Homework #2 CPTS-570 Machine Learning

Ricardo Rivero - WSU ID 011843796

October 19, 2024

Analytical Part.

1.1 Simple "CLOSE" classifier.

1.1a The decision boundary of CLOSE is a linear hyperplane of the form $sign(w \cdot x + b)$.

To start let's define how the algorithm will classifies the examples, since it will assign the class whose centroid is closer to the training example x_i, we can calculate the Euclidian distance between x_i and C+ and C-. Yielding the following classification rule:

$$x \leftarrow +, \quad if \quad \|x - C_+\| < \|x - C_-\|$$
$$x \leftarrow -, \quad if \quad \|x - C_-\| < \|x - C_+\|$$

Then, to find the decision boundary, we compute the squared Euclidean distances and compare them. The classifier will asign x based on the sign of:

$$\|x - C_+\|^2 - \|x - C_-\|^2$$

Expanding these terms and then subtracting the two distances:

$$\|x - C_+\|^2 - \|x - C_-\|^2 = x^T x - 2x^T C_+ + C_+^T C_+ - \left(x^T x - 2x^T C_- + C_-^T C_-\right)$$

Since the decision boundary is defined by the set of points where this difference is equal to zero:

$$-2x^T(C_+ - C_-) + (C_+^T C_+ - C_-^T C_-) = 0$$

Which is a linear equation in x, therefore the decision boundary is a linear hyperplane of the form $sign(w \cdot x + b)$ where $w = C_+ - C_-$ and $b = \frac{1}{2}(C_+^T C_+ - C_-^T C_-)$

1.1b Dual weigth vectors and number of support vectors.

From the previous part, we have that $w = C_+ - C_-$, and we know from theory that the weight vector can be written as a linear combination of the support vectors:

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

Then, let's recall that the centroids are defined as $C_+ = \frac{1}{n_+} \sum_{i:y_i+1} x_i$ and $C_- = \frac{1}{n_-} \sum_{i:y_i=-1} x_i$, then we can write the weight vector as:

$$w = \frac{1}{n_+} \sum_{i:y_i=+1} x_i - \frac{1}{n_-} \sum_{i:y_i=-1} x_i$$

Finally, if we map the weight vector into the dual form, the dual weights α_i are:

$$\alpha_i = \frac{1}{n_+}, \quad \text{for all i where } y_i = +1$$
$$\alpha_i = \frac{1}{n_-}, \quad \text{for all i where } y_i = -1$$

Suggesting that each positive example contributes $\frac{1}{n_+}$ to the weight vector, and each negative example contributes $\frac{1}{n_-}$. In that sense, since all the training examples contribute to the decision boundary, all training examples are support vectors. Therefore:

$$No_S upport Vectors = n_+ + n_-$$

1.2 Radial Basis Function (RBF) Kernel.

1.2a Proof that the mapping $\phi(x)$ corresponding to the RBF kernel has infinite dimensions.

Let's suppose that we have a Radial Basis Function (RBF) Kernel as follows:

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

To prove that the mapping $\phi(x)$ has infinite dimensions we can start by expanding the square, such that:

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

Then, we use the Taylor expansion to approximate the function, which in the case of an exponential function is:

$$f(x) = e^0 + \frac{e^0}{1!}(x-0) + \frac{e^0}{2!}(x-o)^2 + ... + \frac{e^0}{\infty!}x^\infty \tag{3}$$

Which can be further simplified as:

$$f(x) = 1 + x + \frac{1}{2}x^2 + ... + \frac{1}{\infty!}x^\infty \tag{4}$$

If we use this expansion to replace $\exp(xy)$, the equation becomes:

$$\exp\left(-\frac{1}{2}(x_i^2 + x_j^2)\right)(1 + x_i x_j + \frac{1}{2}x_i^2 x_j^2 + ... + \frac{1}{\infty!}x_i^\infty x_j^\infty) \tag{5}$$

Then, we replace the $\exp\left(-\frac{1}{2}(x_i^2 + x_j^2)\right)$ with $s = \sqrt{\exp\left(-\frac{1}{2}(x_i^2 + x_j^2)\right)}$, such that:

$$K(x_i, x_j) = s^2(1 + x_i x_j + \frac{1}{2!}x_i^2 x_j^2 + ... + \frac{1}{\infty!}x_i^\infty x_j^\infty) \tag{6}$$

Finally, we can rewrite the RBF into a dot product as described below:

$$K(x_i, x_j) = \langle s, sx_i, s\sqrt{\frac{1}{2}}x_i^2, ..., s\sqrt{\frac{1}{\infty!}}x_i^\infty\rangle \cdot \langle s, sx_j, s\sqrt{\frac{1}{2}}x_j^2, ..., s\sqrt{\frac{1}{\infty!}}x_j^\infty\rangle \tag{7}$$

Proving that the RBF kernel maps each data point into an infinite dimensional space.

1.2b Proof that for any two input examples x_i and x_j, the squared euclidean distance in the higher-dimensional space is less than 2.

