**CptS 570 Machine Learning**

# Homework - 2

Submitted by
**Athul Jose P**
11867566

School of Electrical Engineering and Computer Science
Washington State University

# Contents

# List of Tables



# List of Figures

# 1 Q1

Expanding the Euclidean Distances

$$(x - C^+) \cdot (x - C^+) = (x - C^-) \cdot (x - C^-)$$

$$x \cdot x - 2x \cdot C^+ + C^+ \cdot C^+ = x \cdot x - 2x \cdot C^- + C^- \cdot C^-$$

Since $x \cdot x$ appears on both sides, it cancels out, leaving:

$$-2x \cdot C^+ + C^+ \cdot C^+ = -2x \cdot C^- + C^- \cdot C^-$$

Rearranging:

$$2x \cdot (C^- - C^+) = C^+ \cdot C^+ - C^- \cdot C^-$$

Dividing both sides by 2:

$$x \cdot (C^- - C^+) = \frac{1}{2}\left(C^+ \cdot C^+ - C^- \cdot C^-\right) \tag{1}$$

Equation (1) is in the form:

$$w \cdot x + b = 0$$

where:

$$w = C^- - C^+, \quad b = \frac{1}{2}\left(C^+ \cdot C^+ - C^- \cdot C^-\right)$$

Thus, the decision boundary of the CLOSE classifier is a linear hyperplane.

In the CLOSE classifier, the decision is based on the distance of training examples from the centers $C^+$ and $C^-$. This suggests that the centers themselves serve as support vectors.

The weight vector $w$ can be written as a linear combination of the training examples:

$$w = \sum_{i=1}^{n^+ + n^-} \alpha_i \cdot y_i \cdot x_i$$

Since $w = C^- - C^+$, the dual weights $\alpha_i$ should focus only on the centers $C^+$ and $C^-$. Specifically:

$$\alpha = C^+, \quad \alpha = C^-$$

This implies that only the centers of the positive and negative classes contribute non-zero dual weights, while the rest of the training examples have zero dual weights.

Since the centers $C^+$ and $C^-$ are the only examples that determine the decision boundary, they are the support vectors of the CLOSE classifier.

# 2 Q2

The RBF kernel is defined as:

$$K(x_i, x_j) = \exp\left(-\frac{1}{2}\|x_i - x_j\|^2\right).$$

The squared norm can be expanded as:

$$K(x_i, x_j) = \exp\left(-\frac{1}{2}\left(\|x_i\|^2 - 2x_i \cdot x_j + \|x_j\|^2\right)\right).$$

This can be rewritten as:

$$K(x_i, x_j) = \exp\left(-\frac{1}{2}\left(\|x_i\|^2 + \|x_j\|^2\right) + x_i \cdot x_j\right).$$

Using the exponential property $\exp(a + b) = \exp(a)\exp(b)$,

$$K(x_i, x_j) = \exp\left(-\frac{1}{2}\left(\|x_i\|^2 + \|x_j\|^2\right)\right)\exp\left(x_i \cdot x_j\right).$$

Now, using the Taylor series expansion for $\exp(x_i \cdot x_j)$:

$$\exp\left(x_i \cdot x_j\right) = 1 + x_i \cdot x_j + \frac{(x_i \cdot x_j)^2}{2!} + \ldots$$

Since this expansion has infinitely many terms, the corresponding feature mapping $\phi(x)$ must also have infinite dimensions.

The squared Euclidean distance between two points in the transformed space is given by:

$$\|\phi(x_i) - \phi(x_j)\|^2 = \|\phi(x_i)\|^2 + \|\phi(x_j)\|^2 - 2\phi(x_i) \cdot \phi(x_j).$$

Using the properties of the RBF kernel:

$$\|\phi(x_i)\|^2 = K(x_i, x_i) = \exp\left(-\frac{1}{2}\|x_i - x_i\|^2\right) = 1,$$

and similarly:

$$\|\phi(x_j)\|^2 = K(x_j, x_j) = 1.$$

Also:

$$\phi(x_i) \cdot \phi(x_j) = K(x_i, x_j) = \exp\left(-\frac{1}{2}\|x_i - x_j\|^2\right).$$

