# Rubric for Grading Assignments
# MEC E 694: Applied Computational Intelligence for Engineers

**Student Name:** Amirhossein Iranmehr

**Student Email:** iranmehr@ualbert.ca

**Assignment #:** 3

## Coding Section

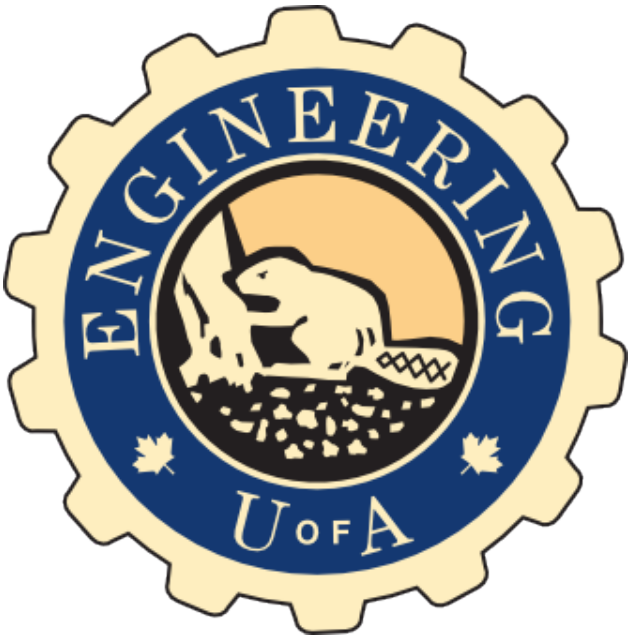
| Level / Category | Contains at least 1 error (0 marks) | Can be Executed (3 marks) | Complete and Correct (5 marks) | Mark (0-5) | Weight | Total (%) |
|---|---|---|---|---|---|---|
| **Sections 1, 2, 3, etc** | | | | | ×7.0 | |
| 35 points for all coding will be divided among the number of codes requested form you. | | | | | | |
| **General Comments** | | | | | | |

## Report Section

| Level / Category | Inadequate (0-2 marks) | Adequate (3-4 marks) | Superior Work (5 marks) | Mark (0-5) | Weight | Total (%) |
|---|---|---|---|---|---|---|
| **Results** | Results simply stated. | Results stated & interpreted. | Results stated & interpreted & compared. Parameters discussed. | | ×6.0 | |
| **Discussion & Engineering Analysis** | - Major errors in analysis.<br>- Misunderstanding the concepts. | Minor errors in analysis. | No error in calculation and analysis. | | ×5.0 | |
| **Data Presentation (Figures, Tables, Symbols)** | - Visuals & captions are poorly formatted.<br>- Symbols are not defined. | Visuals & captions are formatted appropriately. | - Visuals & captions are formatted appropriately.<br>- Symbols are defined. | | ×2.0 | |
| | | | | | **Total:** | |
| **General Comments** | | | | | | |

**Remarks:**

1. Add this page to the beginning of your reports for grading (failing to attached the rubric, or use the provided Overleaf template, or uploading your assignment codes into your GitHub repository will deduct 21% of the assignment mark).
2. Attach the video certificates related to topics covered for this assignment to the end of your report.
3. Submit one ZIP file containing your report and codes. The name of ZIP file must be your last name.
4. Name your files properly. For report: one PDF file with your last name. For codes: all completed codes with names similar to the ones provided for you. If you should complete file name "ABC," we only mark the codes in file "ABC" and not any other file.




# Applied Computational Intelligence for Engineers

## Assignment 3

**Amirhossein Iranmehr**

*Due Date: Nov. 24 at 5 pm*

# Contents

# 1 Multi-layer Perceptron

A Multi-layer Perceptron (MLP) is a type of Artificial Neural Network (ANN) used for solving classification problems. The procedure of MLP is as follows:
1. Basic Structure: An MLP consists of an input layer, one or more hidden layers, and an output layer. Each layer is made up of nodes, also known as neurons, which are connected with weights.
2. Neuron Function: Each neuron in a layer receives input from the neurons of the previous layer. The input to a neuron is a weighted sum of outputs from the previous layer, which is then passed through an activation function. Common activation functions include sigmoid, ReLU (Rectified Linear Unit), and tanh.
3. Learning Process: Forward Propagation: During this phase, the network takes inputs and propagates them forward to compute the output.
4. Back propagation: In this phase, the network adjusts its weights based on the error between the predicted output and the actual output. The error is propagated back through the network, allowing the weights to be updated via gradient descent or other optimization algorithms.
Additionally the **Training process** of MLP is as follows:

- Initialization: Randomly initialize weights.
- Forward Propagation: Compute the output for the model.
- Calculate Loss: Determine the error between predicted and actual classifications.
- Back propagation: Adjust the weights to minimize the loss. Repeat: Perform steps 2-4 for multiple iterations or epochs until the model achieves satisfactory performance.

Figure 1. illustrates the a basic MLP structure used for a binary classification problem. This figure will show the input layer with feature inputs, hidden layers with neurons, and the output layer with a binary outcome.

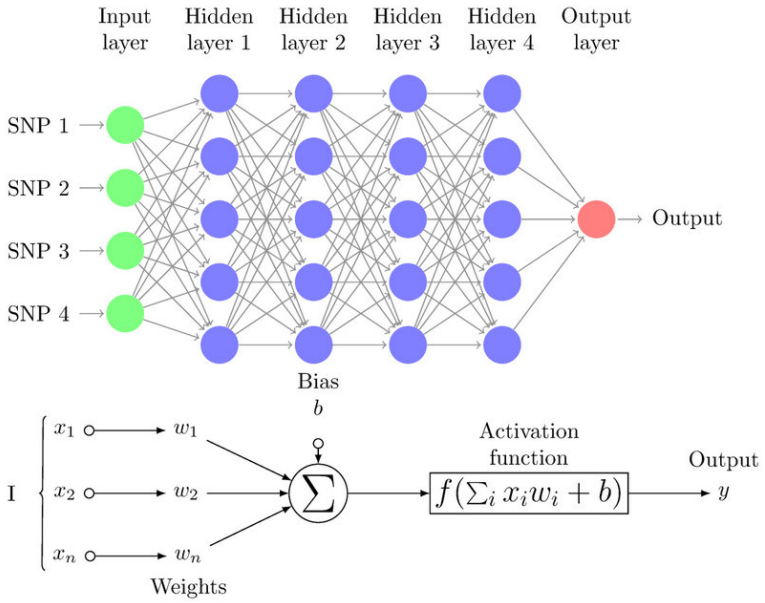
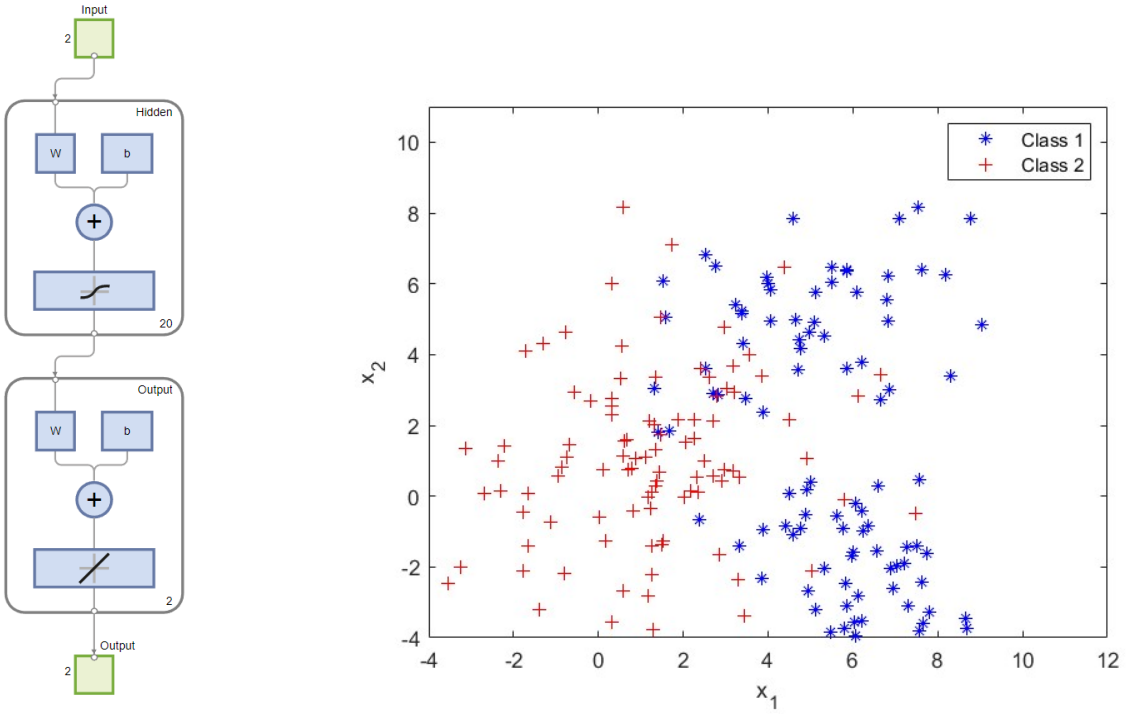



