Anomaly Detection Through a Bayesian Support Vector Machine

Vasilis A. Sotiris
Department of Mathematics
PhD Candidate in Applied Mathematics and Scientific Computation
University of Maryland, College Park, MD
vsotiris@math.umd.edu
Michael Pecht
Department of Mechanical Engineering
Director of the Center for Advanced Life Cycle Engineering (CALCE)
University of Maryland, College Park, MD
pecht@calce.umd.edu

Abstract

This project investigates the use of a one-class support vector machine algorithm to detect the onset of system anomalies and trend output classification probabilities as a way to monitor the health of the system. In absence of "unhealthy" (negative) information, the marginal kernel density estimate of the "healthy" (positive) distribution is used to construct an estimate of the negative class. The output of the one-class SV classifier is calibrated to posterior probabilities by fitting a logistic distribution to the predictor model in effort to reduce and manage false alarms.

1. Introduction

With increasing functional complexity of on-board autonomous systems, there is now an increasing demand for early system level health assessment, fault diagnostics, and prognostics. In the presence of high complexity and remote inaccessibility, the health of electronic parts and systems is difficult to monitor, diagnose and predict. Due to the micro scale packaging and material properties of the integrated components on electronic systems, performance and physics of failure (PoF) models are still uncommon and or intractable in application. There is a need for a fast and dependable way to detect when these systems are degrading, or have sustained a fault or failure that is critical. Also there is a need to predict their remaining useful life. In the absence of sustainable PoF models, a data driven approach in the machine learning framework is suitable for the first task, that of anomaly detection.

A critical part of detection and in general health monitoring is the management of false and positive alarms. Decisions made by the algorithm will not always be ideally 100% accurate, and management of this accuracy is important. False alarms occur in the training and in the evaluation stage. In the training stage, the algorithm uses the training data to construct the predictor model, a model that will function as a decision boundary, inside of which incoming new observations will be classified as positive/healthy and outside of which negative/abnormal. Note that a negative classification is not necessarily unhealthy, but is at least abnormal. The diagnosis of health will need to consider other factors including further knowledge of the system itself. In the training stage, false alarms occur when some training data are misclassified; theoretically all training data should belong to the positive class. In the evaluation stage, the algorithm classifies new observations against the predictor model constructed in the training stage. Here false alarms refer to the algorithm's generalization ability; to correctly classify data for the system it was trained on. High false alarm rates can be indicative of bad training or indeed an unhealthy system.

With an identifiable negative class, training an SV classifier is straightforward and a separation boundary can be computed. In absence of reliable negative class data this training becomes a challenge. One suggested approach is to use the origin as the negative class and maximize the margin only between the origin and the positive class data. This approach can lead to an overly optimistic decision boundary and as a result a higher rate of positive alarms. A more conservative approach is to use the marginal density of the positive class to estimate the negative class data. This step is important because failure or fault characteristics are constructed as conservative as possible by assuming that the fault space is a) Not linearly separable from the training data, b) is prevalent in the space not occupied by the training data and therefore c) conforming to the distribution of the training data with some probability. These assumptions about the nature of faults in relation to what is considered "healthy" can in turn lead to a situation of over-fitting the training data leading to higher rates of false alarms. As a remedy, degrees of classification and posterior classification probabilities can be used to measure uncertainty and in such minimize and manage the false alarms.

An important consideration for this work is to design a detection algorithm that can be used in real time, which means that it has to be fast and robust. A main focus is to process high dimensional and correlated parameter information fast without compromising the original information. For this, a principal component decomposition is used to compress the training data into two or more lower dimensional distributions. In this project the data are decomposed into two lower dimensional spaces called the model and residual spaces; one that estimates the maximum variance and the other the error in the principal component model selection [6]. The compressed data will retain most of the original information and provide insight to the variance of the system, which can be used to detect anomalies, and is anticipated to enhance the interpretation of the SVM output. Additionally by compressing the data in such a way, we can overcome the computational intractability of high dimensional kernel density estimate computations.

1.1. Project Objectives

The objectives of this project were to a) develop a toolset for the use of anomaly detection in multivariate system data sets, b) provide a detection accuracy probability at each system evaluation and c) perform analysis in the absence of failure or fault information about the system. Additionally d) compare the performance accuracy of the toolset, which I called CALCEsvm to a commercially available and tested software called LibSVM.

1.2. Project Schedule

The schedule for the completion of the project was planned to start in September 2007 and finish on May 19th 2008. A detailed schedule and milestone chart with descriptions is provided in Appendix B.

1.3. Data Structure and Description

As illustrated in Figure 1, data is collected at times T_i from a multivariate distribution of random variables $x_{1i} \ldots x_{mi}$, where x_i*'s* are the system covariate random variables. Each row observation called X_i are independent random vectors consisting of $x_1, x_2, \ldots x_m$. We are interested in estimating the conditional class $\{+1,-1\}$ of X_i*'s* and the corresponding posterior class probability $p(class|X_i)$.

		given						estimate	
		Ti	**x1**	**x2**	**x3**	**…**	**xm**	**Class**	**Class Probability**
X	**X1**	T1	x11	x21	x31	…	xm1	1	0.95
	X2	T2	x12	x22	x32	…	xm2	1	0.96
	Xn	Tn	x1n	x2n	x3n	…	xmn	-1	0.45

Figure 1 – Data Structure

1.4. Report Organization

This report is organized as follows: section 2 gives an overview of the algorithm functionality. Section 3 discusses the theory and methodology for principal component model decomposition. Section 4 discusses the two class support vector machine theory and the approach to novelty detection with support vector classification. Section 5 discusses the approach taken to develop a one-class support vector based classifier. Section 6 discusses the method for estimating the posterior class probability based on the output of the support vector classifier. Section 7 outlines the construction of the joint posterior class probabilities that are based on the models discussed in section 3. Section

2. Algorithm Overview

Figure 2 illustrates the approach methodology. The multivariate training data $X \in R^{nxm}$ where n is the number of observations and m the number of parameters is pre-processed first through a Principal component analysis. The principal component analysis (PCA) is used to decompose the signal into two or more orthonormal subspaces, in this project into two subspace, the model [M] and the residual [R] subspaces. The distribution of the projected data in the model subspace is used to estimate the maximum variance in the original parameters and the distribution on the residual is used to test the fit of the model to the data. Greater variance in the residual distribution is an indication of a poorly chosen model subspace. In addition, the residual subspace is anticipated to uncover hidden behaviors in the system degradation by highlighting abnormal variation in parameters that are overshadowed by dominant ones, usually present in the model subspace.

The decomposition of the training data X into more than two subspaces as illustrated in Figure 2 constructs m orthonormal subspaces which can be used to estimate the joint posterior class probability (Jp). The benefit of the multiple models is that they independently capture a unique identifiable subset of information related to the covariance of the random variables in X. The dimension for each model can be chosen to be as low as 1 and as high as $m-1$ with m potential models for each respectively, considering all the possible combinations. In this project, only two models are considered, each of which is two dimensional. The reason for this selection is largely due to computational constraints in computing the kernel density estimates described next. Kernel density computations in three and higher dimensions have a computation cost of order 2 larger for each increase in dimension. The dimensionality of the models is an important factor in the accuracy of the algorithm because too few dimensions might loose too much information about the correlation structure of the variables and too high might not reflect enough decomposition detail to benefit the detection accuracy.

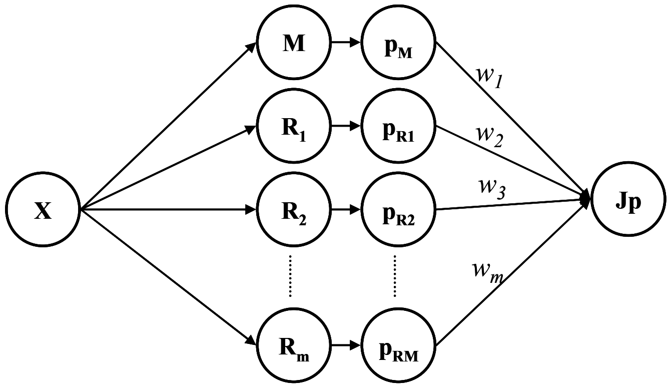

Figure 2 – CALCEsvm algorithm flow diagram.

A kernel density estimate (KDE) is computed for the two models [M] and [R] to estimate likelihood of the positive class (training data) and from it construct the negative class for each respectively. The SV classifier, constructs two predictor models $D_1(y_1)$ and $D_2(y_2)$ for each model respectively. A soft decision boundary is constructed by fitting the training data with a model for posterior class probabilities using a logistic distribution that maps classified data to posterior classification probabilities. The joint class probability from the two subspaces will in the end be used for the decision classification.