To prove that $\|\phi(x_i) - \phi(x_j)\|^2$, let's start by expanding the squared euclidean distance between the two vectors:

$$\|\phi(x_i) - \phi(x_j)\|^2 = \|\phi(x_i)\|^2 + \|\phi(x_j)\|^2 - 2(\langle\phi(x_i), \phi(x_j)\rangle \tag{8}$$

Then, recalling that the RBF kernel is given by:

$$K(x_i, x_j) = \exp(-\frac{1}{2}\|x_i - x_j\|^2) \tag{9}$$

And that $\|\phi(x)\|^2 = K(x_i, x_i)$, which gives:

$$K(x_i, x_i) = \exp(-\frac{1}{2}\|x_i - x_i\|^2) = \exp(0) = 1 \tag{10}$$

Similarly, $K(x_j, x_j) = 1$, therefore:

$$\|\phi(x_i)\|^2 = 1, \|\phi(x_j)\|^2 = 1 \tag{11}$$

If we substitute the values into the distance formula:

$$\|\phi(x:i) - \phi(x:j)\|^2 = 1 + 1 - 2\langle\phi(x_i), \phi(x_j)\rangle \tag{12}$$

Which using the definition of the RBF kernel and substituting the RBF equation, can be expressed as:

$$\|\phi(x_i) - \phi(x:j)\|^2 = 2 - 2\exp(-\frac{1}{2}\|x_i - x_j\|^2) \tag{13}$$

Finally, since the exponential term $\exp(-\frac{1}{2}\|x_i - x_j\|^2)$ is always positive and less than or equal to 1, the entire expression is always less than or equal to 2:

$$2 - 2\exp(-\frac{1}{2}\|x_i - x_j\|^2) \leq 2 \tag{14}$$

1.3 Decision boundary of a SVM with RBF, prove that $f(x_far; \alpha, b) \approx b$.

Let's recall that the SVM decision function $f(x; \alpha, b)$ is given by:

$$f(x; \alpha, b) = \sum_i y_i \alpha_i K(x, x_i) + b \tag{15}$$

Therefore, if we apply the decision function for the test point x_far which is far away from any of the training points in the original feature space:

$$f(x; \alpha, b) = \sum_i y_i \alpha_i K(x_far, x_i) + b \tag{16}$$

And substitute RBF kernel, knowing that x_far is far from all training examples, then the distance $\|x_far - x_i\|$ becomes large, making the RBF kernel term approach zero. Therefore, if the kernel becomes approximately zero for all x_i, the summation term in the decision function approaches zero:

$$f(x_far; \alpha, b) = \sum_i y_i \alpha_i \cdot 0 + b = 0 + b$$

Therefore:

$$f(x_far; \alpha, b) \approx b$$

1.4 is$-\langle x_i, x_j \rangle$ a valid kernel?

For any function to be a kernel, it must satisfy the Mercer's theorem, which states that it must result in a symmetric and positive semidefinite matrix.

If the kernel is symmetric, then: $K(x_i, x_j) = K(x_j, x_i)$ In this case, since $K(x_i, x_j) = -\langle x_i, x_j \rangle$, we can apply the commutative property of the inner product, and therefore:

$$K(x_i, x_j) = K(x_j, x_i) \tag{17}$$

Therefore, it is symmetric.

Then, the kernel should be a positive definitive matrix: $z^T M z > 0$ So, let's construct the kernel matrix K for a small set of points $x_1, x_2, ..., x_n$:

$$K = \begin{pmatrix} -\langle x_1, x_1 \rangle & -\langle x_1, x_2 \rangle & \cdots & -\langle x_1, x_n \rangle \\ -\langle x_2, x_1 \rangle & -\langle x_2, x_2 \rangle & \cdots & -\langle x_2, x_n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ -\langle x_n, x_1 \rangle & -\langle x_n, x_2 \rangle & \cdots & -\langle x_n, x_n \rangle \end{pmatrix} \tag{18}$$

For a vector $z = (z_1, Z - 2, ..., z_n)^T$, we have $z^T K z = -\sum_{i,j} z_i z_j \langle x_i, x_j \rangle$, this expression can be written as:

$$z^T K z = -\langle z_1 x_1 + z_2 x_2 + \cdots + z_n x_n, z_1 x_1 + z_2 x_2 + \cdots + z_n x_n \rangle \tag{19}$$

Resulting in the negative of a squared norm:

$$z^T K z = -\|z_1 x_1 + z_2 x_2 + \cdots + z_n x_n\|^2 \tag{20}$$

And since the squared norm is always non-negative, $z^T K z$ must always be non-positive, showing that $K(x_i, x_j) = -\langle x_i, x_j \rangle$ is **not positive semi-definite**, failing to satisfy the second condition of the Mercer's theorem, and therefore is not a valid kernel.

1.5 Modification of the Soft-margin SVM formulation to include mistake cost.

To leverage the additional information on the cost of making misclassifications, given by C+ and C- for a misclassified positive and negative example respectively I would propose a weighted SVM. In this approach, the penalty terms are scaled differently for positive and negative classes, therefore the classifier would treat the two types of error asymmetrically.