Figure 1: Diagram Multi-layer Perceptron (MLP)

### 1.1 Training with Back propagation Algorithm

In this section we aim to implement the backpropagation algorithm to train an MLP with a simple architecture (input layer, one hidden layer, and output layer) to solve a two-class classification problem and then compare your trained network with the one that can be obtained using MATLAB libraries. Eventually, we are gonna solve a binary classification based and the output should be as shown in Figure 3.

In the third segment of the script titled "ClassificationANNIncomplete", a systematic methodology is employed to execute forward propagation within an Artificial Neural Network (ANN). This process is crucial for deducing the network's output. Subsequently, for each data point, the logistic regression objective value is meticulously computed. This objective value plays a pivotal role in assessing the network's performance against the actual outputs.The objective




(a) (b)

Figure 2: a) MLP network b) Random data utilized for this study

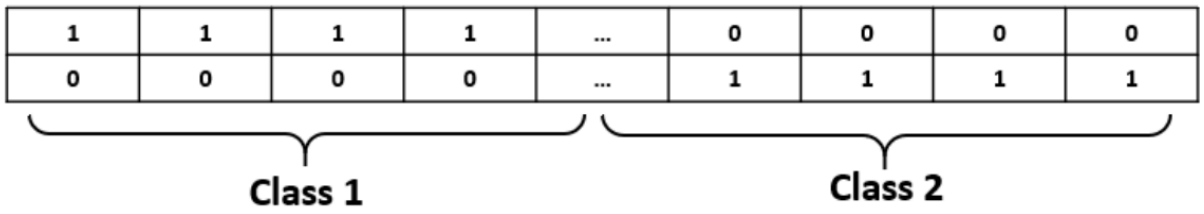



Figure 3: The roper output format for MLP

function (J) for MLP is as follows:

$$J_{x_i,\theta} = -y_i \log(\hat{y}_i) - (1 - y_i) \log(1 - \hat{y}_i) \tag{1}$$

$$J = -\frac{1}{M} \sum_{i=1}^{M} [J_{x_i,\theta})] + \frac{\lambda}{2M} \sum_{l=1}^{L-1} \sum_{i=1}^{S_l} \sum_{j=1}^{S_{l+1}} \left(\theta_{ji}^{(l)}\right)^2 \tag{2}$$

Where the right therm in the Equation 2 is the regulariztion term, and M is the number of observations.
Post the computation of the objective value, the algorithm engages in backpropagation, a quintessential training mechanism. Backpropagation is instrumental in iteratively reducing the magnitudes of Delta 1 and Delta 2. These parameters are critical as they signify the magnitude of error in the input and hidden layers of the network, respectively. Notably, the error in the hidden layer for each data point is calculated using the sigmoid gradient function. The sigmoid gradient is a derivative of the sigmoid function, which is widely used as an activation function in neural networks. This gradient helps in fine-tuning the weights of the network by providing a measure of how much each neuron's output contributes to the error.
By adjusting the weights associated with Delta 1 and Delta 2 through backpropagation, the ANN progressively learns from the training data. This learning process is aimed at minimizing the overall error, thereby enhancing the model's predictive accuracy. The use of sigmoid gradients in computing the error in the hidden layer is a testament to the intricate design of neural networks, where even the rate of change of activation functions is leveraged for model optimization.

## 1.2 Performance of Different MLP Techniques

- **1. MATLAB MLP:**
  In this section we are gonna compare the performance of the MATLAB function, and our code for two-class classification. The MATLAB multilayer perception is conducted using **"trainFcn"**. "trainFcn" can be trained using different functions. Each of these functions employs a unique algorithm for training, and their suitability can vary depending on the specifics of your neural network problem, that are as follows:

- trainlm (Levenberg-Marquardt algorithm)
- trainbr (Bayesian Regularization)
- traingdx (Gradient Descent with Momentum and Adaptive Learning Rate)
- trainscg (Scaled Conjugate Gradient):
- traingdm (Gradient Descent with Momentum)
- traingda (Gradient Descent with Adaptive Learning Rate)

After, the implementation, we are gonna compare the accuracy of our implemented code and MATLAB function. In our implementation we are gonna use 'trainlm', 'trainbr' and trainscg' for our classification problem.

**MATLAB MLP (with trainlm):** This method uses Levenberg-Marquardt optimization, which is a trust-region approach that's very efficient for small- to medium-sized networks. It can achieve superior outcomes because it combines the speed of conjugate gradient methods with the robustness of quasi-Newton methods. However, it requires more memory, which can be a limiting factor for larger networks.

**MATLAB MLP (with trainbr):** Bayesian Regularization (trainbr) adjusts the weights and biases of the network according to a Bayesian framework, which can prevent overfitting and often results in better generalization. Its performance tends to be more consistent across different network sizes but may not reach the highest accuracy due to its conservative nature in adjusting weights.

**MATLAB MLP (with trainscg):** Scaled Conjugate Gradient (trainscg) is designed to have the speed of conjugate gradient methods but uses less memory, making it suitable for larger networks. Its performance can sometimes surpass trainlm on larger networks because it doesn't require the calculation of the Hessian matrix and can still effectively navigate the error surface.

- **2. Our MLP with "Sigmoid" activation:**
  To ensure the reliability of our outcomes, we run our principal code "Classification-ANN-Incomplete" ten times for each setup, such as an MLP with five neurons in the hidden layer. We then present the mean accuracy along with its standard deviation. Note that each iteration produces a unique dataset and initializes the MLP with different random weights. For this purpose **"classperf"** function is used to evaluate the classification accuracy in each step. Then, the "CorrectRate" were stored and average accuracy ± standard deviation were calculated for different number of Neurons in Hidden Layer.

### 1.3 Comparison and Conclusion

Table 1 and 2 illustrates the average accuracy and standard deviation for 10 different set of data for MATLAB MLP, and our implemented MLP. Figures 3 is a bar chart indicating the average accuracy and standard deviation for each technique for different number of neurons in hidden layers.