Thus:

$$\|\phi(x_i) - \phi(x_j)\|^2 = 2 - 2\exp\left(-\frac{1}{2}\|x_i - x_j\|^2\right).$$

Bounding the Squared Distance
Since $\exp\left(-\frac{1}{2}\|x_i - x_j\|^2\right)$ takes values between 0 and 1 (inclusive),

$$\|\phi(x_i) - \phi(x_j)\|^2 = 2\left(1 - \exp\left(-\frac{1}{2}\|x_i - x_j\|^2\right)\right).$$

The maximum value of this expression occurs when $\exp\left(-\frac{1}{2}\|x_i - x_j\|^2\right) = 0$, giving:

$$\|\phi(x_i) - \phi(x_j)\|^2 \leq 2.$$

The squared Euclidean distance between any two transformed points in the feature space is bounded by 2.

# 3 Q3

The decision function for an SVM is given by:

$$f(x; \alpha, b) = \sum_{i \in SV} y_i \alpha_i K(x_i, x) + b.$$

Using the RBF kernel:

$$K(x_i, x) = \exp\left(-\frac{1}{2}\|x_i - x\|^2\right).$$

When the test point $x_{\text{far}}$ is far away from any training instance $x_i$, the distance $\|x_i - x_{\text{far}}\|$ becomes very large. As a result:

$$K(x_i, x_{\text{far}}) = \exp\left(-\frac{1}{2}\|x_i - x_{\text{far}}\|^2\right) \approx \exp(-\infty) = 0.$$

Thus, for all support vectors $x_i \in SV$, the contribution of the RBF kernel becomes negligible:

$$K(x_i, x_{\text{far}}) \approx 0.$$

Since $x_{\text{far}}$ is far from all training instances, the dual weights $\alpha_i$ for these terms effectively become zero:

$$\alpha_i K(x_i, x_{\text{far}}) \approx 0 \quad \text{for all} \quad i \in SV.$$

Thus, the decision function simplifies to:

$$f(x_{\text{far}}; \alpha, b) = \sum_{i \in SV} y_i \alpha_i K(x_i, x_{\text{far}}) + b \approx b.$$

Since the contribution from the support vectors vanishes as the test point $x_{\text{far}}$ moves far from the training data, the decision function becomes:

$$f(x_{\text{far}}; \alpha, b) \approx b.$$

This behavior demonstrates that the output of the decision function for distant points depends solely on the bias term $b$.

# 4 Q4

The given function is:

$$K(x_i, x_j) = -\langle x_i, x_j \rangle,$$

where $\langle x_i, x_j \rangle$ denotes the standard inner product between vectors $x_i$ and $x_j$.

The given function is symmetric, as:

$$K(x_i, x_j) = -\langle x_i, x_j \rangle = -\langle x_j, x_i \rangle = K(x_j, x_i).$$

Thus, the symmetry condition is satisfied.

The quadratic form of the kernel is given by:

$$z^T K z = \sum_{i=1}^{n}\sum_{j=1}^{n} z_i z_j K(x_i, x_j) = \sum_{i=1}^{n}\sum_{j=1}^{n} z_i z_j \left(-\langle x_i, x_j\rangle\right).$$

Simplifying:

$$z^T K z = -\sum_{i=1}^{n}\sum_{j=1}^{n} z_i z_j \langle x_i, x_j\rangle.$$

Let $x_i = Ax$ and $x_j = Ay$ for any vectors $x$ and $y$, where $A$ is a symmetric matrix. Then:

$$\langle x_i, x_j\rangle = (Ax)^T(Ay) = x^T A^T A y.$$

Thus:

$$K(x_i, x_j) = -\langle x_i, x_j\rangle = -x^T A^T A y.$$

Since $A^T A$ is positive semi-definite by definition, multiplying it by $-1$ makes it negative semi-definite.

Since the quadratic form is negative for non-zero vectors:

$$z^T K z = -\left\|\sum_{i=1}^{n} z_i x_i\right\|^2 \leq 0,$$

the kernel function $K(x_i, x_j) = -\langle x_i, x_j\rangle$ is not positive semi-definite.