This could be implemented by scaling the C parameter by using the C+ and C- for positive and negative examples, such that the objective function would be:

$$\min_w \frac{1}{2}\|w\|^2 + C_+ \sum_{i \in positive} \xi_i + C_- \sum_{i \in negative} \xi_i \tag{21}$$

1.6 Coarse-to-fine framework for a very large dataset.

1.6a K+ and K- clusters.

Let's assume that the clustering is done using k-means clustering, defined by:

$$\sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2 \tag{22}$$

Since we have already defined which clusters are positive or negative, we can implement an asymmetric penalization, as implement in the previous question, where the cost of misclassifying negatives are positives and vice versa is different given the sign of the cluster.

1.6b Refine the clusters based on the SVM classifier.

We can refine each cluster K_i by performing sub clustering and represent each sub cluster by its centroid (μ_k), which will be used for training the local SVMs.

1.6c Stopping point.

The stopping point must be defined by the accuracy after i iterations if the accuracy is not increasing and the accuracy in the testing and validation data starts to decrease, which might indicate overfitting due to having too many clusters, and therefore too many support vectors.

1.7 Online kernelized perceptron with B support vectors.

1.7a Algorithm to select B support vectors from SV.

Since the goal is the reduce the number of kernel evaluations from SV to B, we can greedily select the support vectors that have the highest contribution to the decision boundary as follows:

1. **Compute the contribution of each support vector in SV.**: For each support vector $x_i \in SV$, we compute the average contribution to the margin across all data points.

$$c_i = \sum_{j=1}^{n} \alpha_i y_i K(x_i, x_j) \tag{23}$$

2. **Rank support vectors**: Rank the support vectors in SV based on the magnitude of their contribution to the decision boundary (c_i).
3. **Select the top B contributing support vectors**
4. **Train with the selected B support vectors**

Following this approach, Tom will be able to subset the number of support vectors, decreasing the number of kernel evaluations from SV to B while keeping the support vectors that contribute the most to the margin, preserving the classification performance as much as possible.

1.7b Algorithm to train an online kernelized perceptron with $SV \leq B$.

Here, we can modify the perceptron's algorithm to ensure that the number of support vectors never exceeds B, which will require us to bound the number of support vectors during training while keeping the update rule for misclassifications.

1. **Initialization:** Starting with an empty set of support vectors $SV = []$
2. **For each training example** (x_i, y_i)**:** compute the prediction using the current sent of support vectors: $f(x_i) = \sum_{j \in SV} \alpha_j y_j K(x_j, x_i)$
3. **if error:** if the number of support vectors (SV) is less than B, add x_i to SV.
4. if $\mathbf{SV = B}$: replace one of the existing support vectors with x_i based on the contributions to the margin, such that the support vector that contributes the least is replaced with x_i.
5. **Update the weights:** After each update, adjust the corresponding α_i for the new support vector.

Following this approach, Tom can keep the number of support vectors under control, yielding a bounded complexity that will remain scalable even for large datasets. Additionally, the vector replacement strategy will ensure that the model will adapt to new examples while maintaining a fixed number of support vectors.

Programming and Empirical Analysis Part.

2.1 Support Vector Machine Classifier.

2.1a Linear Kernel

To implement an SVM with a Linear Kernel, we tried the implementations available in Scikit-Learn, namely SVC and LinearSVC, however, the latter was found to have a better efficiency, explained by the use of a "One-versus-the-rest" multiclass approach compared to the regular SVC which uses a "one-versus-one" approach. Then a grid search with 5 folds was done to determine the highest scoring C value, which was 10e-2.

Training data was split into training and validation data following a 80/20 proportion, and both datasets were scaled using the StandardScaler function from Scikit-learn to standardize the data. A linearSVC model was trained using C values ranging from 10e-4 to 10e4 in log10 increments for each iteration. For this approach the highest scoring C value was also 10e-2, with an accuracy of 0.845 on the validation data.