Table 1: Average Accuracy for different MLP implementations with 10 different datasets

| **Number of artificial neurons in the hidden layer** | **2** | **5** | **10** | **15** | **20** |
|---|---|---|---|---|---|
| **Our MLP with "Sigmoid" activation** | 81.6 | 91.25 | 91.8 | 92.65 | 92.2 |
| **MATLAB MLP (with trainlm)** | 90.8 | 92 | 94.1 | 95.8 | 97.5 |
| **MATLAB MLP (with trainbr)** | 90.5 | 91.5 | 91.6 | 91.9 | 90.7 |
| **MATLAB MLP (with trainscg)** | 89.8 | 91.7 | 94.5 | 94.4 | 94.9 |

Based on the results shown in the previous tables and figures we could have several concluding remarks:

- Comparing the results drived from our MLP implementation and MATLAB MLP, it can be found that "Our MLP" which achieves a maximum accuracy of 92.65% with 15 neurons, it shows commendable performance but does not surpass the "trainlm" method, which leads at 97.5% accuracy

Table 2: Standard Deviation of accuracy for different MLP implementations with 10 different datasets

| **Number of artificial neurons in the hidden layer** | **2** | **5** | **10** | **15** | **20** |
|---|---|---|---|---|---|
| **Our MLP with "Sigmoid" activation** | 0.111 | 0.032 | 0.025 | 0.021 | 0.022 |
| **MATLAB MLP (with trainlm)** | 0.019 | 0.017 | 0.015 | 0.017 | 0.013 |
| **MATLAB MLP (with trainbr)** | 0.019 | 0.031 | 0.022 | 0.018 | 0.016 |
| **MATLAB MLP (with trainscg)** | 0.022 | 0.025 | 0.014 | 0.018 | 0.022 |

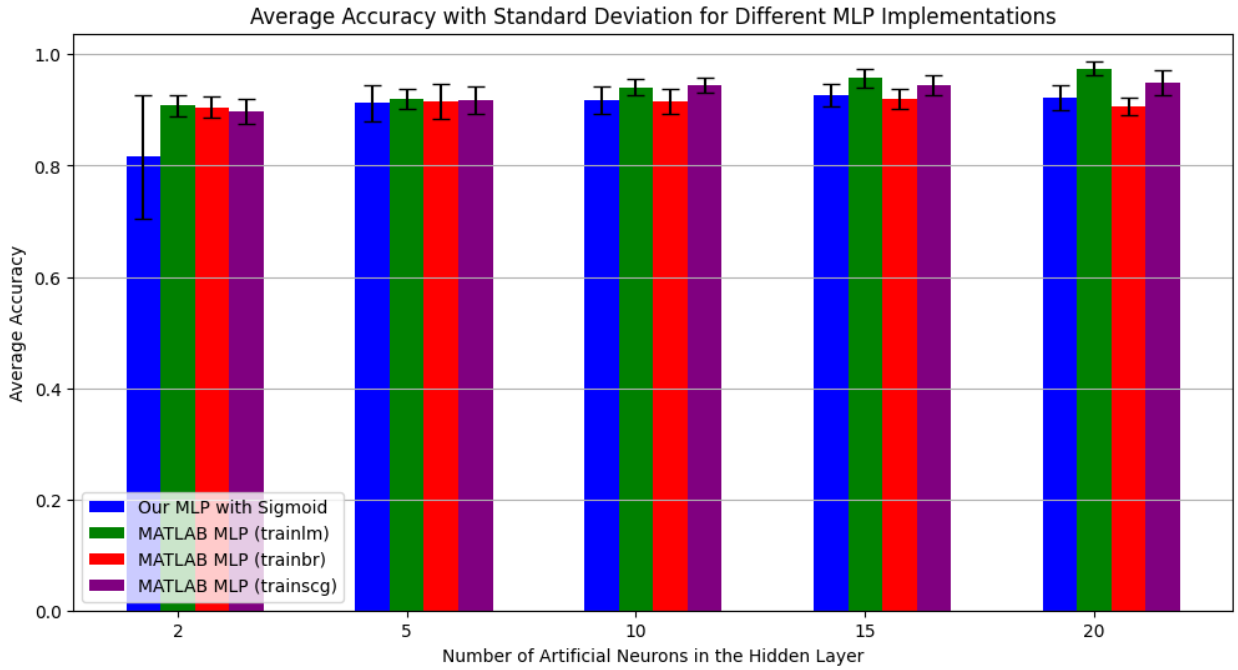



Figure 4: Average accuracy and standard deviation of each MLP technique for different number of neurons in hidden layers.

with 20 neurons. This difference in performance could be attributed to the more advanced optimization techniques used by MATLAB's "trainlm", such as the Levenberg-Marquardt algorithm, which is known for its fast convergence in medium-sized neural networks. Additionally, the discrepancy might also result from different data preprocessing, network initializations, or regularization techniques used in the MATLAB implementations

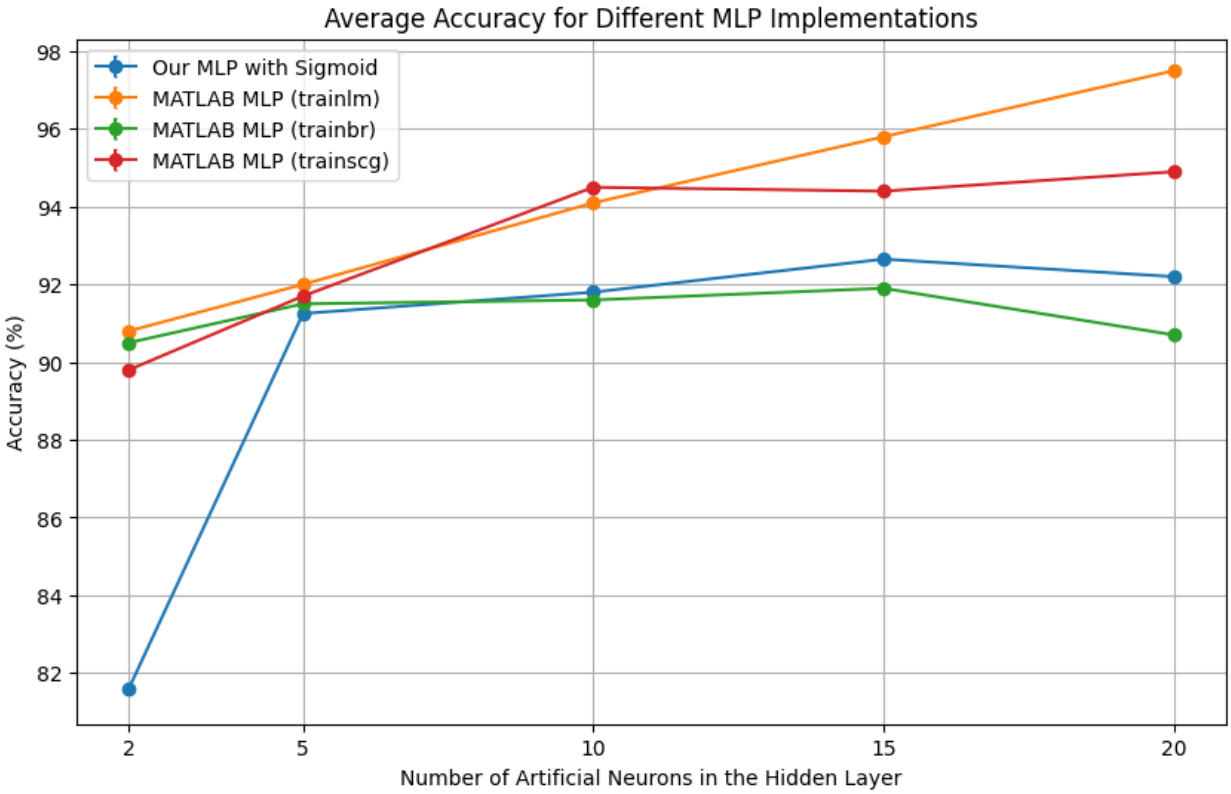



Figure 5: Average accuracy(%) of each MLP technique for different number of neurons in hidden layers.

that could lead to better generalization and performance on the given datasets. The "trainbr" and "trainscg" methods offer more conservative accuracies, peaking at 91.9% and 94.9%, respectively, with their best configurations. Overall, the "trainlm" method remains the top performer, suggesting that its optimization strategy is well-suited for the tested network sizes and complexity.

- If we want to implement MLP with any desired number of hidden layers with our code we have to make some changes in the code's strucuture.This involves altering the network initialization code to iterate over a list or array of neurons per layer, creating a corresponding number of hidden layers with the specified number of neurons. Additionally, the weight and

bias initialization, along with the data flow between the layers during forward propagation and error propagation back through the network during backpropagation, must be adapted to handle the variable number of layers.

- To adapt the code for a multi-class classification problem, the output layer must be modified to have a neuron for each class, with a softmax activation function typically used to output a probability distribution over the classes. The cost function should be changed to one that is suitable for multi-class classification, such as cross-entropy loss. Additionally, the labels for the training data need to be one-hot encoded to match the output layer's neuron structure for the respective classes.