Because support vectors produce an un-calibrated value that is not a probability, the algorithm uses the support vector classifier to produce a posterior probability *P(class|input)* according to a bayesian formulation. The predictor function can benefit from an uncertainty estimate associated with each prediction and give realistic interpretations for the classification output, reduce the number of false alarms. Finally the joint posterior class probability can be weighted with a weight vector $W = [w_1 \dots w_m]$ to emphasize some models as opposed to others. This weighting could be beneficial when one model, usually the principal model [M] captures more of the data covariance information. In this project all models are weighted equally, $W=I$. The CALCEsvm function are listed and described in Appendix A.

3. Principal Component Analysis

Subspace decomposition into Principal Components can be accomplished using singular value decomposition of the input data X [3] [4]. The SVD of data matrix X, is expressed as $X=USV^T$, where $S=diag(s_1,\dots,s_m) \in R^{nxm}$, and $s_1>s_2>\dots>s_m$. The two orthogonal matrices U and V are called the left and right eigen-matrices of X. Based on the singular value decomposition, the subspace decomposition of **X** is expressed as:

$$X = X_s + X_r = U_s S_s V_s^T + U_r S_r V_r^T \qquad (1)$$

The diagonal matrix S_s are the singular values $\{s_1,\dots,s_k\}$, and $\{s_{k+1},\dots,s_m\}$ belong to the diagonals of S_r. The set of orthonormal vectors $U_s=[u_1,u_2,\dots,u_k]$ form the bases of signal space S_s. The original data is decomposed into three matrices, U, S and V, where matrix U contains the transpose of the covariance eigenvectors. The original data ***X*** is projected onto the signal subspace as defined by the principal model [M]. In terms of the SVD of ***X***, the projection matrix H can be expressed as UU^T where $u=USV^T$. Then the projection of vector ***x*** onto [M] can be expressed as $\mathbf{x}_{[M]}=UU^T\mathbf{x}$ and onto the residual subspace as $\mathbf{x}_{[R]}=(I-UU^T)\mathbf{x}$. Any vector ***x*** can then be represented by a summation of two projection vectors from subspaces S_s and S_r, where the total dimension size of [M] + [R] sums up to the original data dimension.

$$x = x_{[M]} + x_{[R]} = P_M \vec{x} + (I - P_R)\vec{x} \qquad (2)$$

$P_S=UU^T$ and $P_R=I-UU^T$ are the projection matrices for the model and residual subspace respectively, where both subspaces respectively comprise the total data dimension. In this framework, we can apply SVC having oriented the data such that we can better capture system faults that are due to changes in variance, changes in correlation and changes in the distribution of the data. In this framework we can break down the effects of multivariate data into separate and independent models, each examining a different effect of the data, and envision combining the results in the end to achieve a global detection result.

4. Overview of Two-Class Support Vector Machine Theory

Support vector machines (SVMs) alleviate the need for algorithms with statistically grounded frameworks, that is, ones that require knowledge of the distribution of the random variables. This makes the use of SVMs very practical in providing system level health decisions with a minimal footprint on computational resources. SVMs are also trained to generalize well with a reduced training set making the training of SVMs much simpler and economical.

We first introduce the theory behind hard margin classification to build the framework for the soft margin and nonlinear classification in subsequent sections. The hard-margin theory is the basis for all further analysis. In hard-margin classification, training data are linearly separable whereas in soft and nonlinear classification the training data are mostly not linearly separable. Suppose ***x*** is the input vector, y is the class label, d is the number of dimension and n is the number of samples. Training data (x_i,y_i) where $\mathbf{x} \in X^m$, $y_i \in \{+1,-1\}$ *and* $i=1,\dots,n$ can be separated by the hyperplane predictor function $D(x)$ with appropriate ***w*** and b:

$$D(x) = (w^T x) + b = \sum_{i=1}^{n} w_i x_i + b \qquad (3)$$

where $\mathbf{w}=[w_1,\dots,w_n]^T$ is the weight vector of the hyperplane and $\mathbf{x}=[x_1,\dots x_n]^T$. Thus, training data with $y_i=+1$ will fall into $D(x)>0$ while the others with $y_i=-1$ will fall into $D(x)<0$.

There are many possible separating hyperplanes which can classify the training samples, but we only need to choose one which is called the optimal separating hyperplane (OSH). Finding the (OSH) is important because it determines the accuracy of the detection and prediction process after the training. In detection the OSH defines the boundary in which new observations are considered normal/healthy and outside of which they are considered abnormal/unhealthy. To determine the (OSH), support hyperplanes are used. The input vectors pass through the support hyperplanes are called support vectors. The distance between two support hyperplanes is defined as the margin *M*. The separating hyperplane with the maximum margin is called the (OSH).

Support hyperplanes are parallel to the (OSH) $D(x)=w^Tx+b=0$, and their equations can be written as: $D(x)=w^Tx+b=k$ and $D(x)=w^Tx+b=-k$, where *k* is a positive integer. However, the above two equations are over-parameterized. If we multiply a constant to ***w***, *b* and *k*, the equations still hold. However, if we set $k=1$ so that only one set of ***w*** and *b* will be the solution the equations become: $D(x)=w^Tx+b=1$ and $D(x)=w^Tx+b=-1$. In hard-margin classification, no input data fall between two support hyperplanes. The training data are restricted to the following constraints: $w^Tx_i+b\geq+1$ for $y_i=+1$, and $w^Tx_i+b\leq-1$ for $y_i=-1$, $i=1,\ldots,n$, which can be rewritten as:

$$y_i(\mathbf{w}^T\mathbf{x}_i+b)-1\geq 0 \quad \text{for} \quad i=1,\ldots,n \tag{4}$$

where n is the number of input vectors. The margin is then given by $M=2/||\boldsymbol{w}||$, called the objective function. Therefore, the maximal margin *M* can be found by minimizing $||\boldsymbol{w}||$. To simplify computations, the objective function is squared: $2/||\boldsymbol{w}||^2=1/2\boldsymbol{w}^T\boldsymbol{w}$ and minimized subject to the constraints $y_i(w^Tx_i+b)-1\geq0$ *for* $i=1,\ldots,n$. Lagrangian multipliers are used to convert the objective function from the primal space (input space) to a dual space to simplify the calculations. The primal form of the lagrangian function can be stated as:

$$L_P(\mathbf{w},b,\alpha)=\frac{1}{2}\mathbf{w}^T\mathbf{w}-\sum_{i=1}^{n}\alpha_i y_i\left(\mathbf{w}^T\mathbf{x}_i+b\right)+\sum_{i=1}^{n}\alpha_i \tag{5}$$

The lagrangian functions in primal and dual spaces are written as L_P and L_D respectively. The idea is to minimize L_P with respect to **w** and *b* or to maximize L_D with respect to *α*. The optimal solution ($\boldsymbol{w}^*$, b^*, α^*) exists if and only if KKT conditions are satisfied. The KKT conditions state that: i) the partial derivative with respect to w and b is equal to zero ii) $\alpha_i[y_i(w^Tx_i+b)-1]=0$ and iii) $\alpha_i\geq0$ *for* $i-1,\ldots,n$. Taking the partial derivative with respect to ***w*** and *b* and setting the derivative to zero gives a set of equations to solve for ***w*** and *b*. By applying the KKT conditions we find *w* and *b*:

$$\mathbf{w}=\sum_{i=1}^{n}\alpha_i y_i\mathbf{x}_i \tag{6}$$

$$\sum_{i=1}^{n}\alpha_i y_i=0 \tag{7}$$

which in turn can be used to formulate the dual optimization problem:

$$L_D(\alpha)=\sum_{i=1}^{n}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j\mathbf{x}_i^T\mathbf{x}_j \tag{8}$$

subject to (4). Quadratic Programming is used to solve for the optimal lagrange multipliers α^* in the dual form, through which $\boldsymbol{w}^*$ and b^* can be found. In matrix form the dual problem is expressed as

$$L_D(\alpha)=-\frac{1}{2}\alpha^T\mathbf{H}\alpha+p^T\alpha \tag{9}$$

where Hessian matrix $\boldsymbol{H}=y_i y_j x_i^T x_j$, $\alpha=[\alpha_1,\ldots,\alpha_n]$ and $p^T=[1,\ldots,1]^T$ which has a size of (n x 1). The dual optimization formulation is: minimize (6) subject to (4) where $\alpha_i\geq0$.