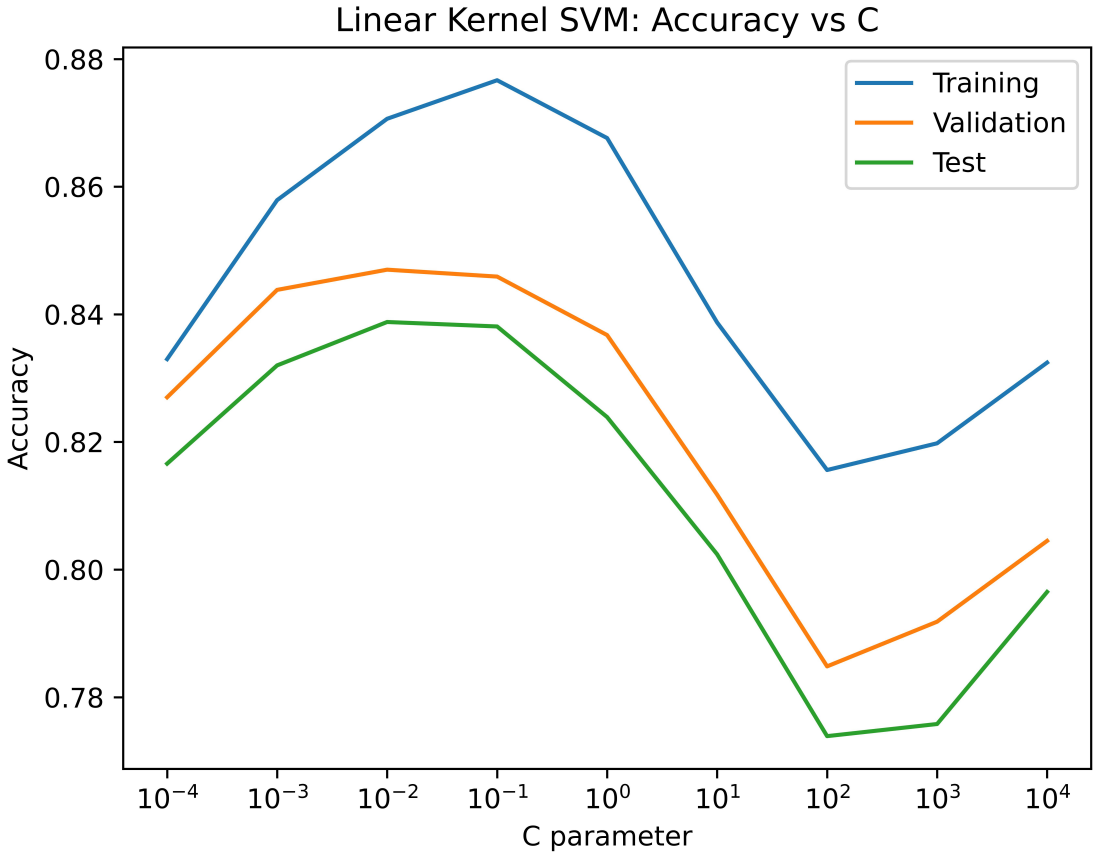

Figure 1: Accuracy of the SVM with Linear Kernel as a function of the hyperparameter C.

Regarding the number of support vectors vs C, since the slack variable C dictates the size of the margin, a higher number of support vectors for the lowest C value indicates a higher number of data points that lie on the margin or violate it, which can be seen in the lower accuracy of the SVM when C=10e-4. On the other hand, as C increases, the number of support vector stabilizes, suggesting a narrower margin that is able to classify the data better, however, as observed in the accuracy plot, the accuracy decreases after C¿1, likely due to overfitting.

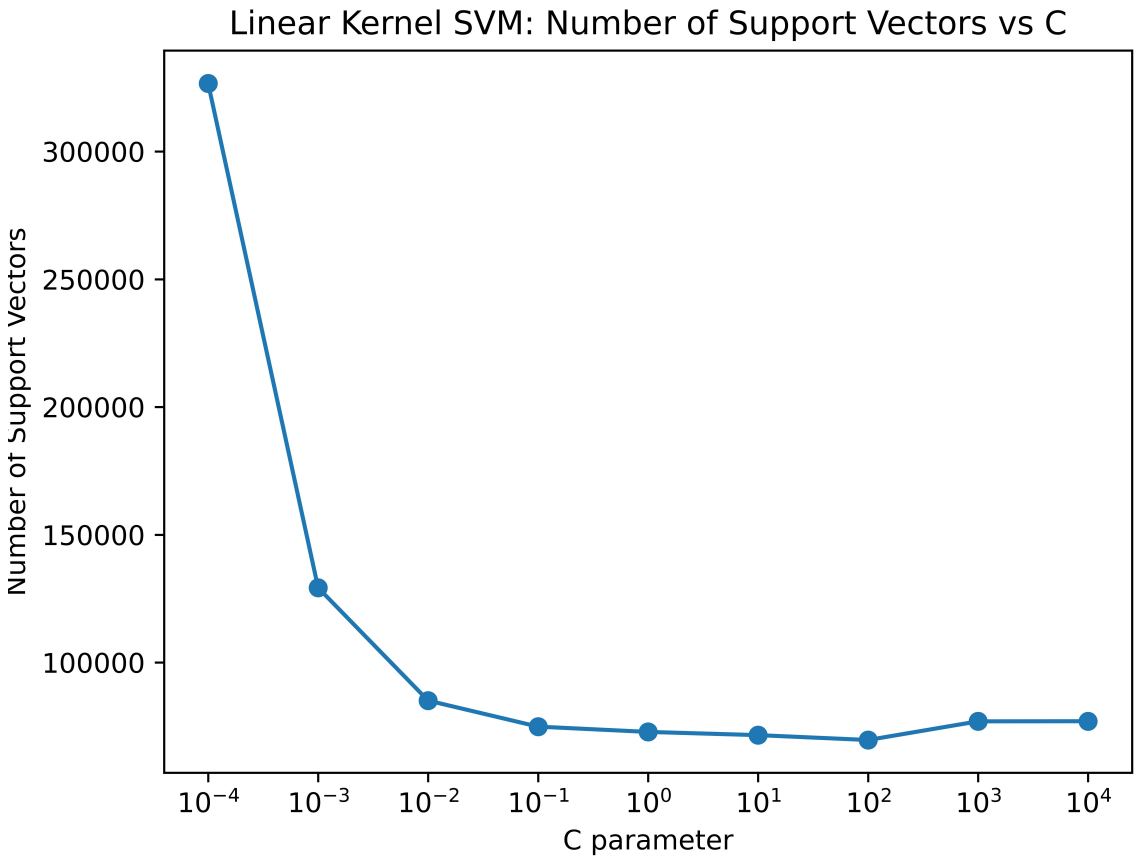

Figure 2: Number of support vectors as a function of the hyperparameter C.