## 2 Kernel Support Vector Machine

The Kernel Support Vector Machine (KSVM) is an advanced version of the standard SVM, a popular machine learning algorithm used for classification and regression tasks. But lets briefly discuss the key components of the KSVM:

- **Support Vector Machine (SVM)**: SVM is a supervised learning model. It is primarily used for classification but can also be applied to regression. In classification, it works by finding the hyperplane that best separates data points of different classes.
- **Hyperplane and Margin:** The hyperplane is a decision boundary separating different classes. SVM aims to find the hyperplane that maximizes the margin, which is the distance between the hyperplane and the nearest data points from each class (these points are called support vectors).
- **Kernel Trick**: The "kernel" in Kernel SVM refers to a function used to transform non-linearly separable data into a higher dimension where it is linearly separable. This is the "kernel trick," which allows SVM to effectively classify data that is not linearly separable in its original space.

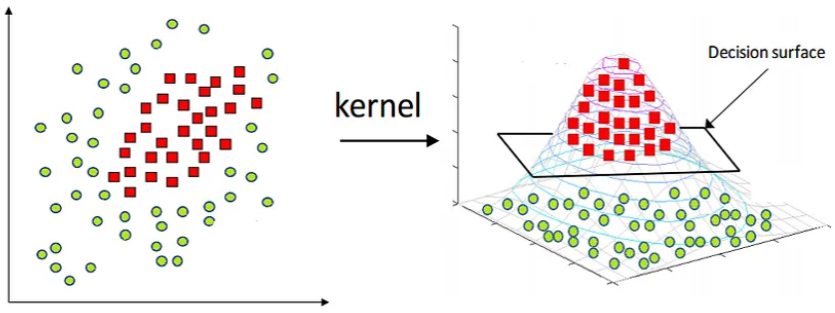
- **Types of Kernel Functions:** Common kernels include linear, polynomial, radial basis function (RBF), Gaussian, sigmoid, logistic, and Hyperbolic tangent. Each kernel function has its characteristics and is chosen based on the data distribution.
- **Advantages:** Kernel SVM is powerful for handling non-linear data distributions and works well with high-dimensional data. It is also relatively memory-efficient, as it only needs to store the support vectors during training.
- **Applications:** Kernel SVM is used in various applications like image classification, voice recognition, bioinformatics, and many areas where pattern recognition is important.

In summary, Kernel Support Vector Machines extend the capabilities of standard SVMs by employing kernel functions to enable effective classification and regression in scenarios where the data is not linearly separable. This makes them highly versatile and powerful for a wide range of machine learning tasks.

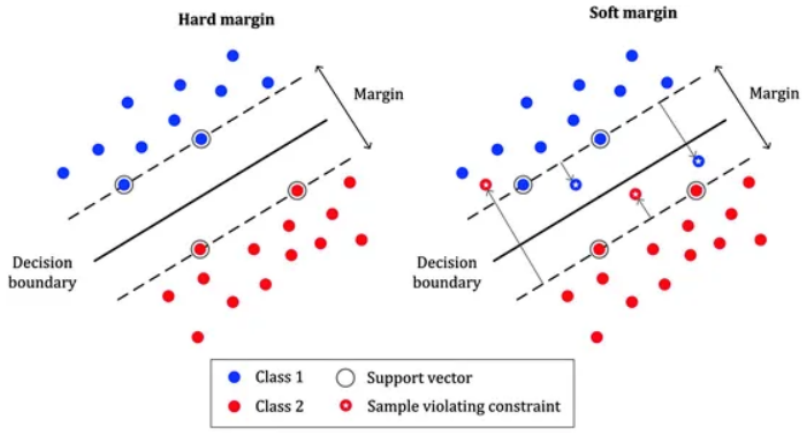
### 2.1 Kernel Support Vector Machine for Classification

In this section, we are gonna implement kernetl support vector machine for a classification problem. It is important to consider Data Preparation, Choosing a Kernel Function, Feature Scaling, Model Training, Model Evaluation to conduct a KSVM for a classification witht the lowest possible erro. The following are the Kernel functions, which they are implemented in our code.

- Linear
- Gaussian
- Hyperbolic tangent
- Logistic




(a)




(b)

Figure 6: a) Kernel function applied on a nonlinear decision boundary b) Soft-margin and Hard-maargin concepts illustrations

After the code implementation, we are gonna discuss these Kernel functions for solving a two class classification using SVM.

- **1. Gaussian Kernel:** A Gaussian Kernel Support Vector Machine (SVM) is a powerful machine learning algorithm used for binary classification and sometimes extended to multiclass problems. It is particularly effective for non-linearly separable datasets.The Gaussian Kernel is a mathematical function that measures the similarity (or distance) between data points in a high-dimensional feature space. It is defined as:

$$K(x_i, x_j) = \exp(-1/2\sigma^2 \cdot ||x_i - x_j||^2) \tag{3}$$

The performance of a Gaussian Kernel SVM depends on the choice of Kernel Parameter. The appropriate values for these parameter are typically found through techniques like cross-validation. In summary, a Gaussian Kernel SVM is a powerful tool for classification tasks, especially when dealing with non-linearly separable data. It uses a kernel function to implicitly map data into a higher-dimensional space. The Figure 7 illustrates the Gaussian kernel graph. Also, the Figure 8 and Table 3 shows the results for Gaussian KSVM with different kernel parameters. The Table

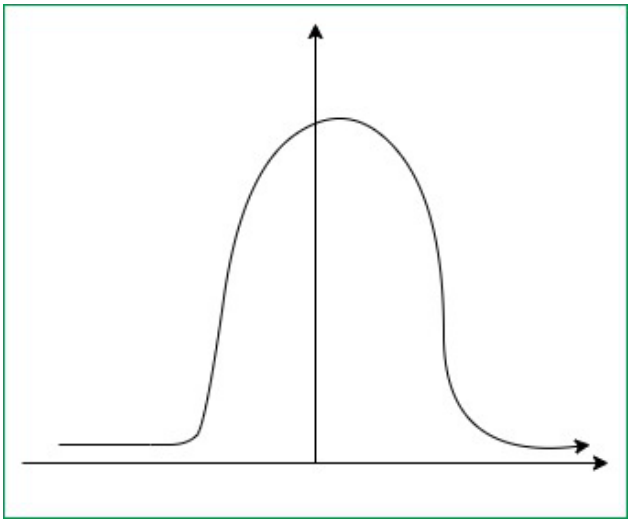


Figure 7: Gaussian kernel graph

Table 3: Gaussian KSVM accuracy for different kernel parameters

| **Kernel Parameter** | **Accuracy** % |
|---|---|
| 0.1 | 100 |
| 1 | 97 |
| 2 | 91 |
| 3 | 85.5 |
| 5 | 83.5 |

3 demonstrates the trade-off between the kernel parameter and model accuracy. Smaller values of the kernel parameter (e.g., 0.1 and 1) result in