Since $p^T=[1,\ldots,1]^T$ and $\boldsymbol{H}=y_i y_j x_i^T x_j$ (which can be calculated by the given training samples), α^* can be calculated. The weight vector $\mathbf{w}^*$ can also be found. From (ii) in the KKT conditions, when $\alpha_i>0$, $y_i(w^Tx+b)-1=0$ and $b=1/y_i-x^Tw=y_i-w^Tx$. To improve the estimate of the value of *b*, the average value of *b* is found by averaging over all support vectors:

$$b=\frac{1}{n_s}\sum_{i=1}^{n_s}y_i-\frac{1}{n_s}\sum_{i=1}^{n_s}\sum_{j=1}^{n}\boldsymbol{\alpha}_j y_i y_j\mathbf{x}_i^T\mathbf{x}_j \tag{10}$$

$$b=\frac{1}{n_s}\sum_{i=1}^{n_s}y_i-\frac{1}{n_s}\sum_{i=1}^{n_s}\sum_{j=1}^{n}\mathbf{H}\boldsymbol{\alpha}_j \quad \text{where } \mathbf{H}=y_i y_j\mathbf{x}_i^T\mathbf{x}_j$$

$$b = \frac{1}{n_s}\sum_{i=1}^{n_s} y_i \left(1 - \sum_{j=1}^{n_s} \mathbf{H}_{ji}\boldsymbol{\alpha}_j\right) \tag{11}$$

4.1. Soft-Margin Linear Support Vector Classification Theory

In real world applications data are rarely linearly separable and therefore we are interested in the nonlinear classifier. Before, we examined the theory for separable data (hard-margin), here we look at data that are inseparable (soft-margin). In this case an input data point can have an error ξ which is called the slack variable if it falls inside the margin. For $0< \xi_i<1$, the data are not well separated but still correctly classified and for $\xi_i>1$, data are misclassified. The summation of slack $\sum^{i=1,n}(\xi_i)$ is the upper bound on the errors. When the slack variable ξ is introduced, the constraints on (x_i, y_i) are always met, so feasible solutions always exist. Thus, we need to penalize the objective function by adding an error term to the optimization equation. The objective function *f(w,b)* now becomes:

$$f(\mathbf{w}, b, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 - C\sum_{i=1}^{n} \xi_i^p \tag{12}$$

subject to $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{for} \quad i = 1,\ldots,n$ where C is the margin penalty parameter that determines the trade-off between the maximization of the margin and minimization of the classification error. The slack variables ξi together with the constraints ensure that the decision function $D(x)=w^Tx+b$ selected fits the training set: almost all the data points verify that $D(x)-b\geq 0$ (ie., $\xi i=0$), and are located inside R (Figure 3). Some data points however, are such that $\xi i>0$, these are the outliers. The number of outliers is kept low by minimizing $\sum^{i=1,n}(\xi_i)$. Moreover the term $1/2\|w\|^2$ ensures that $D(x)$ has a minimum norm, which results in minimum volume for R. The dual optimization problem is given by:

$$\min L_P(w,b,\alpha,\beta) = \min!\ \tfrac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i$$
$$-\sum_{i=1}^{n}\alpha_i\left[y_i\left(w^T x_i + b\right) - 1 + \xi_i\right] - \sum_{i=1}^{n}\beta_i\xi_i \tag{13}$$

where α and β are the Lagrange multipliers for the original function $1/2\|w\|^2$ and the error term ξ respectively. The idea now is to minimize L_P with respect to w, b and ξ or maximize with respect to the non-negative Lagrange multiplier α and β. The Karush-Kuhn-Tucker (KKT) conditions give (4), (5) and

$$\alpha_i + \beta_i = C \quad \text{for} \quad i = 1,\ldots,n \tag{14}$$

The dual formulation is given by:

$$L_D(\alpha) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i{}^T\mathbf{x}_j \tag{15}$$

subject to (5) and $C \leq \alpha_i \leq 0 \quad \text{for} \quad i = 1,\ldots,n$

In order to find the optimal hyperplane $D(x)$, a dual Lagrangian $L_D(\alpha)$ has to be maximized with respect to the non-negative Lagrange multiplier α by Quadratic Programming. When $\alpha_i=0$, $\beta_i=C$ which is a positive number, $\xi_i=0$, which means that the input data $\mathbf{x}_i$ is correctly classified and it is under the constraint $y_i(w^Tx_i + b)-1\geq 0$. When $\alpha_i=C$, $y_i(w^Tx_i + b)-1+\xi_i=0$ and $\xi_i\geq 0$. The input vector x_i corresponding to $\alpha_i=C$ is called a bounded support vector which might be inseparable or misclassified, that is $y_i(w^Tx_i + b)-1\geq 0$. If $0<\xi_i<1$, x_i is correctly classified. If $\xi_i\geq 0$, x_i is misclassified. When $C<\alpha_i<0$, $y_i(w^Tx_i+b)-1+\xi_i=0$ and $\xi_i=0$. Thus, $y_i(w^Tx_i + b)=1$ and x_i is the unbounded support vector.

4.2. The Nonlinear Classifier

The basic idea in designing nonlinear SV machines is to map input vector $\mathbf{x} \in X^m$ into vectors $\Phi(x)$ of a higher dimensional *feature space F* (where Φ represents mapping $X^m \rightarrow X^f$), and to solve a linear classification problem as developed in the preceding theory in this feature space. The nonlinear decision function $D(x)$ is given by

$$D(x) = sign\left(\sum_{i=1}^{n} y_i\alpha_i\Phi^T(x_i)\Phi(x) + b\right)$$
$$= sign\left(\sum_{i=1}^{n} y_i\alpha_i k(x_i, x) + b\right) \tag{16}$$

where k is the kernel function and *sign* decides the membership of the data point between the two classes. The mathematical formulation for the nonlinear classifier is solved through the same optimization formulation as with the linear classifier. The difference is that the input space dot product is replaced by a new dot product defined by a chosen kernel function k. With the use of a kernel trick the mapping to the higher dimension can be accomplished though a dot product manipulation of the input space. The very efficiency of the SVC comes from Vapnik's principle: instead of designing *D(x)* from an estimated underlying density, we design *D(x)* directly. This avoids devoting unnecessary estimation accuracy to regions located far away from the decision boundary of *D(x)* (The limiting hypersurface in X^m enclosing *R*) and to devote high estimation accuracy to regions close to the boundary.

4.3. Support Vector Novelty Detection Through Classification

Novelty detection using SVC addresses the following problem: given a set of vectors $\mathbf{x}=[x_1,\dots,x_m]^T$ in X^m such that $[x_1,\dots,x_m]^T \sim d_0$, with d_0 unknown, is a new vector $\mathbf{x} \in X^m$ distributed according to d_0, and is considered normal under hypothesis H_0, and abnormal or "novel" under hypothesis H_1. In SVC, this problem is addressed through designing a predictor model *D(x)* (shown as the solid line in Figure 3) over region *R* in X^m and a real number *b* such that $D(x)-b\geq 0$, if $\mathbf{x} \in R$ and $D(x)-b<0$ otherwise. From the illustration in Figure 3 the points **x** which fall outside of the decision boundary are taken as outliers or abnormal observations. Notice the improved classification performance obtained using the predictor model. The predictor model *D(x)* is designed under two constraints: firstly, most of the training vectors $x=[x_1,\dots,x_m]^T$ should be in *R,* except for a small fraction of abnormal vectors, called outliers, and secondly, it must be such that *R* in X^m has minimum volume. In order to estimate *R*, or equivalently *D(x)* and *b*, we use a kernel function *k* in a higher dimensional space *F*. Space *F* can be implicitly selected by first choosing a positive definite kernel function *k*. A common choice is the Gaussian RBF kernel (where $\| \, . \, \|_{\Phi}$ denotes the norm in X^m)

$$k(x_1,x_2)=e^{-\frac{\|x_1-x_2\|^2}{2\sigma^2}} \qquad (17)$$

A positive definite kernel *k* induces a linear feature space *F* of functions that utilize a dot product. Given a positive definite kernel *k* and the corresponding feature space *F*, the support vector novelty detection approach finds a linear optimal separating hyperplane in *F* that can be mapped back to the input space X^m as a nonlinear function resulting in *R*.

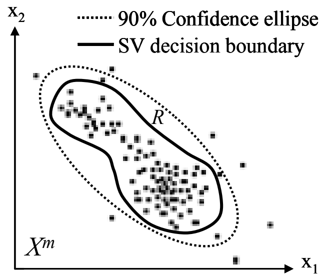

Figure 3: Illustration of the SV classifier and a 90% confidence decision boundary

From a Bayesian representation *D(x)* can be shown to be the maximum a posteriori solution to the problem of maximizing the conditional probability of correct classification given the data *x.* This means that *D(x)* is the best classifier for the given training data. This fact will be used later in the report to provide rational for the design of a posterior classification probability calibration.