2.1b Linear SVM on the combined set of training and validation examples.

As determined before, the best value of C is 10e-2, therefore we fitted a linear SVM using the combined training and validation data and 1000 iterations, applying a StandardScaler normalization before training the model, obtaining an accuracy of 0.8376. Then, a confusion matrix was built using the true and predicted label for the 10 classes.

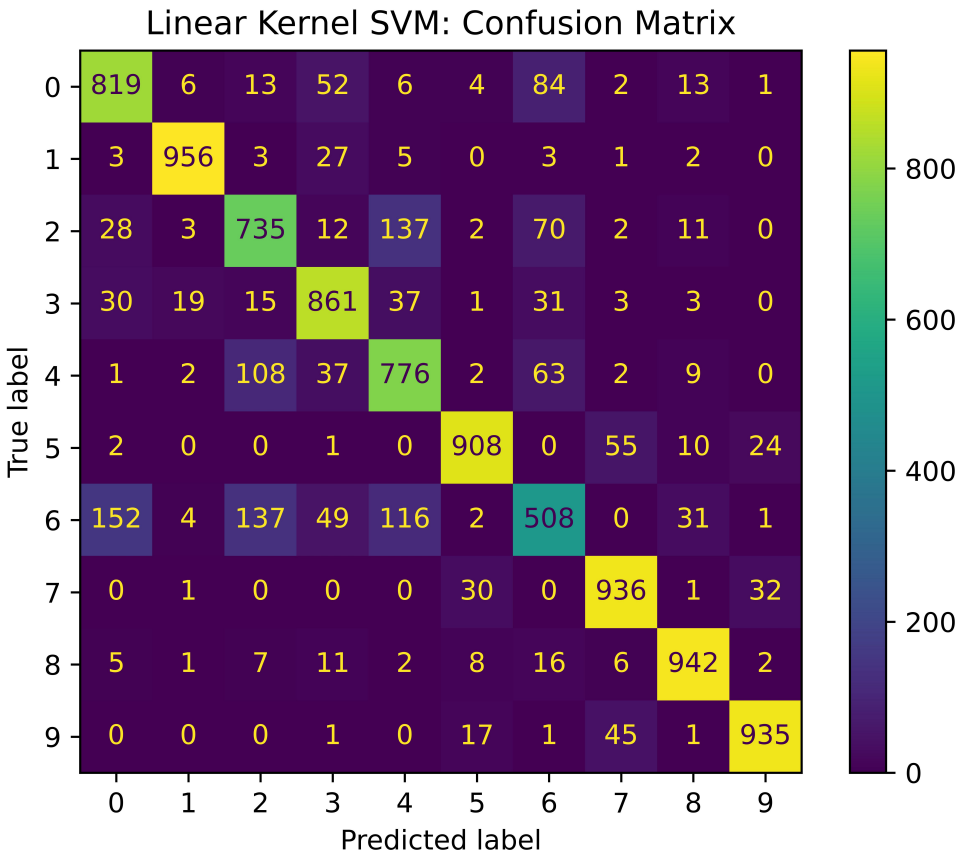

Figure 3: 10x10 Confusion Matrix for the Linear Kernel SVM.

As observed, some labels where classified with a higher accuracy than other, with labels 2, 4, and 6 having the lowest accuracy.

2.1c Polynomial SVM.

A support vector machine classifier was trained using polynomial kernels of degree 1, 2, 3, and 4. For the polynomial kernel of degree 1, since this is equivalent to a linear kernel function, we used the LinearSVC within Scikit-learn as described before. For the non-linear polynomial kernels, the regular function SVC was used with the parameter "kernel = 'poly'". A C value of 10e-2 and a maximum number of iterations were used for all the classifiers, data was normalized using a Standard Scaler.

After training, the polynomial degree that yielded the highest accuracy was the linear polynomial, with a validation accuracy of 0.79 and 68314 support vectors. For the polynomials of degree 2, 3, and 4, the validation accuracy was 0.58, 0.56, and 0.524 while the number of support vectors was 20,182, 18,222, and 16,894 respectively. Additionally, the polynomial kernels took a significant amount of time to be trained (15 minutes each) compared to 1.9 minutes for the linear kernel.

Overall, these experimental results suggest that the linear polynomial kernel performs better for the specific classification problem of the fashion_mnist dataset, resulting in a higher accuracy in a considerably lower amount of time.