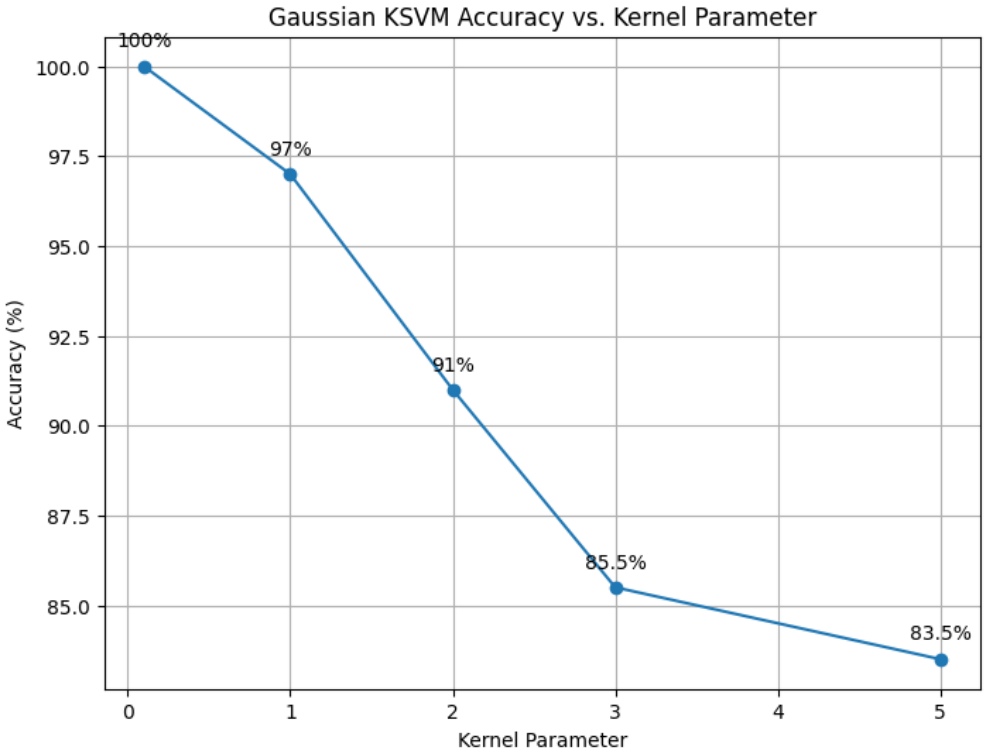


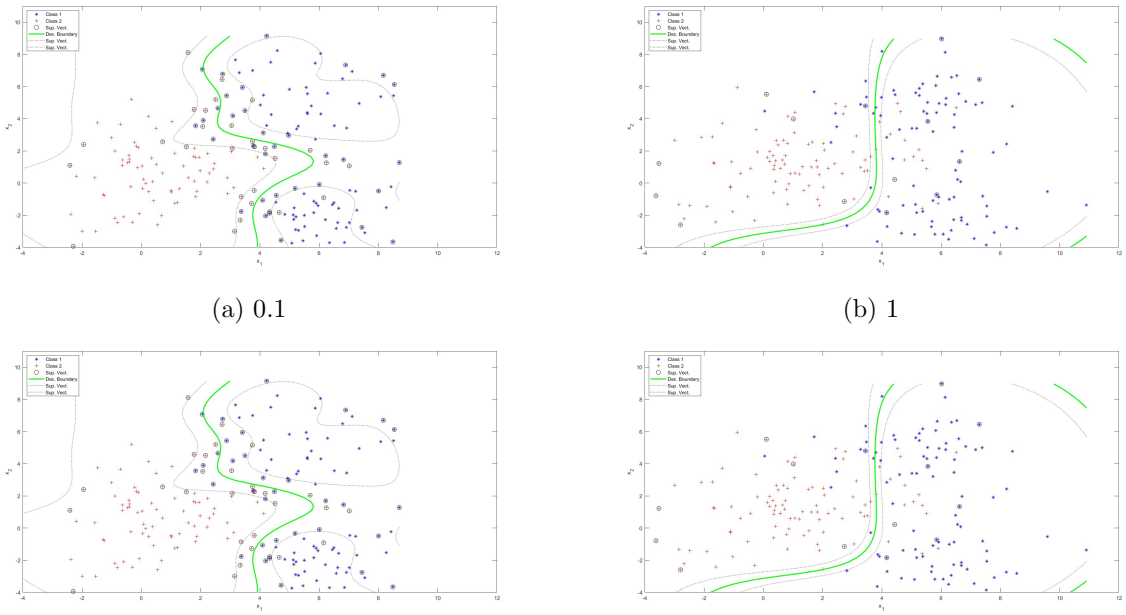

Figure 8: Gaussian KSVM accuracy for different Kernel Parameter

higher accuracy, indicating that the model can fit the training data very closely. However, this may lead to overfitting, where the model's performance on new, unseen data may not be as good.

On the other hand, larger values of the kernel parameter (e.g., 3 and 5) lead to lower accuracy, potentially indicating underfitting, where the model is too simple to capture the underlying patterns in the data. Overally, for our problem the **kernel parameter 2 is the best choice.**

Selecting an appropriate kernel parameter is a crucial hyperparameter tuning step in training a Gaussian KSVM. The choice depends on the specific dataset and the balance between bias and variance that best suits the problem.

- **2. Logistic Kernel:** The Logistic function can be also used as kernel



(a) 0.1 (b) 1

(c) 2 (d) 3

Figure 9: Gaussian Kernel SVM two-class classification for different kernel parameter a) 0.1, b) 1, c) 2, d) 3

function for SVM classification. It is suitable to handle non-linear data for two-class classification a problems. The formulation of the logistic function is as follows:

$$K(x_i, x_j) = \frac{1}{1 + \exp(\beta ||x_i - x_j||^2)} \tag{4}$$

After the code implementation, the results for logistic kernel SVM is shown in the Following Tables and Figures. Table 4, shows statistics considering the two-class classification using logisitic Kernel function for different kernel parameters. The provided table showcases a progression of model accuracy as a function of various negative kernel parameters in a Logistic Kernel SVM. Intriguingly, the accuracy increases with more negative values of the kernel parameter, a trend that is atypical since kernel parameters are generally positive. Starting at 87.5% accuracy for a kernel parameter of -0.1, the model's accuracy improves steadily to a peak of 99%

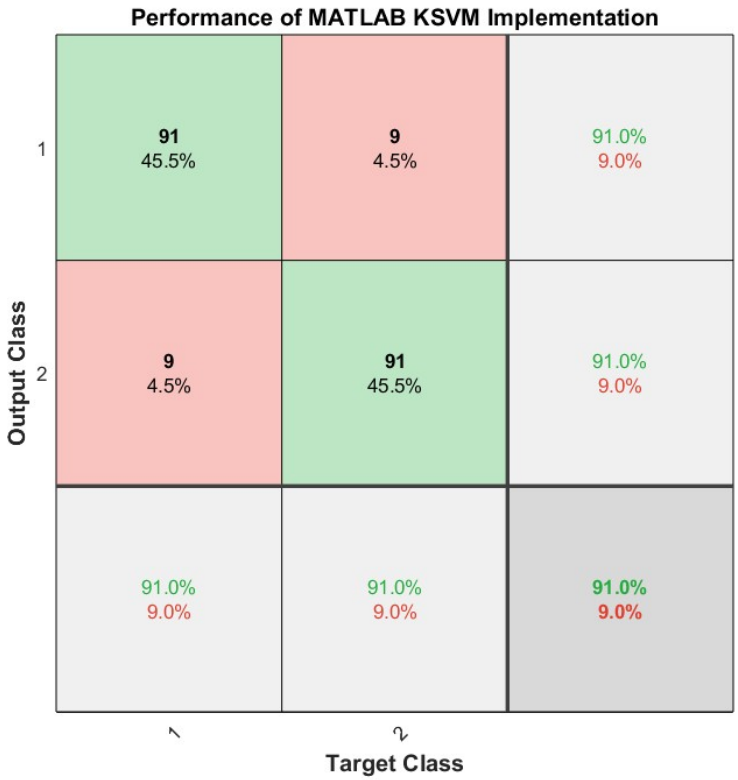



Figure 10: Confusion matrix for the best Gaussian KSVM parameter (Kernel parameter=2)

at a kernel parameter of -5. This could imply that as the magnitude of the parameter increases, the model is better able to capture the complexity of the data, possibly by adjusting the decision boundary more fittingly.

The results suggest that, for this particular application of Logistic Kernel SVM, a parameter of **-1 yields the most balanced model** with a high accuracy of 94.5%, avoiding both underfitting and overfitting. This indicates that -1 is the most suitable kernel parameter for the given dataset.