5. One Class Classifier

Novelty detection in many real world, especially mission critical systems for which failures are not known, requires a predictor model that is constructed using the training data (positive class) and an estimate for the fault space (negative class). The estimate of the negative class is a conservative representation of the system fault space, an assumption that could lead to poor generalization of the algorithm in situations where the predictor model is not updated to reflect changes in the system performance characteristics. Such changes are plausible for example in a reliability setting in which the system has aged so its performance signature has changed but it is still functioning at a "healthy" state. Another example is the case where the original training data were not complete enough to represent the global system performance regimes, and in such situations the predictor model will naturally fall victim to large numbers of false alarms. Therefore a one-class-classifier approach to novelty detection is subject to complete and updatable training of the predictor model *D(x).*

5.1. Data Preparation for Kernel Density Estimation

Before estimating the density of the positive class, the CALCEsvm code uses a cleaning procedure in which the data points with low likelihood are excluded from the training population. The idea here is that data in the positive class that are outliers should not be included in the density estimation. This step in the algorithm is optional, and its use makes the one-class classifier more conservative. If it is believed that the outlier/s in the positive class are indeed representative of a "healthy" operating system then this functions should be excluded from the analysis. This is accomplished using a kernel density estimation of the data discussed next.

5.2. Kernel Density Estimation and the Negative Class

As discussed above a one-class SVM training can be required in cases where faults or failures are not available or not reliable. For this project we worked on estimating the negative class based on the assumption that the fault space is a) not linearly separable from the healthy training data, b) is prevalent in the space not occupied by the healthy training data and therefore c) conforming to the distribution of the healthy training data.

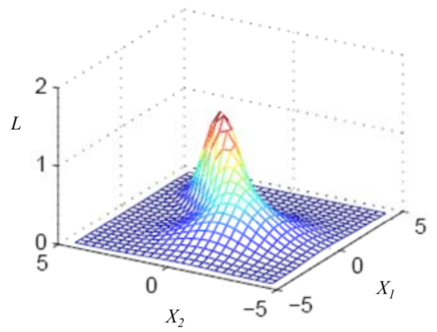

Figure 4 – Positive class kernel density estimate for bivariate distribution

To estimate the negative class $X_n \sim R^m$, we used the marginal kernel density estimate of the positive class X ~ R^m shown in illustration in Figure 4. This was accomplished by first partitioning the parameter space R^m into a grid of separate regions R^d, of length size h and dimension d. A general parzen windowing approach with Gaussian kernels is used to compute the density of each data point by centering a Gaussian kernel function φ on each point x with a bandwidth equal to the size of the grid length h. All neighboring data x_i are evaluated against the Gaussian centered at x and their corresponding influence weighted according to their distance from x. A good choice for a smooth φ is the normal *N(0,1)* function

$$\varphi(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} \tag{18}$$

where the density estimate of for parameter x is given by:

$$\hat{f}_X(x) = \frac{\sum_{i=1}^{n} \varphi\left(\frac{x - x_i}{h}\right)}{nh^d} \tag{19}$$

To overcome over-parameterized density estimates that do not generalize well, the bandwidth h is determined through a nearest neighbor approach; in which h is selected as the value that produces a volume around x containing $\sqrt{n}$ neighbors. This approach personalizes the value of h to each data point x and effectively smoothes out the density in areas of sparse information. The negative class data are constructed by selecting grid coordinates on which the likelihood of the training data is below a threshold τ. The estimated negative class data are augmented to the positive class data set and together construct the two class training data required for the two class SV classifier.

6. Posterior Class Probability

Once the predictor model *D(x)* is constructed using the positive and estimated negative training data, the argument is that *D(x)* is the best classifier, that is:

$$D(x) = \underset{a=-1,+1}{\arg\max}\, p(Y = a \mid X = x) = \begin{cases} -1, & \textit{if} \quad p(y = +1 \mid X = x) < 0.5 \\ +1 & \textit{if} \quad p(y = +1 \mid X = x) \geq 0.5 \end{cases} \tag{20}$$

The objective is to correctly classify data *X=x* by comparing the probability that the class membership of *x* is +1, vs the probability that the class membership of x is -1. The larger probability classifies *x* into the corresponding class. It can be shown that *D(x)* is the MAP solution to problem of maximizing the conditional probability of class given the data (20). Here we can start thinking about the function as a boundary, where classifications close to it will be associated with probabilities close to 0.5, and classifications far from it with probabilities closer to 1 or 0. Data that fall exactly on the boundary are randomly and fairly classified as either +1 or -1 with a classification probability of 0.5.

The classification problem defined by *p(y=+1|X=x)* can now be expressed as *p(y=+1|D(x))*, where *D(x)* is the sufficient statistic to classify data *X=x* into class +1 or -1.Intuitively because *D(x)* is the optimal classifier on which the probability of interest is exactly 0.5, then distances to it can be calibrated to probabilities. The distribution of these posterior class probabilities can be modeled by a logistic distribution centered at *D(x)=0* (Figure 5*)*. The shape parameter for the distribution can be thought to reflect the confidence in *D(x)*, a statistic dependent on the data.

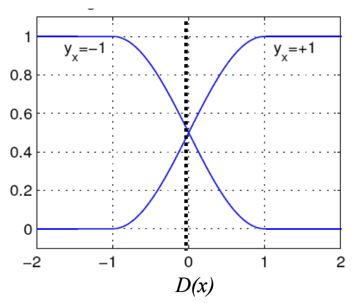

Figure 5 **-** Logistic distribution model for posterior class probabilities

The positive posterior class probability for *X=x* can be given as:

$$P(y=+1 \mid x_i) = \frac{P(Y=+1, X=x_i)}{P(X=x_i)}$$

$$= \frac{P(X=x_i \mid Y=+1)P(Y=+1)}{\sum_{a=-1,+1} P(X=x_i \mid Y=a)P(Y=a)} = \frac{1}{1+e^{-\alpha_i}} \qquad (21)$$

With the intuition that distances of data *X=x* to *D(x)* can be calibrated to probabilities leads to the justification for using a logistic type distribution to model these probabilities. From bayes rule and the law of total probability, and re-expressing the sum in the denominator, we get a function with parameter α that is a logistic type distribution (21).

$$\alpha_i = \log \frac{P(X=x_i \mid Y=+1)P(Y=+1)}{P(X=x_i \mid Y=-1)P(Y=-1)}$$

The distribution scale parameter α effects the shape of the distribution by compressing it around *D(x)=0* with large values of α and stretching it for small values. The shape of the distribution reflects the level of uncertainty in the classifier, and should be designed from the training data. From the resulting expression for α, all except one term are known, namely the probability *p(X=x_i|Y=+1)* which was estimated previously. The unknown quantities are: *p(X=x_i|Y*=-1), and the priors *p(Y=+1)* and *p(Y=-1)*. Replacing α by its intuitive interpretation, namely the data's relationship to *D(x)=0* we can evaluate the objective probability given as:

$$P(y=+1 \mid X=x_i) = P(y=+1 \mid D(x_i)) \equiv p_i$$

$$p_i = P(y=+1 \mid X=x_i) = \frac{1}{1+e^{(-AD(x_i)+B)}} \qquad (22)$$

The parameters *A* and *B* are used to optimize the posterior class distribution, of logistic form, and are estimated by maximizing the likelihood of class given data and *A*,*B*.

$$\hat{A}_{MLE}, \hat{B}_{MLE} = \arg\max_{A,B} p(X_1 \ldots X_k, c_1 \ldots c_k \mid A, B) = p(X_1 \ldots X_k \mid A, B)p(c_1 \ldots c_k \mid x_1 \ldots x_k, A, B)$$

The classification probability of a set of test data $X=\{x_1,...,x_k\}$, into a binary classification $c=\{1,0\}$ is given by a product Bernoulli distribution

$$p(c_1...c_k \mid x_1..x_k) = \prod_{i=1}^{k} p_i^{c_i}(1-p_i)^{1-c_i}$$

where p_i is the probability of classification (22) when $c=1$ $(y=+1)$ and $1\text{-}p_i$ is the probability of classification when $c=0$ (y=-1). By taking the log of the likelihood, setting its first derivative to zero and solving for A and B, we can get the maximum likelihood estimate for A_{MLE} and B_{MLE}, and in turn estimate p_{iMLE}. In this project we skip the MLE and set A to 1 and B to 0.

The evaluation of $D(x_i)$ and $sign(D(x_i))$ give the distance from $D(x)=0$ and the class label respectively. Alternatively, in this project, a function $F(x)$ is used instead of $D(x)$, which computes the Euclidian distance of x to the mean of the sample data, which usually is the origin. Data x can be evaluated with this model with

$$F(x) = \left| \frac{(w^T x + b)}{\sqrt{w^T w}} \right| \tag{23}$$

where w and b are parameters estimated through the support vector classification solution. This approach seems to provide better results than the former, as evident in the results obtained from two case studies discussed at the end of the report. Other distance based probability calibrations involve computing the distance of x to $D(x)=0$, but have not been used in this project.