The function $K(x_i, x_j) = -\langle x_i, x_j\rangle$ is not a valid kernel because it does not satisfy the positive semi-definiteness condition, even though it is symmetric. Therefore,

$$K(x_i, x_j) = -\langle x_i, x_j\rangle \quad \text{is not a valid kernel.}$$

# 5 Q5

The standard soft-margin SVM formulation minimizes the following objective function:

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i,$$

subject to:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i \in \{1, \ldots, n\}.$$

Here:

- $w$ and $b$ define the decision boundary.
- $\xi_i$ are the slack variables, allowing for misclassifications or margin violations.
- $C$ is the penalty parameter controlling the trade-off between margin size and slack variables.

Given the costs $C_+$ and $C_-$ for misclassifying positive and negative examples, the soft-margin SVM formulation modified as follows:

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C_+ \sum_{i \in \mathcal{N}^+} \xi_i + C_- \sum_{i \in \mathcal{N}^-} \xi_i,$$

subject to:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i \in \{1, \ldots, n\}.$$

Here:

- $\mathcal{N}^+$ is the set of positive examples (with $y_i = +1$).
- $\mathcal{N}^-$ is the set of negative examples (with $y_i = -1$).

The modified objective function introduces class-specific penalties:

- $C_+$ determines the penalty for misclassifying positive examples.
- $C_-$ determines the penalty for misclassifying negative examples.

This allows the SVM to account for situations where different types of misclassifications have different consequences. For example, if false positives (misclassified negative examples) are more costly than false negatives, setting $C_- > C_+$ ensures that the model focuses more on reducing false positives.

The modified SVM formulation becomes:

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C_+ \sum_{i=1}^{N^+} \xi_i + C_- \sum_{i=1}^{N^-} \xi_i,$$

subject to:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i \in \{1, \ldots, n\},$$

where $N^+$ and $N^-$ are the number of positive and negative examples, respectively.

In the above formulation, the strength of the penalty on misclassification of positive and negative examples is determined by the values of $C_+$ and $C_-$. This adjustment ensures that the SVM model can prioritize minimizing the costlier type of misclassification according to the given problem context. By incorporating class-specific penalties $C_+$ and $C_-$, the modified soft-margin SVM can effectively handle cases with imbalanced data or differing costs for misclassifications. This formulation ensures the SVM learns a decision boundary that reflects the relative importance of each type of misclassification.

# 6 Q6

A large dataset of $n$ training examples is given:

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n),$$

where $x_i$ is the input example and $y_i \in \{+1, -1\}$ is the class label. Due to the size of the dataset (millions of examples), traditional SVM training algorithms are computationally expensive. To address this scalability issue, use a Coarse-to-Fine framework. This approach iteratively clusters the data, refines the clusters, and performs SVM training until convergence. The following solution is inspired by the reference [1].

### 6.1 Q 6.a: Tree-based Clustering and SVM Training

To solve the scalability problem, adopt a tree-based clustering approach.

Steps

1. Construct Positive and Negative Trees: Create two trees:
   - $T^+$ from the $k_+$ positive clusters.
   - $T^-$ from the $k_-$ negative clusters.
2. SVM Training on Root Node Centroids: Train the SVM using the centroids of the root nodes of $T^+$ and $T^-$. The centroid $C$ of a cluster is computed as:

$$C = \frac{1}{N} \sum_{i=1}^{N} x_i,$$

   where $\{x_i\}$ are the data points in the cluster, and $N$ is the number of points in the cluster.
3. Handling Sparse Root Nodes: If the root node contains too few entries, train the SVM on the centroids from the second level of the trees.
4. Declustering Near the Boundary: Identify clusters near the decision boundary and decluster them into the next level. Accumulation Rule: Children entries declustered from a parent node are added to the training set along with the non-declustered parent entries.
5. Iterative Training: Construct another SVM using the centroids of the accumulated entries. Repeat step 4 until no further accumulation occurs.

### 6.2 Q 6.b: Cluster Refinement Based on Low-margin Clusters

To refine clusters, identify the clusters that are close to the boundary and require finer granularity.

Let $D_s$ be the distance from the boundary to the centroid of a support cluster, and $D_i$ be the distance from the boundary to the centroid of a cluster $E_i$.