- **3. Hyperbolic Tangent:** The Hyperbolic Tangent (Tanh) Kernel is used in Support Vector Machine (SVM) classification to create non-linear de-

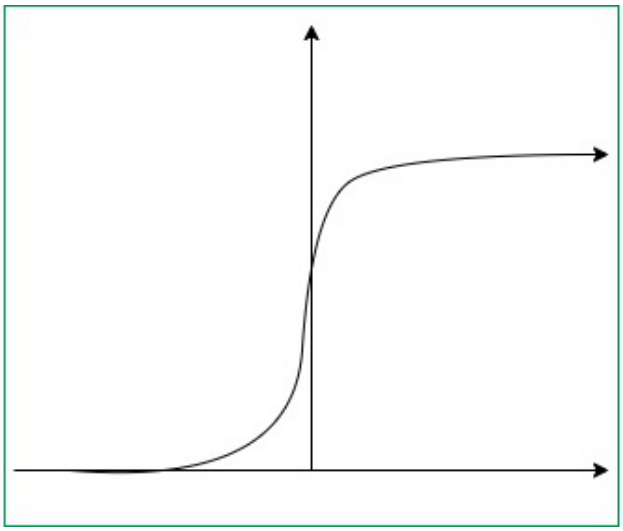


Figure 11: Logistic(Sigmoid) Kernel function Graph

Table 4: Logistic KSVM accuracy for different kernel parameters

| **Kernel Parameter** | **Accuracy** % |
|---|---|
| -0.1 | 87.5 |
| -1 | 94.5 |
| -2 | 95.5 |
| -3 | 98 |
| -5 | 99 |

cision boundaries. It's one of several kernels that allow SVMs to perform complex, non-linear classification tasks. The Tanh Kernel is mathematically expressed as:

$$K(x_i, x_j) = \tanh(\beta_1 x_i x_j' + \beta_0) \tag{5}$$

where $K(x_i, x_j)$ is the kernel function, $\beta_1$ is a scale factor, $x_i$ and $x_j$ are vectors in the input space, and $\beta_0$ is an offset.

This kernel function takes the dot product of two vectors and applies the hyperbolic tangent function to it. The Tanh Kernel is used in situations where data is not linearly separable in its original space. By applying

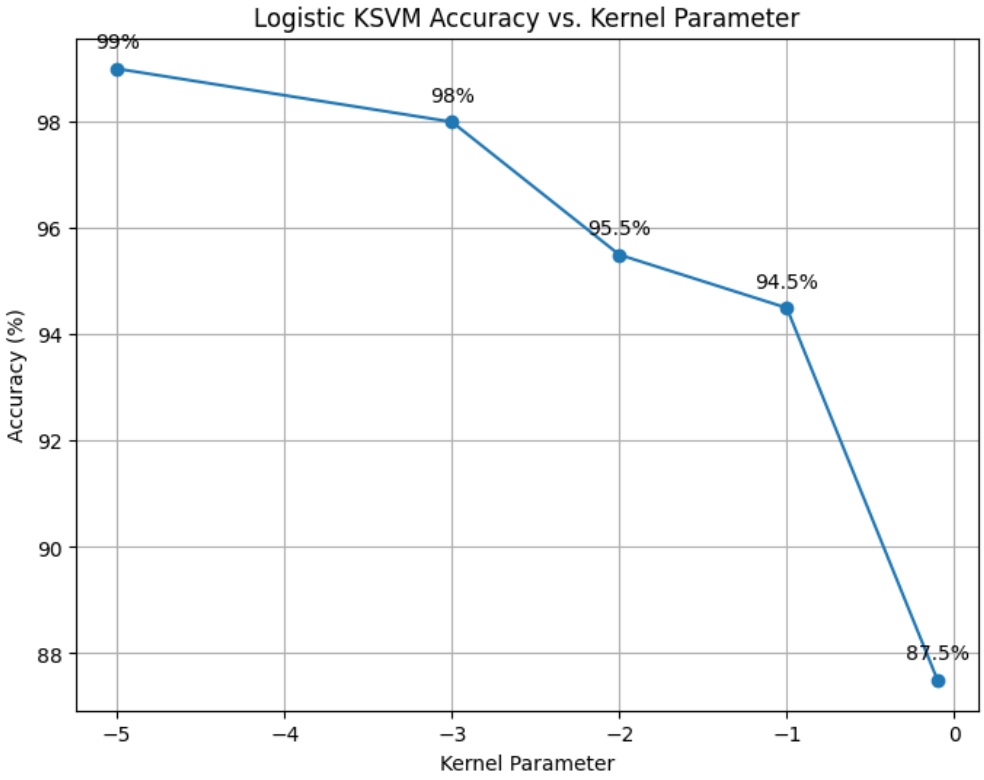
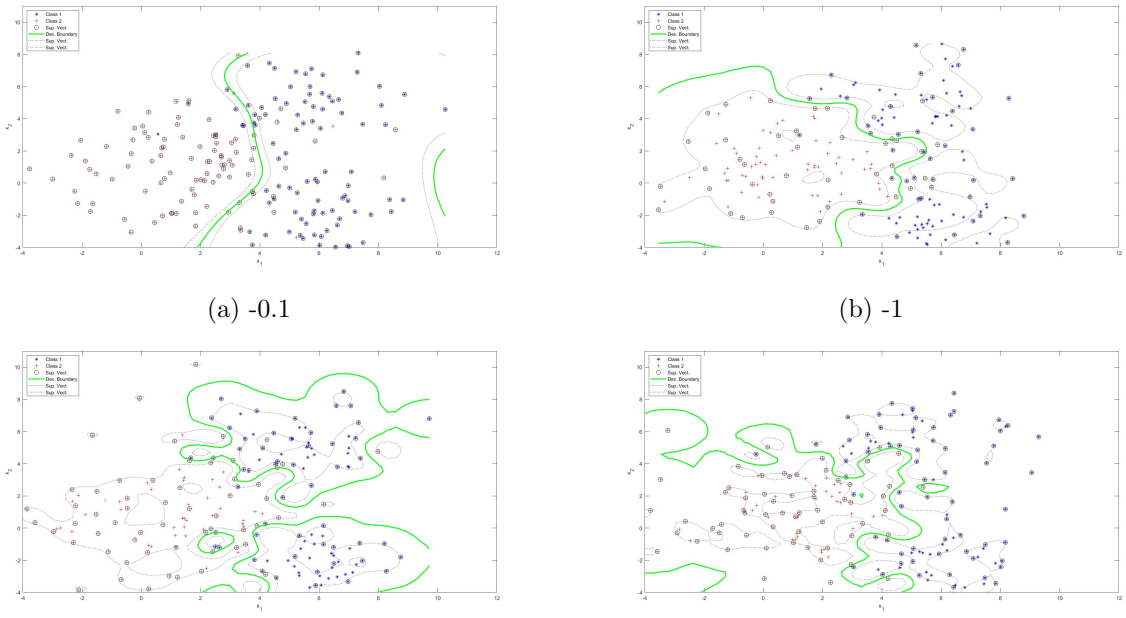



Figure 12: Logistic KSVM accuracy for different Kernel Parameter

this kernel, SVM can map the data into a higher-dimensional space where a linear separator is sufficient to distinguish between classes. The Table 3. illustrates the accuracy of the Hyperbolic tangent KSVM accuracy for different parameters. **It should be noted that after several observations the offset parameter $\beta_0$ should be defined negative, and in our problem value of -2 would be suitable**. A distinct trend

Table 5: Hyperbolic tangent KSVM accuracy for different kernel parameters

| **Kernel Parameter** | **Accuracy %** |
|---|---|
| 0.01 | 92.5 |
| 0.05 | 90 |
| 0.1 | 85.5 |
| 0.5 | 75.7 |



(a) -0.1 (b) -1

(c) -2 (d) -3

Figure 13: Logistic Kernel SVM two-class classification for different kernel parameter a) -0.1, b) -1, c) -2, d) -3

is observable in the data: as the kernel parameter increases, there is a consistent decrease in model accuracy. The highest accuracy, 92.5%, is achieved with the smallest kernel parameter of 0.01. This suggests that a finer, less aggressive transformation of the input data (which a lower kernel parameter value implies) is more effective for this particular dataset. As it is shown in Figure 15 this trend indicates that the dataset might have a complex but not highly nonlinear structure that requires a subtle transformation for optimal classification. Larger kernel parameters generally increase the complexity of the decision boundary, but here it seems to degrade the model's performance. At a kernel parameter of 0.5, the accuracy drops significantly to 75.7%, which could imply that the decision boundary is becoming overly complex, potentially overfitting the training data or failing to capture the essential structure of the data. Eventually,