7. Joint Posterior Probability Model

The end objective of the project is to compute a joint posterior class probability based on the separate and independent results from each lower dimensional model, for example, in this project the model subspace [M] and the residual subspace [R]. The joint result will provide a final classification with associated final positive and negative posterior class probabilities. This result is anticipated to give a more accurate estimate of the classification of the data $X=x$. The conditional joint posterior class probability is expressed in equation (24) with the assumption (supported earlier) that the random variables x_M and x_R are independent.

$$P(x_M x_R \mid y) = P(x_M \mid y) P(x_R \mid y) \tag{24}$$

according to Baye's rule:

$$P(y \mid x_S x_R) = \frac{P(x_S x_R \mid y)P(y)}{P(x_S x_R)} = \frac{P(x_S \mid y)P(x_R \mid y)P(y)}{\sum_y P(x_S x_R \mid y)P(y)}$$

$$P(y \mid x_S x_R) = \frac{1/P(y)P(x_S, y)P(x_R, y)}{\sum_y 1/P(y)P(x_S, y)P(x_R, y)}$$

$$= \frac{1}{P(y)} \frac{P(y \mid x_S)P(y)P(y \mid x_R)P(y)}{\sum_y 1/P(y)P(y \mid x_S)P(y)P(y \mid x_R)P(y)} = \frac{P(y \mid x_S)P(y \mid x_R)P(y)}{\sum_y P(y \mid x_S)P(y \mid x_R)P(y)}$$

$$= \frac{P(y \mid x_S)P(y \mid x_R)P(y)}{P(y=+1 \mid x_S)P(y=+1 \mid x_R)P(y=+1) + P(y=-1 \mid x_S)P(y=-1 \mid x_R)P(y=-1)}$$

where $x_R = [x_{R1}, x_{R2},...,x_{Rn}] \in R^{1\text{xm}}$ is the posterior class probability random vector from the principal subspace and , $x_S=[x_{S1}, x_{S2}, ..., x_{Sn}]$ is the posterior calss probability random vector from the residual subspace, and as before n is the number of sample observations, y is the class membership {-1 or +1} and $D(x)$ the SV classifier predictor model. In the joint probability model, $P(y=c|x_S)$ is the probability that data point $\underline{x}_M$ is classified as class c in [M], $P(y=c|x_R)$ is the probability that data point $\underline{x}_R$ is classified as c class in [R], and $P(y=c|x_S,x_R)$ is the final conditional joint probability that x is classified as class c, where $c \in C = \{-1,+1\}$. The main assumption is that the random variables on each subspace are independent, which allows formulating: therefore, the final joint probabilities of positive and negative classification are given by (25) and (26).

$$\prod_{(+)} = \frac{P(y=+1\mid x_S)P(y=+1\mid x_R)P(y=+1)}{P(y=+1\mid x_S)P(y=+1\mid x_R)P(y=+1)+P(y=-1\mid x_S)P(y=-1\mid x_R)P(y=-1)} \tag{25}$$

$$\prod_{(-)} = \frac{P(y=-1\mid x_S)P(y=-1\mid x_R)P(y=-1)}{P(y=+1\mid x_S)P(y=+1\mid x_R)P(y=+1)+P(y=-1\mid x_S)P(y=-1\mid x_R)P(y=-1)} \tag{26}$$

The final probability vectors are smoothed using an exponential smoothing approach which averages the signal of a time series of classification probabilities as a function of time

$$S(t) = \alpha\left(x(t) + (1-\alpha)x(t-1) + (1-\alpha)^2 x(t-2) + \ldots\right) + (1-\alpha)^t x(0) \tag{27}$$

The inputs, x is the joint positive posterior class probabilities and y the joint negative posterior class probabilities and the outputs are the exponentially smoothed joint positive/negative posterior class probabilities. The parameter α is chosen to give 95% of the weight to the current probability estimate, α=0.95.

8. Application to Lockheed Martin Data-Set

To test the proposed algorithm we used a data-set extracted from Lockheed Martin servers, $X \in R^{nxm}$ where n=2471 observations and m=22 parameters which we name p_1 through p_{22}. Three failure periods are known a priori, identified in the data and labeled as f_i. The failure periods are identified to occur during observations 912 – 1040, 1092-1106 and 1593-1651. The first 800 observations were used as the positive training class, whereas the remaining data was used as the test data. The training data was reduced to 140 samples chosen randomly from the original 800. The results of the algorithm (Figure 6) show promise, having detected the first two successive periods of anomaly, namely those between 912 and 1040, shown here between 112 and 240, and between 1092 and 1106 shown here between 292 and 306. The third anticipated faulty period was identified to start at observation 1593, shown here as observation 793, instead the algorithm starts to identify anomalies starting at observation 500, and indicates the system as faulty throughout the remainder of the data set. The lower joint probabilities in the second and third period of "healthy" behavior are due to the smoothing applied to the raw joint posterior probability time series, as described earlier.

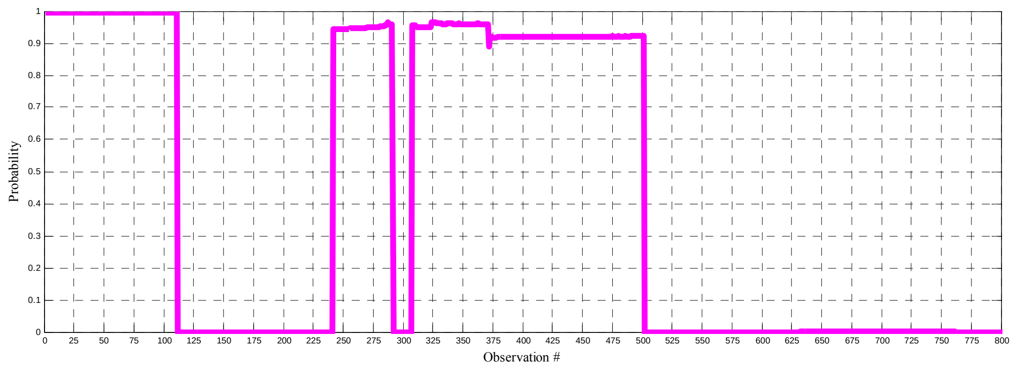

Figure 6 – Joint posterior class probabilities for test data

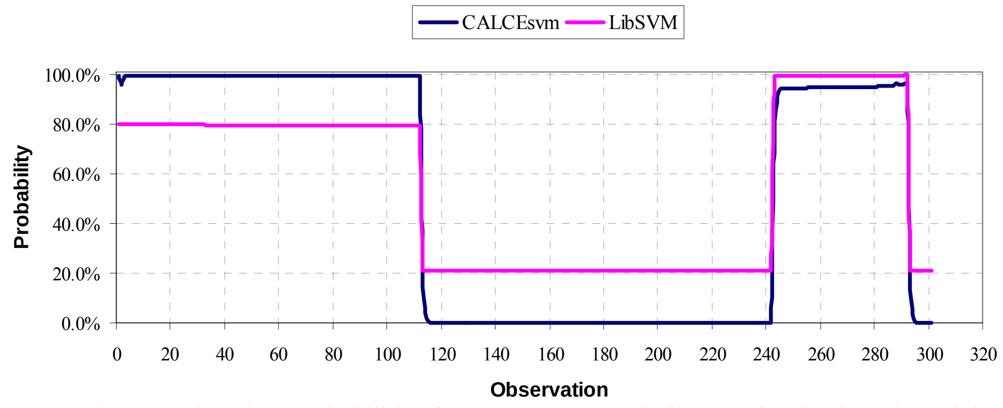

Figure 7 – Joint posterior class probabilities for CALCEsvm and LibSVM for the detection of the first faulty period

CALCEsvm results were compared to commercially available and widely used support vector classification software called LibSVM. The setup for LibSVM used its two class C-SVC setting with input the training data used in CALCEsvm. Because the one-class SVM in LibSVM does not provide posterior class probabilities, the negative class training data were taken from the output estimate of CALCEsvm. Therefore the comparison was made between two, two-class svm algorithms. The option settings used for LibSVM are:

```
-s svm_type : 0 -- C-SVC
-t kernel_type : 2 -- radial basis function
-d degree : 1, degree in kernel
-c cost : 150, margin penalty parameter
-e epsilon : 1e-5, setting for tolerance of termination criterion
-b probability_estimates: 1, outputs class probabilities into file.
```

The accuracy comparison was performed through three tests: a) a direct comparison of the quadratic optimization results: the objective function , the sum of the lagrange multipliers, and the number of support vectors, b) detection accuracy based on class index only and c) detection accuracy based on range of probabilities.