A cluster $E_i$ is considered a low-margin cluster if:

$$D_i - R_i < D_s,$$

where $R_i$ is the radius of the cluster $E_i$, computed as:

$$R_i = \left( \frac{1}{N} \sum_{i=1}^{N} \|x_i - C\|^2 \right)^{\frac{1}{2}},$$

where $C$ is the centroid of the cluster $E_i$, and $\{x_i\}$ are the data points in the cluster.

Clusters satisfying this condition are likely to contain support vectors and should be declustered further for finer-level training.

### 6.3 Q 6.c: Stopping Criteria

The algorithm terminates when no clusters are left for further declustering. This means that no additional clusters can be identified as low-margin clusters requiring refinement.
Algorithm: Coarse-to-Fine SVM Training

1. Initialize Clusters: Construct $T^+$ and $T^-$ trees from the positive and negative clusters, respectively.
2. Train SVM on Root Centroids: Train an SVM using the centroids of the root nodes.
3. Refine Clusters:
   - Identify low-margin clusters near the boundary.
   - Decluster these clusters and accumulate children entries into the training set.
4. Iterative Training: Train a new SVM on the refined clusters. Repeat the process until no further accumulation occurs.
5. Check Stopping Criteria: Stop when no further clusters are identified for declustering.

This approach may fail under the following conditions:

- Highly Nonlinear Boundaries: If the true decision boundary is highly complex, clustering might not capture the full data distribution, leading to suboptimal performance.
- Imbalanced Data: If the dataset is highly skewed, the clustering and declustering steps may not yield meaningful results.
- Poor Initial Clustering: If the initial clusters are not representative, the iterative process may not converge effectively.

## 7 Q7

Given a large dataset with $n$ training examples:

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n),$$

where $x_i$ is the input example, and $y_i \in \{+1, -1\}$ is the class label, standard online kernelized Perceptron algorithms may not scale due to the unbounded growth of support vectors (SVs). In this document, address the challenge of using a limited number of support vectors for both prediction and training in a real-time environment.

### 7.1 Q 7.a: Selecting B Support Vectors for Prediction

After training the kernelized Perceptron, the set of support vectors is denoted as $SV = \{(x_i, y_i)\}_{i=1}^{m}$. Since Tom requires real-time prediction with at most $B$ kernel evaluations per classification, need to select a subset of $B$ support vectors from the complete set $SV$. Algorithm for Selecting B Support Vectors The goal is to select the most informative support vectors to retain the predictive accuracy while satisfying the constraint of $B$ evaluations. The following algorithm describes the process:

1. Compute Margin Contribution: For each support vector $(x_i, y_i) \in SV$, calculate its margin contribution:
$$M_i = y_i \sum_{j=1}^{m} \alpha_j y_j K(x_j, x_i),$$
where $\alpha_j$ is the weight for each support vector from the original Perceptron training.
2. Select Top B Support Vectors: Sort the support vectors by the absolute value of their margin contributions $|M_i|$ in descending order.
3. Choose the Top B Vectors: Select the top $B$ support vectors with the largest contributions.
4. Use for Prediction: During prediction, use only the selected $B$ support vectors to evaluate the classification decision:
$$f(x) = \text{sign}\left(\sum_{i=1}^{B} \alpha_i y_i K(x_i, x)\right).$$

Key motivation for the algorithm

- Margin Contribution: The support vectors with the largest contributions to the margin are likely to be the most influential in making accurate predictions.
- Efficiency: This method ensures that the most critical support vectors are retained while satisfying the $B$-evaluation constraint.
- Simplicity: Sorting by margin contribution is computationally feasible and provides a straightforward way to select the top support vectors.

### 7.2 Q 7.b: Training with B Support Vectors

To address the issue of unbounded support vectors during training, modify the standard kernelized Perceptron training algorithm to allow at most $B$ support vectors at any time.
Modified Kernelized Perceptron Training Algorithm

1. Initialize: Set the support vector set $SV = \emptyset$ and iteration count $t = 1$.
2. For each training example $(x_t, y_t)$:
   - Compute the prediction:
$$\hat{y}_t = \text{sign}\left(\sum_{(x_i, y_i) \in SV} \alpha_i y_i K(x_i, x_t)\right).$$
   - If $\hat{y}_t \neq y_t$ (misclassified example):
     - Add $(x_t, y_t)$ to $SV$ with weight $\alpha_t = 1$.
     - If $|SV| > B$:
       * Remove the Least Informative Support Vector:
$$(x_r, y_r) = \arg \min_{(x_i, y_i) \in SV} |\alpha_i|.$$

* Remove $(x_r, y_r)$ from $SV$.