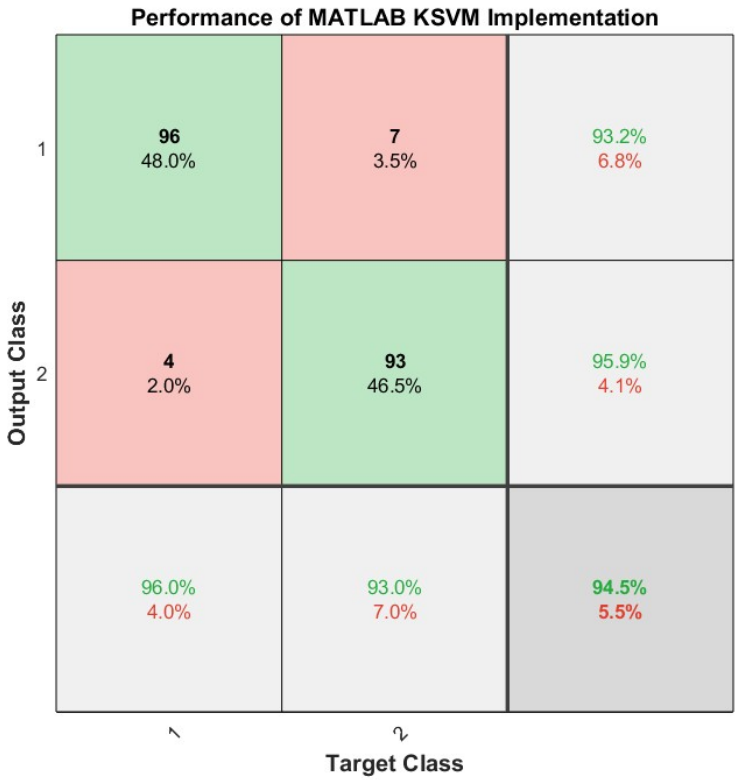



Figure 14: Confusion matrix for the best Logistic KSVM parameter (Kernel parameter=-1)

we can assert that the best parameter of the Hyperbolic tangent KSVM would be **0.01 to avoid underftting and overfitting and have an acceptable accuracy.**

## 2.2 Different KSVM methods for Classification

In this section we are gonna compare different KSVM methods for 3 different random problems. Earlier we discussed about Gaussian, Logistic, and Hyper tangent KSVM implementation using our code; however, this matter is possible using **MATLAB "fitcsvm"** function. The fitcsvm function in MATLAB is designed for training Support Vector Machine (SVM) classifiers for both one-class

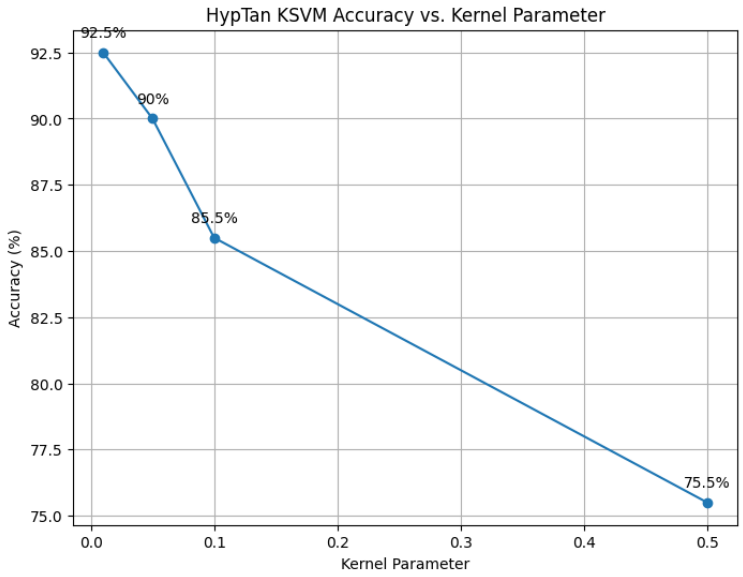
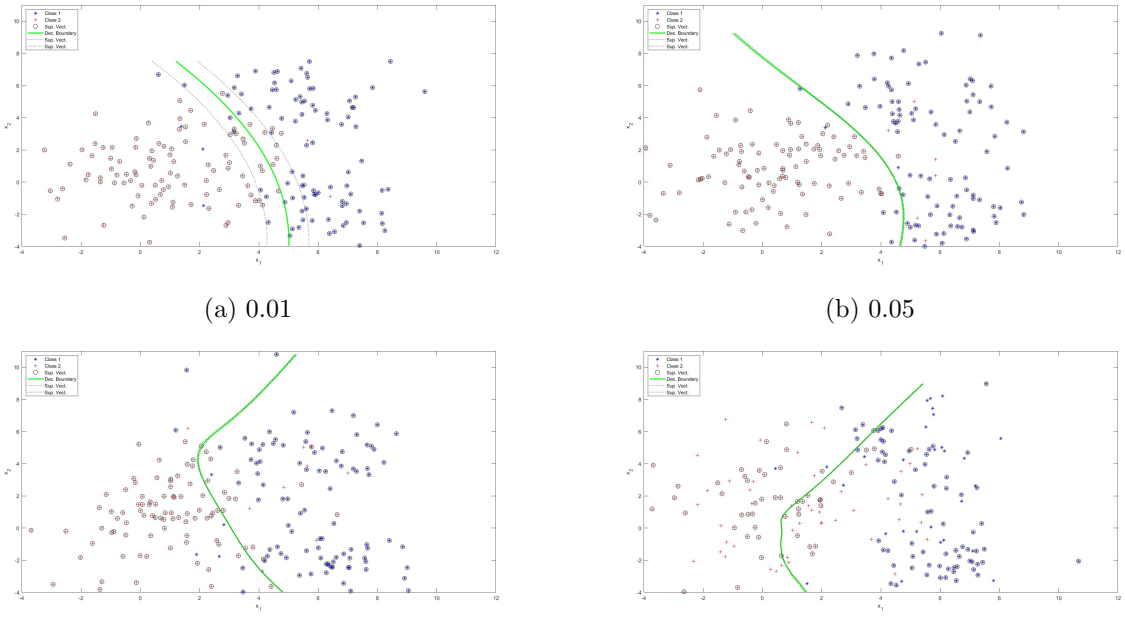



Figure 15: HypTan KSVM accuracy for different Kernel Parameters



(a) 0.01

(b) 0.05

(c) 0.1

(d) 0.5

Figure 16: HypTan Kernel SVM two-class classification for different kernel parameter a) 0.01, b) 0.05, c) 0.1, d) 0.5

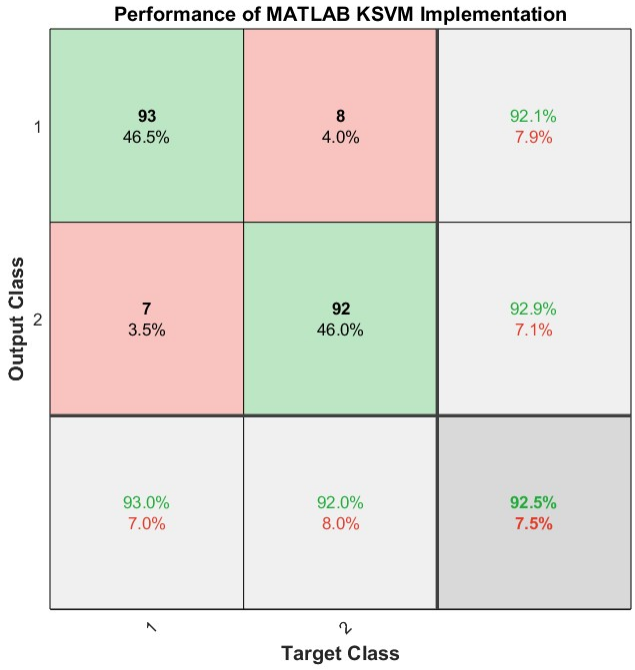



Figure 17: Confusion matrix for the best HypTan KSVM parameter (Kernel parameter=0.01)

and binary classification tasks. The fitcsvm function trains or cross-validates SVM models for one-class and two-class (binary) classification on datasets with low to moderate dimensions. It supports various kernel functions and optimization algorithms like Sequential Minimal Optimization (SMO), Iterative Single Data Algorithm (ISDA), or L1 soft-margin minimization via quadratic programming.