	Model Subspace		**Residual Subspace**	
	CALCEsvm	**LibSVM**	**CALCEsvm**	**LibSVM**
b0	0.1808	0.1808	0.0995	0.0995
w2	15.4	7.7	14.7	5.6
epsilon	0.00015	0.000603	0.00015	0.000509
nSV	24	23	14	14

Table 1 – Optimization results for CALCEsvm and LibSVM

Where b_0 is the bias term, $w2$ is the objective function equal to $\alpha^T H\alpha$, where $\alpha \in R^{1xn}$ is the lagrange multiplier vector and $H \in R^{nxn}$ is the Hessian matrix, where n is the length of the svm training data. Parameter epsilon is the tolerance of the termination criterion, and nSV are the total number of support vectors. The results in Table 1 indicate that the performance of the software is comparable with difference found in the objective function. The number of support vectors found was the same as well as the bias term. The difference in the objective function parameter is an indication to their respective classification output for the Lockheed data set discussed next.

The second and third tests compared the detection accuracy of the two software related to the known periods of anomaly. Each test file was coded with a column variable $z \in \{-1,+1\}$ indicating the known class of each observation. LibSVM counts the number of misclassified observations based on the coded variable z. Table 2 shows the accuracy results comparing LibSVM to CALCEsvm output. The first column in the table shows the detection accuracy based only on the class index, whereas the second column shows the detection accuracy based on a probability index. In the first comparison both performed almost identically, but the second comparison shows a clear favor towards CALCEsvm.

Libsvm Output model test			
Detection Accuracy based on class index	Detection Accuracy based on probab.	Ranges of probabilities	
99.60%	30.50%	0.8	1
	100.00%	0	0.4
Libsvm Output residual test			
99.60%	30.50%		
	100.00%		
CALCEsvm output			
100.00%	98.10%		
	100.00%		

Table 2 – Comparison of LibSVM and CALCEsvm detection accuracy

The second comparison was performed based on a probability index reflecting the expert knowledge of the system "health". This index therefore pertains to a belief and is subjective to the user settings. Non the less, this index is based on an intuitive argument, namely: that since the posterior class probabilities reflect the certainty/uncertainty of the classification/detection, then a known "healthy" and or known "unhealthy" observation should be associated with high and low probabilities respectively. In the Lockheed data-set there are two system levels: "healthy" and "faulty", both of which are entirely known for the whole data set. The "healthy" level is set to be represented by posterior class probabilities between 0.8 and 1. In light of these explanations, CALCEsvm had 0% error in its detection accuracy as opposed to a much reduced performance from LibSVM. The reason LibSVM performed at 30.5% accuracy is because two out of three periods with "healthy" level operation where captured with a posterior class probability at around 0.75 to 0.78. LibSVM as did CALCEsvm captured the faulty periods with 100% accuracy. These results reflect the comparison for this specific data-set. For complete and more concrete comparisons more testing and validation is needed.

9. Application to a Simulated Degradation

A second case study is performed using a simulated correlated data set consisting of three random variables from three different but dependent distributions to construct the training set. The objective in this case study is to test the algorithms in a setting in which a system is degrading, and in which the degradation takes place in the presence of considerable noise. Copulas are used to build a simulation model consisting of three random variables: *Gamma(2,1)*, *Beta(2,2)*, and *t(5)*. Copulas are functions that describe dependencies among variables, and provide a way to create distributions to model correlated multivariate data. A bivariate copula for example is a probability distribution on two random variables, each of whose marginal distributions is uniform. These two variables may be completely independent, deterministically related (e.g., $U_2 = U_1$), or anything in between. The family of bivariate Gaussian copulas is parameterized by $\rho= [1\ \rho;\ \rho\ 1]$, the linear correlation matrix. U_1 and U_2 approach linear dependence as ρ approaches +/-1, and approach complete independence as ρ approaches zero. Using a copula, one can construct a multivariate distribution by specifying marginal univariate distributions, and choosing a particular copula to provide a correlation structure between variables. Bivariate distributions, as well as distributions in higher dimensions, are possible. The Gaussian and t copulas are known as elliptical copulas and can generalize higher number of dimensions. Here we simulate data from a trivariate distribution with *Gamma(2,1), Beta(2,2)*, and *t(5)* marginals using a Gaussian copula.

Test data where generated from the trivariate distribution of *Gamma*, *Beta* and *t* random variables and is setup such that three degradation periods are generated. The first period is designed to be "healthy", the second introduces a shift in each variable separately while maintaining the correlation structure, and the third period a larger shift in mean. The anticipation is that the algorithm will capture the trend of degradation present in the simulated data, and correctly classify each observation according to some event rules, similar to those applied for the Lockheed data, but in this case with more event levels, specifically four event levels illustrated by the CALCEsvm output shown in Figure 8.

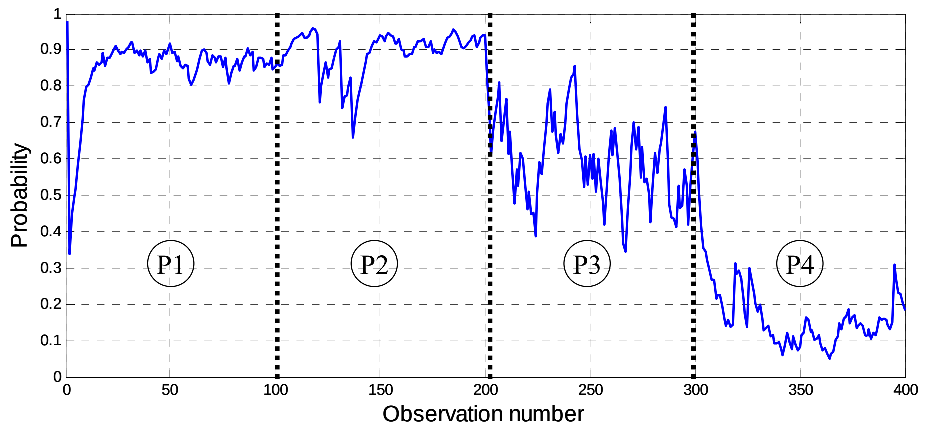

Figure 8 – Joint positive posterior class probability for simulated degradation levels P1 through P4

The CALCEsvm results are shown in Figure 8, with the four periods identified by breaking perforated lines and an index *P1* through *P4*, where *P1* is the indentifier for the "healthy" period, with mean equal to nominal, and *P2* through *P4* have successively increasing changes in mean. The result of the algorithm shows the ability to capture the trend of simulated degradation in the presence of considerable noise. The beginning period that shows a dip in the probability estimate is a direct result of an initial oversmoothing (implementation of the exponential smoothing). The larger result though is the algorithms ability to correctly classify the data for each period of operation and to capture the trend. CALCEsvm results where compared to the results obtained from LibSVM and tabulated in Table 3. The probabilities as in the Lockheed Martin case study, again reflect a belief about the interpretation of the posterior class probabilities. In this case, posterior class probabilities between 0.8 and 1 are acceptable for a "healthy" system, probabilities between 0.7 and 0.85 acceptable for the next level of "health" allowing for some overlap, and so on.

Libsvm Output			
Based on Class Index	Based on Probability	Ranges for probability	
1	1	0.8	1
0.038	0.144	0.7	0.85
0.29	0.46	0.3	0.7
0.88	0.79	0	0.4

CALCEsvm output			
Based on class index	Based on probability	Ranges for probability	
1	0.8125	0.8	1
0.038	0.317	0.7	0.85
0.29	0.31	0.3	0.7
0.88	0.84	0	0.4

Table 3 – CALCEsvm and LibSVM accuracy results for simulated case study

The comparison of accuracy results based on the class index show that both algorithms performed virtually identically for the given probability ranges. Both CALCEsvm and LibSVM had a detection accuracy rate of 100% in P1, noticeably in P2 both algorithms perform poorly, and improve in P3 and P4 to 88% when the degradation becomes more distinct. The comparison of accuracy results based on the posterior class probabilities shows a slight improvement in the performance of each algorithm for P1, and about the same performance for the other periods. CALCEsvm performed better than LibSVM in detecting the anomalies with the appropriate probability in periods P2 through P4. Figure 9 plots the detection accuracy of CALCEsvm and LibSVM vs the start value for the probability index for level 2 (P2). For example, from the plot, it can be seen that when the lower bound on the probability index is 55% and the upper limit fixed at 100% CALCEsvm has a detection accuracy of 96% versus approximately 89% for LibSVM.