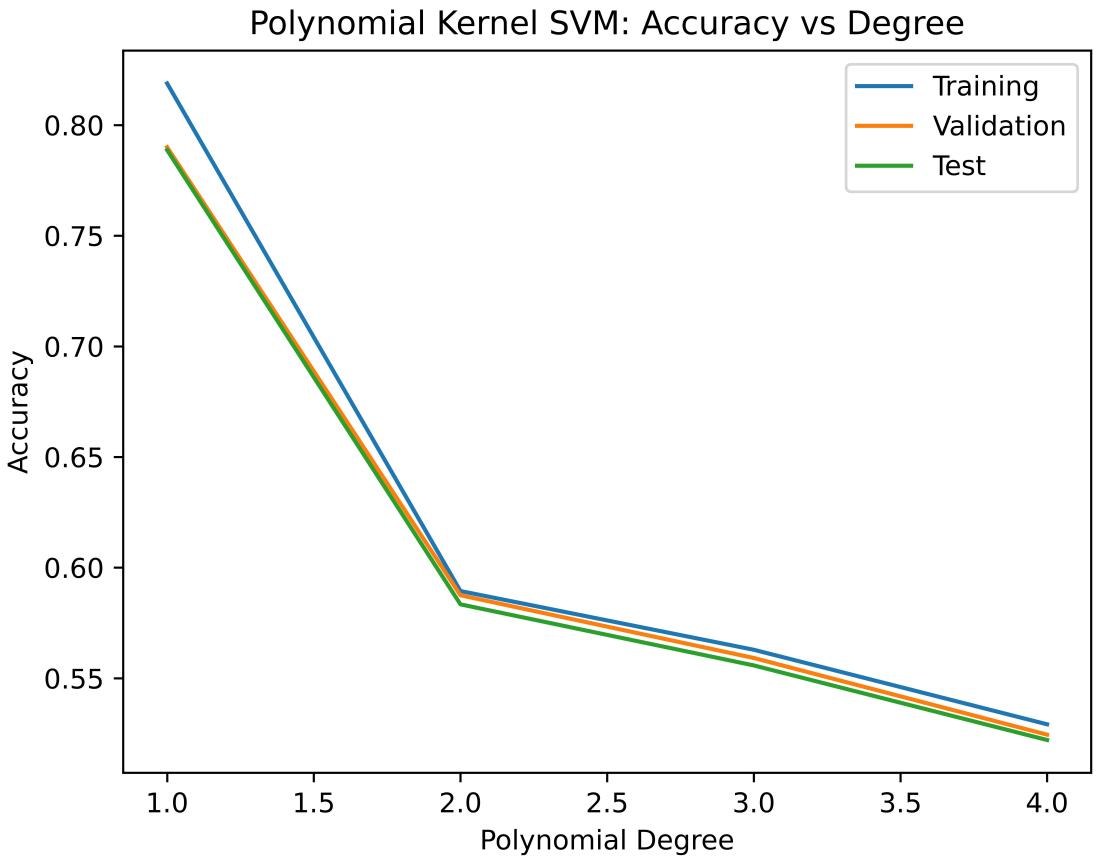

Figure 4: Accuracy of a polynomial SVM as a function of the degree of the polynomial kernel.