3. Increment $t$ and repeat until all training examples have been processed.

Key motivation for the algorithm

- Fixed Size Support Vector Set: By ensuring that the support vector set size never exceeds $B$, control the memory and computational cost of the algorithm.
- Removing the Least Informative Vector: By removing the support vector with the smallest weight $\alpha_i$, minimize the impact on classification performance.
- Online Learning: The algorithm processes each example in an online fashion, making it suitable for large-scale datasets.

# 8 P1: Support Vector Machine

## 8.1 a

Figure 1 shows the training, test, and validation accuracies for different values of the hyperparameter $C$ using LinearSVC from 'scikit-learn'. The model was trained with a maximum iteration limit of 30,000 to ensure convergence. The x-axis represents the values of $C$ ranging from $10^{-4}$ to $10^{4}$, while the y-axis displays the corresponding accuracies. As seen in the plot, training accuracy increases steadily with larger $C$, but both test and validation accuracies start to decline for higher values, indicating potential overfitting.

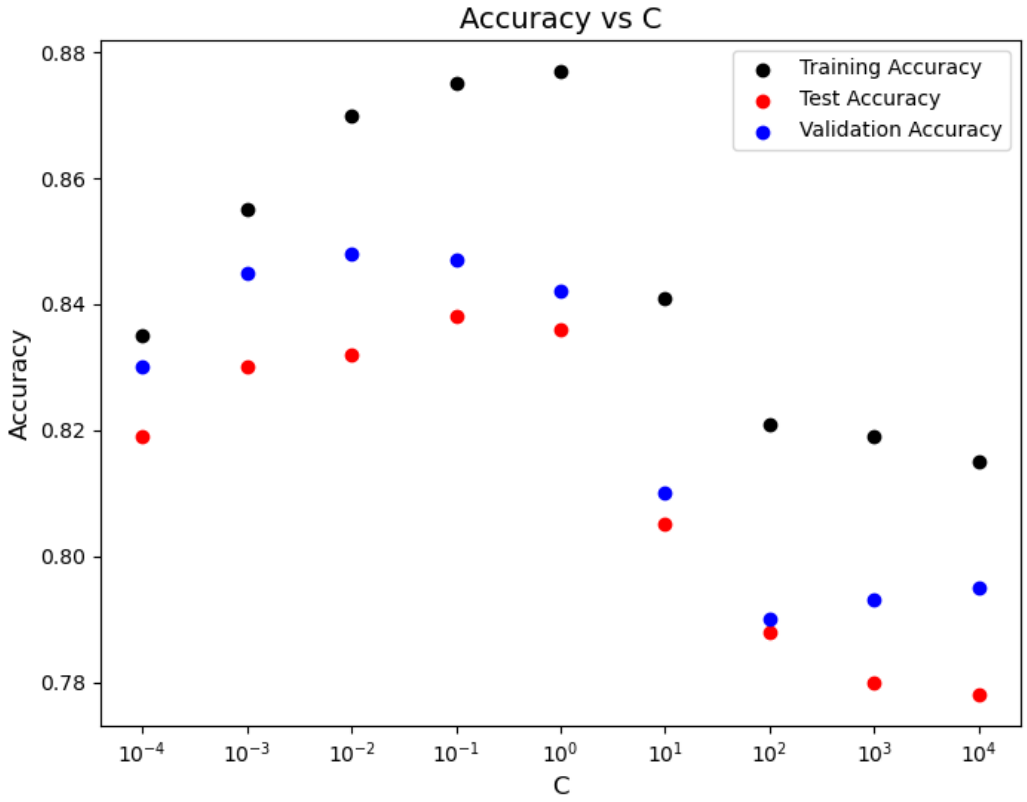


Figure 1: Accuracy vs C parameter

The highest validation accuracy is achieved at $C = 10^{-2}$, making it the recommended value for optimal performance. Lower or higher values of $C$ lead to poorer generalization, as reflected by decreased validation accuracy. While LinearSVC offers good performance, it does not provide the number of support vectors, unlike standard SVM implementations.