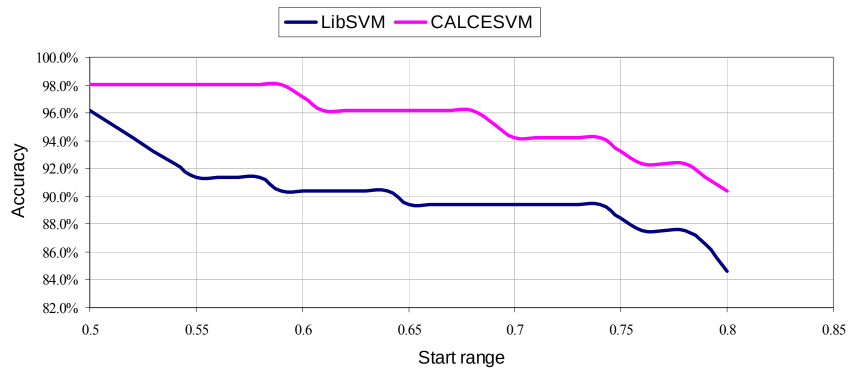

Figure 9 – Comparison of CALCEsvm and LibSVM detection accuracy for level 2 (P2)

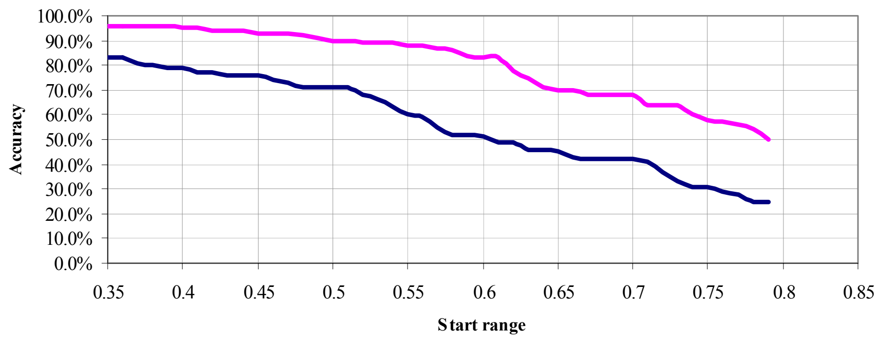

Figure 10 – Comparison of CALCEsvm and LIbSVM detection accuracy for the simulation case study for operation level 3 (P3)

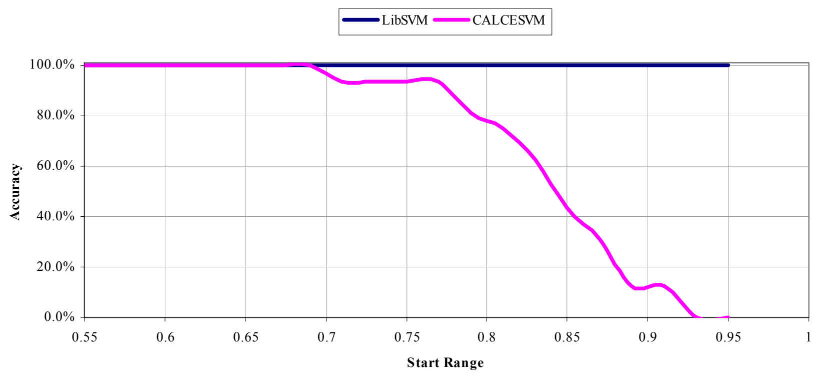

Figure 11 – Comparison of CALCEsvm and LibSVM detection accuracy for the simulation case study for operation level 1 (healthy)

In Figure 11 the accuracy of CALCEsvm is sharply reduced when the requirement of the probability index becomes more stringent. Here for example, when the lower bound on the probability index is 90% and the upper 100%, CALCEsvm has a low ~10% detection accuracy, whereas LibSVM has a much better performance 100%. This result here as mentioned earlier is due to the over-smoothing of the joint probabilities in the early stages (can also be seen in Figure 8).

10. Summary and Conclusions

At the start of the project the goal was to develop an algorithm based on a support vector machine framework that would detect the onset of anomalies in multivariate systems. In the absence of fault and failure data, the objective grew to include estimating that information in a conservative approach through the use of non-parametric density estimation. The needs of the algorithm grew to accommodate required functionality and modifications to enhance the accuracy of detection; the nearest neighbor optimization of the density estimation bandwidth, an optimal grid selection for the estimation of the negative class, posterior class probability calibration through distance computations, etc. The decomposition of the training data into two lower dimensional models was useful because it allowed detection on each separately, and from each to extract different performance degradation characteristics. The end joint posterior classification probability tied together the results from each model to determine the "combined effect" detection.

To test and validate the CALCEsvm implementation of these algorithms, simulated and real data were used against both CALCEsvm and commercially available software for support vector machine analysis called LibSVM. A simulation test-bed was created to model time series data indicative of a degrading multivariate system, with correlated system parameters. Metrics were used to measure the detection accuracy of both algorithms based on a) the class index only, b) on their posterior probability estimates for known system states, and given probability ranges. It was found from the two case studies that CALCEsvm performed on average better than LibSVM. These results are not in any way generalizable, and need extensive cross-validation. The point is that for these metrics and this particular data CALCEsvm so a favorable performance in constract to LibSVM.

Allthough the CALCEsvm implementation proved successful in meeting the goals set in this project and has proven useful for detection in multivariate systems, it has some limitations and areas for great improvement. Its limitations lie in two key areas: a) The estimation of the negative class can in many cases be over-conservative, depending on how the user specifies the threshold τ, and or if the number of training samples is to small. Also, in this area a limitation also arises in the selection of the grid size and dimension. Not much attention has been put into automating and optimizing the grid construction as to best and most accurately compute the kernel density estimation. The second key area is decomposition of the training data into two dimensional subspaces. This was used in this project because KDE on higher dimensional spaces is costly and therefore time consuming. For a proof of concept in this project we only used two dimensional subspace models. This was also designed to better visualize the data and validate the results. In many circumstances, especially when the multivariate data are highly correlated then two dimension are potentially justifiable and reasonable. For the Lockheed data for example, retaining only two dimensions to construct each subspace model only retained 45% of the original information available in the training data. In data sets with more than 6 or 7 parameters, usually more than two dimensional models are required. In the same are, another limitation is the fact that only one residual model was used, totaling two subspace models. This was again done to keep computational time low. A parallel processing approach that could run the classification independently in each of the possible combinations of residual models as shown in Figure 2 could have enhanced the detection accuracy of the algorithm and reduced the influence of noise.

11. Acknowledgements

I would like to thank Dr. R. Balan for his help, patients and time spent discussing the project with me, as well as Dr. A. Zimin for constructive feedback and ideas throughout the course. I would also like to acknowledge my advisor Dr. M. Pecht for his guidance, motivation and support of my ideas and work.

12. Appendix A

CALCEsvm Functions

Function calcesvm.m – is the main function to call

>>out=calcesvm(ker,p1,C,straind,datatype,plottype), and replace ker by 'rbf', p1 by the degree of the kernel, usually 1, the margin penalty paremter, usually 100 to 150, the size of the training data, which is used in simulations, the data type, choose 'sim0', 'sim1', etc… for various degradation simulations, use 'load' to load an excel file, and 'input' to type in the data. For plottype, set to 'on' to view the result plots.

>>[pos,t]=getdata(datatype,straind); – code to simulate or load file. Currenlty there are five simulations 'sim0' through 'sim4', each of which has a different model for degradation. Input: the user input command on simulation or input or load and the size of the data if simulation. Output: the positive class data matrix *pos* and a test data matrix *t*.

>>[pos,t]=normalize(pos,t); – code to normalize training data *X* and test data T by subtracting by the mean of the training data and dividing by its standard deviation for both. Input: the positive class data pos and the test data matrix t. Output: the normalized pos and t.

>>[posm,tm,k]=pca('m',pos,t); - code to decompose X into two lower dimensional subspaces according to a linear principal component analysis. The principal subspace is constructed using the first two principal components and the residual subspace the two last. Input: the principal or residual model indication 'm' or 'r', the positive class and the test data. Output: the transformed positive class and test data, and the model dimension k.

>>[gm,negm,dm,cm,xcm,ycm] = parzen2a(posm(:,[1:k]),xmin,xmax,ymin,ymax,a,b); - This function estimates:

a) the kernel density of the positive class *dm*
b) the negative class *negm*
c) the contour plots of the positive class *cm*

and provides the grid points used for the kernel density estimation *xcm* and *ycm*. Input: the transformed positive class data for either the principal or residual subspace, the dimensions of the grid for the KDE. Currently this is set manually to allow user to optimize the dimensions to best estimate the negative class. User set parameters a and b which are used to better define a threshold τ with which to construct the negative class. The threshold t is inversely proportional to a, $t \propto 1/a$, large values for a make the threshold more strict. Parameter b takes values between 1 and 2 (although can take any non-negative value) and it regulates the size of the negative class, values of b close to 1 minimize the size of the negative class. This is important to keep the computations (for this code) reasonably fast.