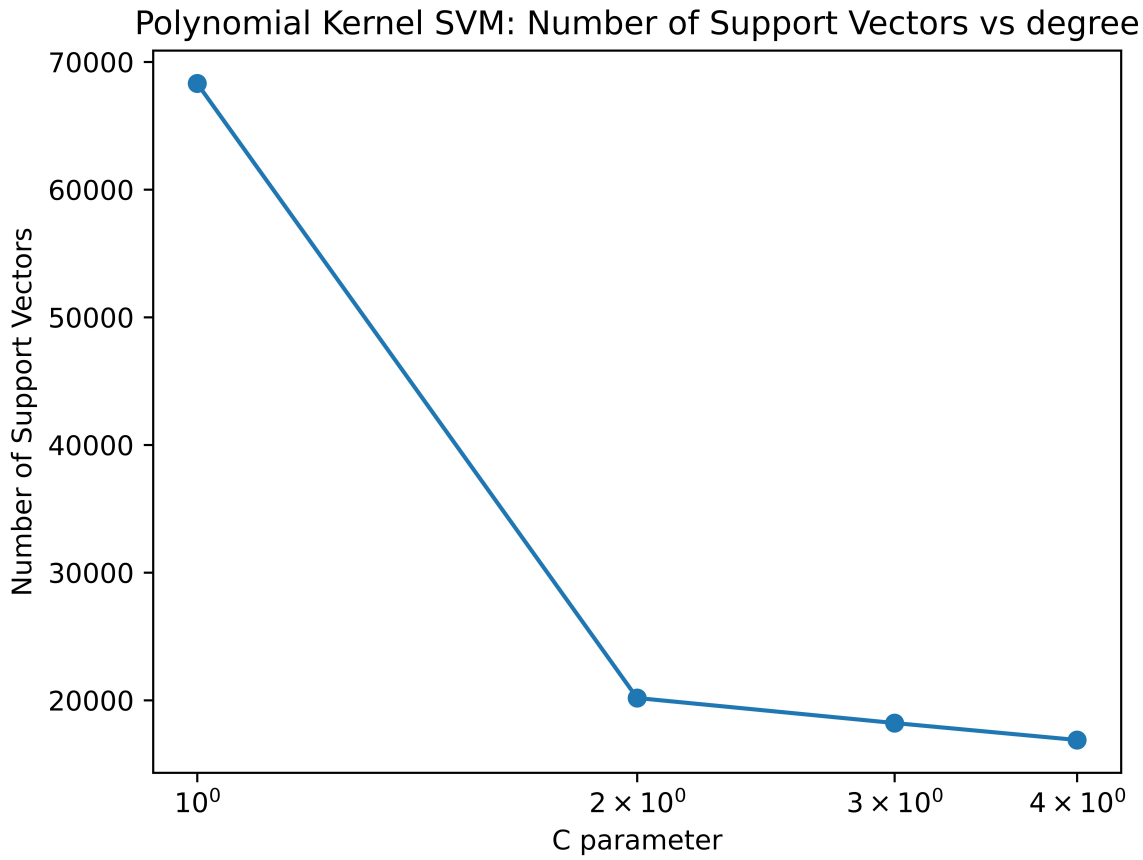

Figure 5: Number of support vectors as a function of the degree of the polynomial kernel for a polynomial SVM.

2.2 Kernelized Perceptron for multi-class classification.

We implemented a Kernelized Perceptron using a polynomial kernel of degree 2, which was previously determined to perform the best among the non-linear polynomials. therefore we defined the polynomial kernel function as $K(x, y) = (1 + (x_n^T \cdot y_n))^p$, where **p** is the polynomial degree, equal to 2 in this case. The training and testing datasets were normalized using a Standard Scaler before splitting the training dataset into a training and validation datasets using a 80/20 proportion.

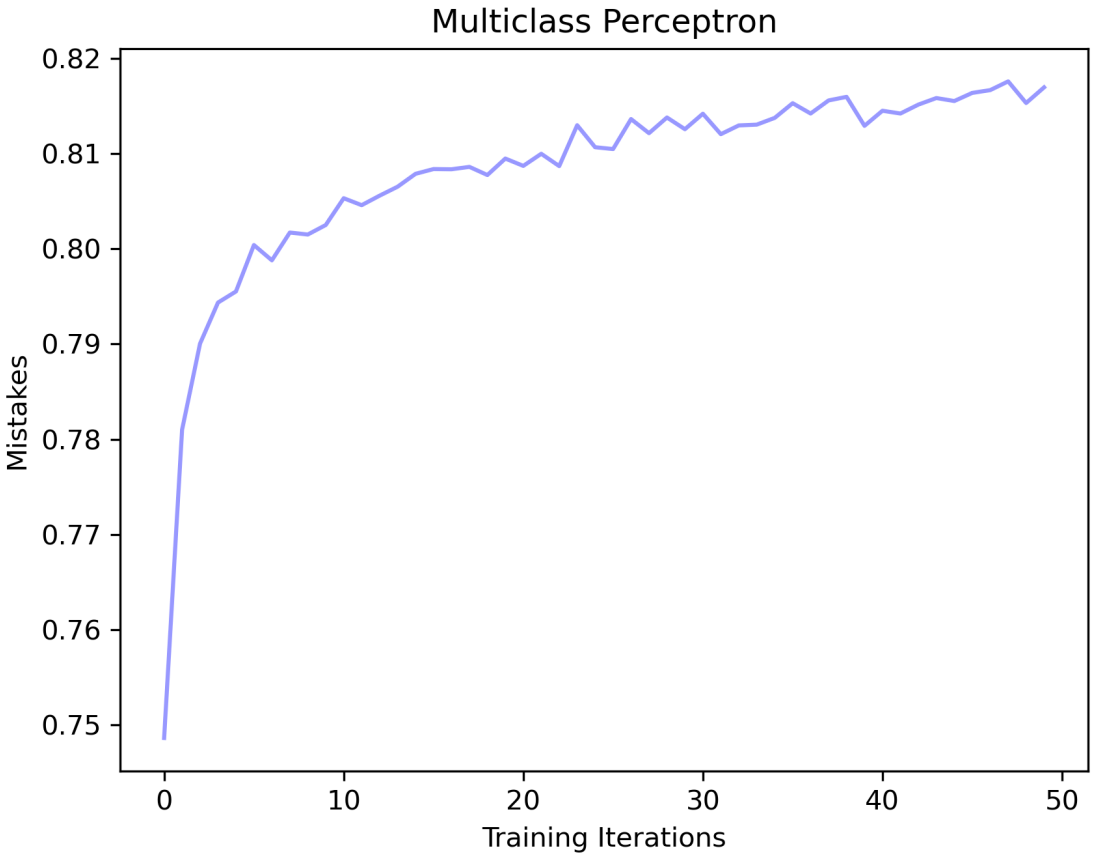

Figure 6: Number of mistakes made by a Polynomial Kernelized Multi-class perceptron over 5 training iterations.

