems*, Vol. 12, 1999.

[26] Tax, D.M.J., Duin, R.P.W., ”Support Vector Data Description,” *Machine Learning*, Vol 54., 2004.

[27] Yi, Y., Shi, Y., Wang, W., Lei, G., Dai, J., Zheng, H., ”Combining Boundary Detector and SND-SVM for Fast Learning,” *International Journal of Machine Learning and Cybernetics*, 2021. https://doi.org/10.1007/s13042-020-01196-2

[28] Zhu, F., Yang, J., Gao, C., Xu, S., Ye, N., Tongming, Y., ”A weighted one-class support vector machine,” *Neurocomputing*, Vol. 189, 2016. http://dx.doi.org/10.1016/j.neucom.2015.10.097

[29] Silva, S., Vieria, T., Martinez, D., Paiva, A., ”On novelty detection for multi-class classification using non-linear metric learning,” *Expert Systems With Applications*, Vol. 167, 2021. https://doi.org/10.1016/j.eswa.2020.114193

[30] Zhu, F., Zhang, W., Chen, X., Gao, X., Ye, N., ”Large margin distribution multi-class supervised novelty detection,” *Expert Systems With Applications*, Vol. 224, 2023. https://doi.org/10.1016/j.eswa.2023.119937