Within parzen2a.m there are three functions:

>>sigma=smoothparam(pos,pos(j,:),length(pos)); - This function estimates the optimal band width sigma for each data point *pos(j,:)* by counting the nearest neighbors. The required size of nearest neighbors is taken as the square root of n, the total number of samples.

>>pos=cleanpos(pos,L); - This function is used to clean (delete data with low likelihoods) the positive class data pos based on their likelihood L.

>>[p,c,neg]=negclass(pos,p,x,y,mar,a,b); - This function is used to estimate the negative class based on the positive class *pos* data and their corresponding likelihoods *p*. Input: positive class data *pos*, positive class likelihoods *p*, grid coordinates *x, y,* threshold mar (τ), and parameters a and b described above. Output: The new density profile *p* and contours *c* for the given grid that includes altered densities that reflect the coordinates of the negative class (colored in orange). And the estimate for the negative class *neg*.

>>[b0m,alpham,epsilonm,wm]=svc(C,n,ker,xm,ym); - This function is called from calcesvm.m and is the core support vector classification part of the algorithm. Input: the margin penalty parameter C, the samples size n, the kernel used, the training data for either the principal model *xm* and its corresponding class index

vector *ym* or the residual model *xr* and its corresponding class index vector *yr*. Output: the quadratic optimization solution parameters.

Whithin svc.m there are three functions:

>>svkernel(ker,X(i,:),X(j,:)); - This function computes the appropriate kernel for the given data. Usage: k = svkernel(ker,u,v). Parameters: ker - kernel type u,v - kernel arguments Values for ker: 'linear', 'poly' - p1 is degree of polynomial', 'rbf'   - p1 is width of rbfs (sigma).

>>[alpha lambda how] = qp(H, c, A, b, vlb, vub, x0, neqcstr); - This function computes the quadratic optimization problem. Input: the Hessian matrix H, c a unit vector, A class label vector, b=0, vlb and vub set the bounds for the support vectors, x0 starting point is zero (vector), and neqcstr sets the number of equality constraints (1 or 0) to 1.

>> Dsm=distance(dsm(:,1:k)',zeros(size(dsm(:,1:k),2),size(dsm(:,1:k),1))); - This function computes the Euclidean distance between two vectors A and B. $||A-B||=sqrt(||A||^2 + ||B||^2 -2.A.B)$. Input: test data and origin.

>>[classm,idxm,pnm1,ppm1]=svmDetection(xm,ym,b0m,alpham,epsilonm,ker,tm(:,1:k)); - This function computes the class index and the posterior class probabilities for the test data *tm* and *tr*. The probability calculation *pnm1* and *ppm1* are based on evaluating *tm* on *D(.), ppm1=1/(1+exp(D(tm))), pnm1=1-ppm1*. The class is determined by *sign(D(x)) = sign(SUM(y(i)*alpha(i)*K(xi,x)), i=1:n*. Input: the training data (positive + negative), class labels, SVM parameters, kernel function and the test data. Ouput: class index classm and classr, index, and posterior class probabilities ppm1 and pnm1.

>>[pnm,ppm]=probfit(Dsm(:,1),classm); - This function provides the alternative posterior class probability calculations based of *F(.)*, distance from the origin. Input distances from origin and class index. Output, posterior class probability based on distance from origin.

Plotting options

>>zs=plotsvc(xm,ym,b0m,epsilonm,alpham,ker); - This function plots the positive and negative class data as well as the predictor function *D(x)*, and the support hyperplanes at *D(x)=-1* and *D(x)=+1.*

13. Appendix B

Project Schedule

PSVC Project Schedule		Oct-07				Nov-07				Dec-07				Jan-08				Feb-08				Mar				Apr				May			
Probability Model	Joint Probability model for SVM output	0	0	50	50	50	70	80	90	90	95	95	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100		
	Matlab - SVM Probability function fit	0	0	0	0	50	50	70	80	90	95	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100		
	Design logic for output decision function	0	0	0	0	0	0	0	0	0	10	10	20	30	30	40	40	30	30	30	40	45	50	60	70	70	80	100	100	100	100		
	Matlab code for joint probability model	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	40	40	50	70	80	85	100	100	100	100	100		
Support Vector Machines	SVM Theory document	0	20	20	20	20	50	70	70	75	80	90	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100		
	LibSVM setup	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	20	20	20	20	30	30	30	30	30	40	60	100	100	100		
	Matlab code for CALCEsvm	0	0	0	0	0	0	0	0	30	40	50	50	50	50	50	50	50	60	60	65	70	70	70	80	90	95	95	95	95	100		
	Matlab code for SVM model parameter tuning	0	0	0	0	0	50	50	90	90	90	90	90	90	90	90	90	90	100	100	100	100	100	100	100	100	100	100	100	100	100		
	Matlab code for kernel functions	0	0	0	50	50	70	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100		
	Matlab code for interface with PCA	0	0	0	50	50	90	90	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100		
	Matlab - negative class data generation	0	0	0	0	0	0	0	0	0	10	20	30	30	40	40	50	30	40	50	55	60	65	70	80	90	95	95	100	100	100		
PCA	PCA Theory document	0	0	90	90	90	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100		
	Matlab code for PCA	0	90	90	90	90	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100		
Kernel Density Estimation	KDE theory for multivariate data	0	0	0	0	0	0	0	0	0	0	0	20	30	40	40	50	30	40	60	65	70	80	85	90	100	100	100	100	100	100		
	Matlab code for KDE/SVM implementation	0	0	0	0	0	0	0	0	0	0	0	0	20	40	40	50	20	40	50	60	80	80	80	90	100	100	100	100	100	100		
	Matlab code for optimal bandwidth selection	0	0	0	0	0	0	0	0	0	0	0	0	0	10	20	30	40	40	40	60	70	80	90	90	100	100	100	100	100	100		
Data Acquisition and Storage	Training data for simulated test case	0	0	0	0	0	90	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100		
	Create test data set for simulated test case	0	0	0	0	0	50	70	80	95	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100		
	Get Experimental data (identiy faults)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	40	40	40	70	90	100	100	100	100	100	100		
	Database	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	20	20	100	100	100	100	100	100	100	100		
Testing & Validation Reports & Software	LibSVM results on test data	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	30	50	70	90		
	CALCEsvm results on test data	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	30	50	60	70	80	100	100		
	CALCEsvm full report	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	20	50	90		
	Software documentation	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	20	30	30	30	30	50	60		
	CALCEsvm C-code	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

The schedule was focused on six stages, listed in the schedule gant chard above. A) Development of a principal component analysis code to transform the data into lower dimensional subspace models. B) Develop the framework for the support vector machines developing most of the code from scratch except for the quadratic optimization algorithm. C) Develop the code for the posterior class probabilities and the final joint probability calculations. D) Develop the kernel density estimation code and integration to a support vector machine analysis. Also develop code for the optimal bandwidth selection using a nearest neighbor approach. E) Develop code to simulate correlated multivariate data that simulate a degrading system. Acquire real data from Lockheed Martin with identified faults and failures. Develop a database to access the information easily. F) Develop a methodology and code to test, validate and compare the CALCEsvm accuracy results to a commercially used SVM software, LibSVM. Document code and write up reports.

14. References

A. Smola, P. Bartlett, B. Schölkopf, D. Schuurmans, “Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods”, Advances in Large Margin Classifiers, pp. 61-74, MIT Press, (1999).

H. T. Lin, C. J. Lin, R. C. Weng, “A note on Platt's probabilistic outputs for support vector machines”, Machine Learning, Vol 68, 3, pgs 267 – 276, October 2007

W. Chu, S.Sa.Keerthi, C.J.Ong, “A New Bayesian Design Method For Support Vector Classification”, Proceedings of the 9th International Conference on Neural Information Processing (ICONIP'OZ) , Vol. 2

G. Yves, M. Johnny, B. Samy, “A Probabilistic Interpretation of SVMs with an Application to Unbalanced Classification”, Advances in Neural Information Processing Systems, NIPS 15, 2005

J. Gao, P. N. Tan, “Converting Output Scores from Outlier Detection Algorithms into Probability Estimates”, Proceedings of the Sixth International Conference on Data Mining, pp 212-221, 2006

D. Tax, P. Juszczak, "Kernel whitening for one-class classification," Proceedings of the First International Workshop on Pattern Recognition with Support Vector Machines, pp 40 – 52, 2002

P. J. Moreno, P. P. Ho, N. Vasconcelos, “A Kullback-Leibler Divergence Based Kernel for SVM Classification in Multimedia Applications”, Advances in Neural Information Processing Systems 16, MIT Press

C. J. C. Burges. "A Tutorial on Support Vector Machines for Pattern Recognition". Data Mining and Knowledge Discovery 2:121 - 167, 1998