Table 4 illustates the Kernel SVM methods accuracy including Gaussian, Logistic, Hyperbolic Tangent, MATLAB Gaussian, and MATLAB Polynomial. Based on their accuracy, the table reveals the following insights:

- Logistic Kernel SVM shows the highest accuracy across all three problems (95.5%, 94%, 96.5%), indicating superior performance in this context. Its consistently high accuracy suggests strong generalizability and robustness.

Table 6: Gaussian KSVM accuracy for different kernel parameters

| **Random Problem Accuracy %** | **1** | **2** | **3** |
|---|---|---|---|
| Gaussian (kernel Parameter= 2) | 94.5 | 91 | 95.5 |
| Logistic (kernel Parameter= -1) | 95.5 | 94 | 96.5 |
| Hyperbolic Tangent (kernel Parameter= 0.01) | 88.5 | 86.5 | 90 |
| MATLAB Gaussian KSVM | 95 | 95 | 95.5 |
| MATLAB Polynomial KSVM | 67.5 | 58 | 71 |

- Gaussian Kernel SVM also demonstrates high accuracy, though slightly lower than the Logistic Kernel (94.5%, 91%, 95.5%). This suggests effective performance, likely due to the Gaussian kernel's ability to handle non-linear data patterns.

- Hyperbolic Tangent Kernel SVM has lower accuracy compared to the others (88.5%, 86.5%, 90%). This could indicate that this kernel is less effective for the datasets tested, possibly due to its sensitivity to the data's distribution.

- **The reasons for any performance difference between different Kernel functions for our implementation of KSVM would be due to the several reasons. First, Gaussian and Logistic kernels might be better suited to the structure and distribution of the data in your specific problems. Gaussian kernels are particularly adept at handling cases where the boundary between classes is not linear and can capture complex relationships. Second, The Hyperbolic Tangent kernel often requires data to be centered around zero. If the data isn't preprocessed accordingly, this kernel may not perform optimally. Also, Gaussian kernels, in particular, are known for their good balance between bias and variance, leading to better generalization on unseen data. The Logistic kernel might also offer a similar balance in your specific scenarios**

- MATLAB Implementations: The MATLAB Gaussian KSVM shows very consistent performance, indicating stability. In contrast, the MATLAB Polynomial KSVM has significantly lower accuracy, suggesting it may not be as well-suited for these particular datasets or problems.
- In summary, the Logistic and Gaussian kernel SVMs outperform the Hyperbolic Tangent and Polynomial kernel SVMs in terms of accuracy, with the Logistic kernel SVM showing the best overall performance.

Eventually the following plots show better visualization for the performance of different Kernel functions for the specified problem.

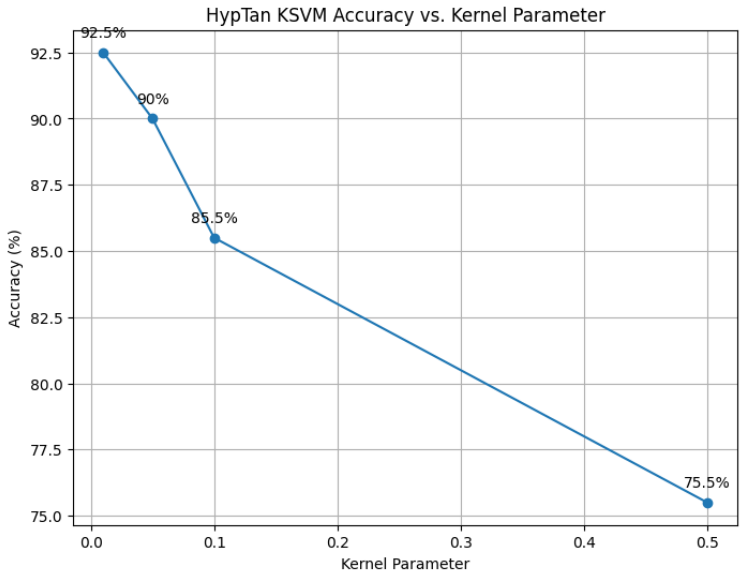
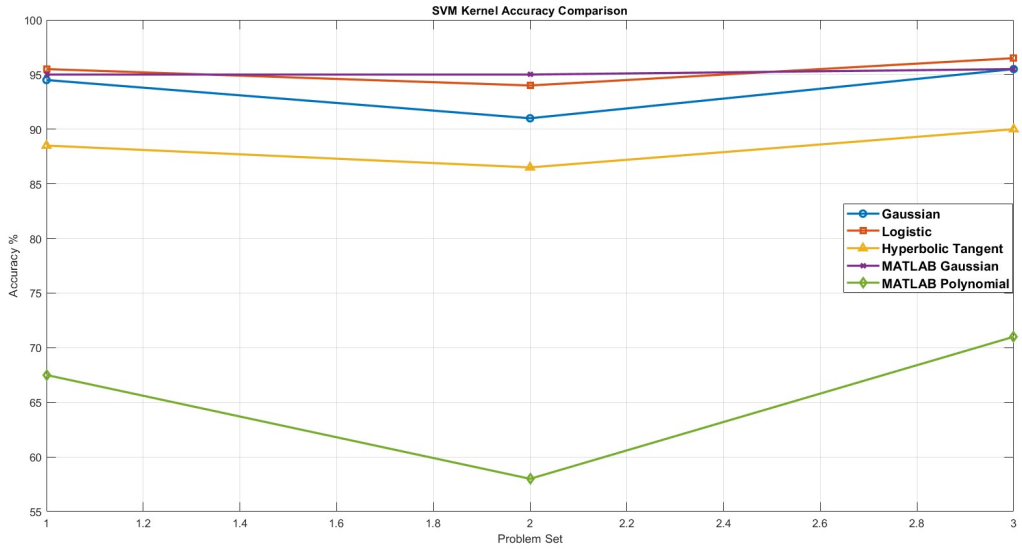
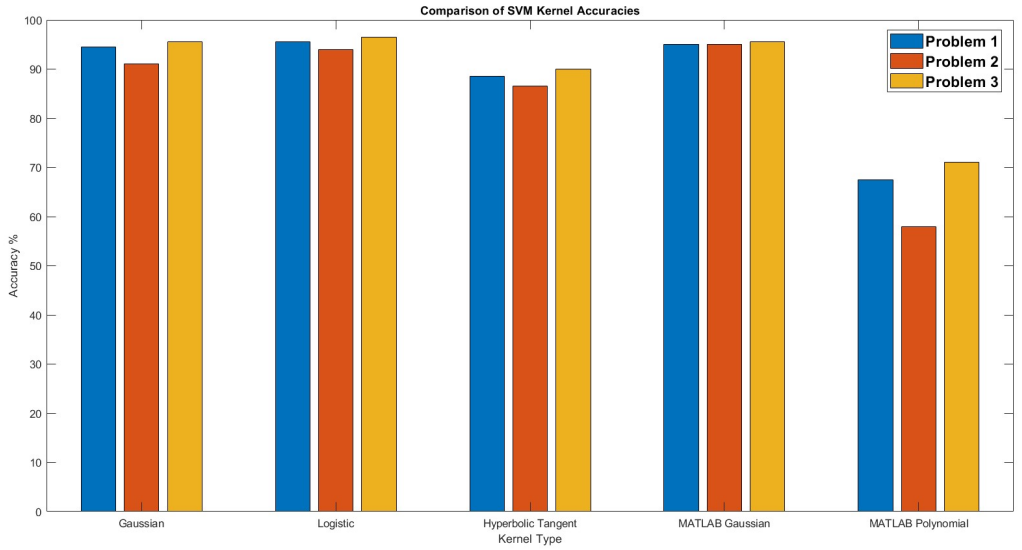



Figure 18: HypTan KSVM accuracy for different Kernel Parameters




(a)




(b)

Figure 19: Comparison between different Kernel functions performance for solving two-class classification problem using SVM