## 8.2 b

In Figure 1, the validation accuracy reaches its peak at $C = 10^{-2}$, indicating that this value offers the best generalization on unseen data. As a result, $C = 10^{-2}$ was selected as the optimal hyperparameter for the model. This choice ensures a good balance between underfitting and overfitting, maximizing performance on the validation set.

After selecting $C = 10^{-2}$, the model was evaluated on a combined set of training and validation examples, achieving a test accuracy of 0.838. This combined evaluation provides a clearer picture of how well the model performs across a broader range of examples. The performance details are further summarized using a confusion matrix, which highlights the breakdown of true positives, false positives, true negatives, and false negatives.

Table 1: Confusion Matrix

| **Actual Class** | **Predicted Class** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 819 | 3 | 14 | 52 | 7 | 2 | 88 | 3 | 11 | 1 |
| 1 | 3 | 955 | 4 | 27 | 5 | 0 | 3 | 1 | 2 | 0 |
| 2 | 26 | 4 | 726 | 13 | 138 | 0 | 78 | 2 | 13 | 0 |
| 3 | 29 | 18 | 17 | 861 | 33 | 2 | 31 | 3 | 5 | 1 |
| 4 | 1 | 2 | 110 | 40 | 775 | 2 | 60 | 0 | 10 | 0 |
| 5 | 2 | 1 | 0 | 0 | 0 | 918 | 0 | 48 | 11 | 20 |
| 6 | 154 | 5 | 140 | 49 | 113 | 0 | 509 | 0 | 28 | 2 |
| 7 | 0 | 0 | 0 | 0 | 0 | 32 | 0 | 937 | 1 | 30 |
| 8 | 7 | 1 | 7 | 11 | 3 | 8 | 18 | 5 | 940 | 0 |
| 9 | 0 | 0 | 0 | 2 | 0 | 14 | 1 | 42 | 1 | 940 |

### 8.3 c

Figure 2 illustrates the variation in training, validation, and test accuracies with respect to the degree of the polynomial kernel. The accuracy for the linear kernel is not included, as the training for the linear kernel was performed using LinearSVC (as described in Problem 1a), whereas the polynomial kernel training was conducted using SVC from 'scikit-learn'. From the figure, it is evident that the accuracies decline as the degree of the polynomial increases for the given dataset.

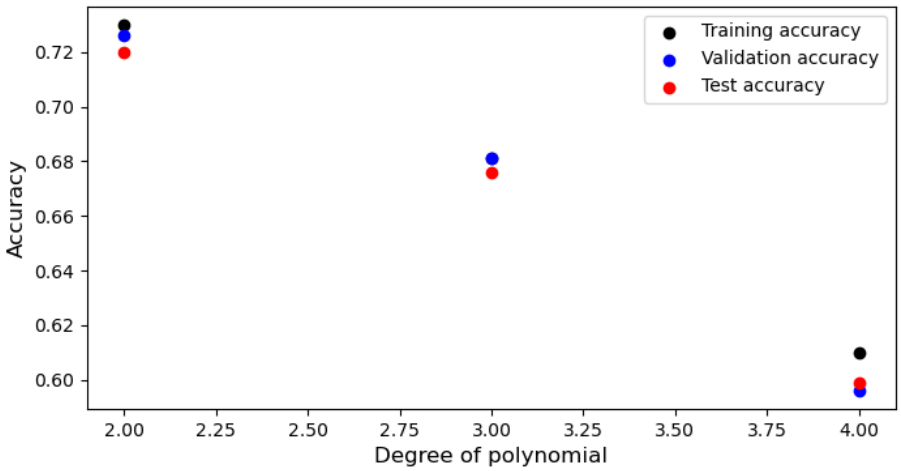


Figure 2: Accuracy vs degree of polynomial

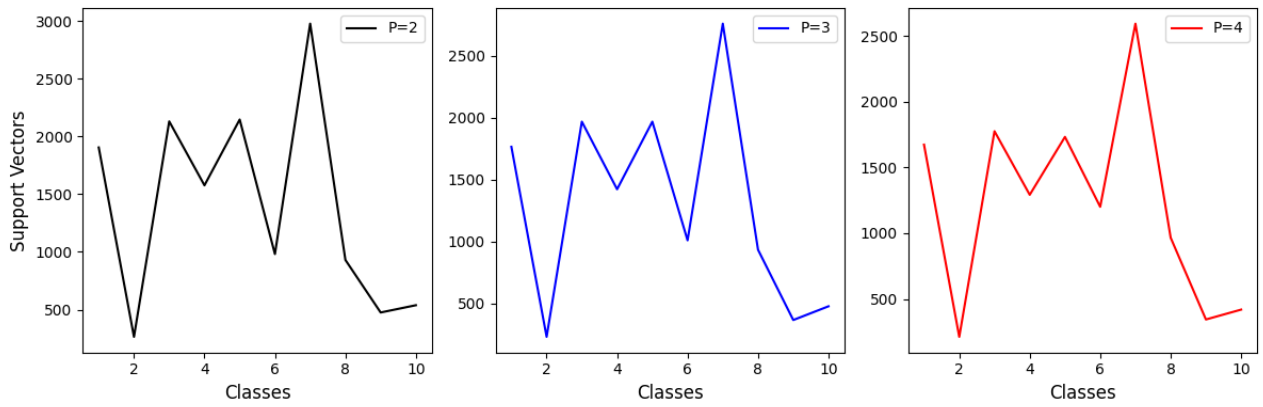


Figure 3: Support vector vs classes for each degree of polynomial

Figure 3 illustrates the number of support vectors for each class across different polynomial degrees ($P = 2$, $P = 3$, and $P = 4$). Each subplot corresponds to a specific degree of the polynomial kernel, showing the variation in the count of support vectors across the classes.

From the figure, it is evident that the number of support vectors remains relatively similar across all three polynomial degrees for each class. This observation suggests that the count of support vectors is more dependent on the underlying structure of the data rather than the degree of the polynomial used in the kernel function. In other words, the nature and complexity of the data itself, rather than the degree of the polynomial, primarily influence the number of support vectors used by the model.

# 9 P2: Kernelized Perceptron


Figure 4: Number of mistakes vs iterations

Figure 4 illustrates the number of mistakes vs iterations for a kernelized perceptron model, demonstrating how the model improves with each iteration. The x-axis shows the iteration number, and the y-axis indicates the number of mistakes made during training. The steady decline in mistakes reflects the model's learning and convergence over time.

A second-degree polynomial kernel was chosen since it provided the highest accuracy during previous implementation. This kernel maps the input data to a higher-dimensional space, allowing the perceptron to learn more complex, non-linear patterns effectively.

| Training Accuracy | Validation Accuracy | Testing Accuracy |
|---|---|---|
| 0.7779 | 0.7732 | 0.7662 |

Table 2: Accuracy table for kernelized perceptron

Table 2 compares the accuracy across different kernel choices, confirming that all kernels achieved comparable performance, but the second-degree polynomial kernel was selected due to its slight edge in accuracy.

# 10 P3: Decision Trees

## 10.1 a

The Decision Tree was implemented in Python and tested on the Breast Cancer Wisconsin Diagnostic dataset, evaluating performance in classifying tumors as malignant or benign.

## 10.2 b

The code was implemented, and the accuracy is shown in the table. The validation and testing accuracies are nearly identical.

| Validation Accuracy | Testing Accuracy |
|---|---|
| 0.9298 | 0.9211 |

Table 3: Decision Tree Accuracy

## 10.3 c

The Decision Tree was implemented along with pruning algorithm in Python and tested on the Breast Cancer Wisconsin Diagnostic dataset, evaluating performance in classifying tumors as malignant or benign.

## 10.4 d

After pruning, the validation accuracy improved significantly, indicating that the original model faced overfitting. Pruning helped reduce overfitting, making the decision tree more generalizable.

| Validation Accuracy | Testing Accuracy |
|---|---|
| 0.9825 | 0.9386 |

Table 4: Decision Tree Accuracy after pruning