After training the Kernelized perceptron for 5 iterations, the number of errors decreases from 18,000 to 13,500 while the accuracy on the validation dataset increased from 0.70 to 0.76. When compared to the previous work done with standard and Passive-Aggressive perceptron algorithms, these results suggest that although the accuracy of the kernelized perceptron is lower, the accuracy on both the validation and testing datasets have a low variance in respect to the training accuracy, suggesting a better generalization capability compared to a regular perceptron.

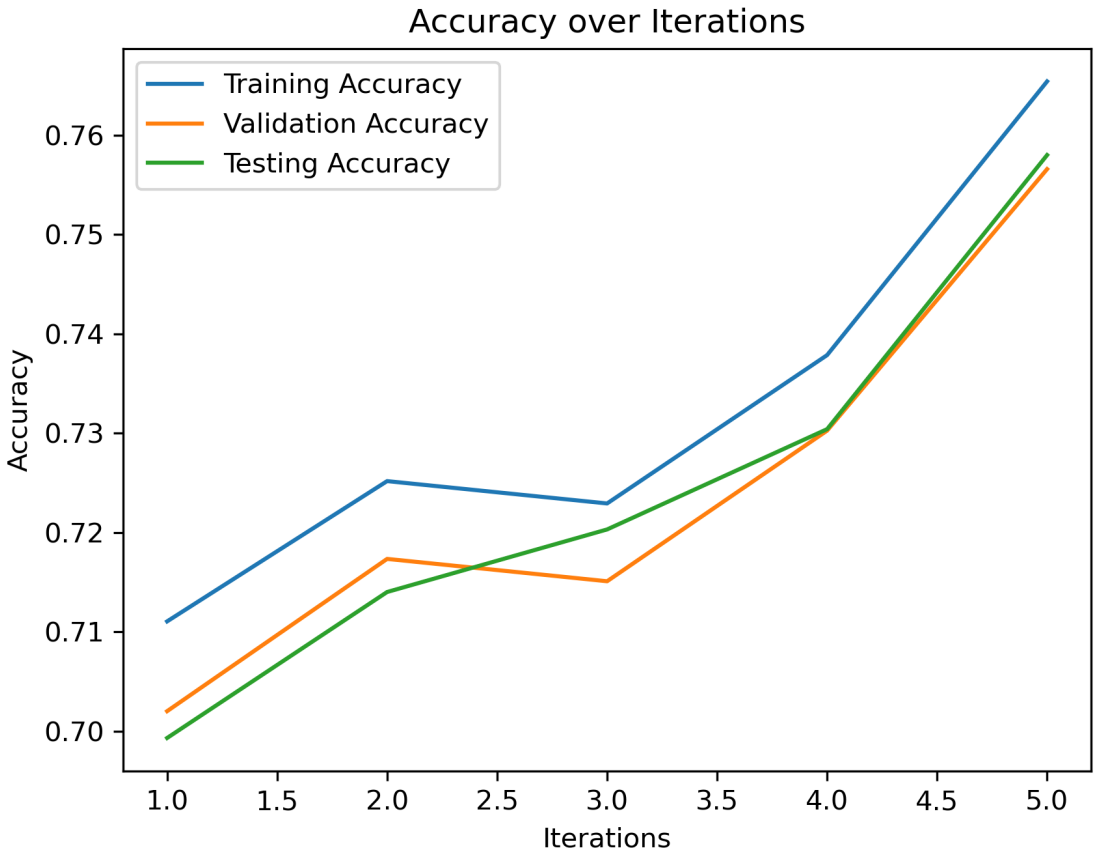

Figure 7: Accuracy on the training, validation, and testing datasets of a Polynomial Kernelized Multi-class Perceptron over 5 training iterations.

2.3 Breast cancer dataset classification using decision trees.

2.3a Implementation of the ID3 algorithm.

an ID3 algorithm was built by implementing functions for calculating the entropy of the class variable and a function to calculate the information gain from each predictor variable. The variable with the highest information gain was used in the root node, and choosing the split value for continuous data by obtaining all the unique values to recursively evaluate the best split for the variable. Then, the tree was evaluated using the validation and test accuracies.

2.3b Tree construction.

The tree building algorithm was run with the training dataset, giving accuracy values of 0.982 and 0.930 for the validation and testing datasets respectively.

2.3c implementation of a post-pruning algorithm.

As seen in class, we implemented an algorithm that evaluates each node of the tree, from the bottom up an calculates the accuracy in the validation dataset with and without the node, and if the accuracy does not decrease when removing the node, the algorithm removes the node from the tree.

2.3d Performance of the tree with the post-pruning algorithm

Building the tree with the post-pruning algorithm resulted in an increase of 0.09 in the test accuracy while the validation accuracy remained the same. A total of 4 nodes where pruned from the tree: 'mean compactness' with value ≤ 0.06217, 'mean compactness' with value ≤ 0.07698, 'worst concave points' with value ≤ 0.09331, and 'worst texture' with value ≤ 30.25.

2.3 Extra. Validation with Scikit-learn.

The implementation of decision tree classifiers available from Scikit-learn was run on the dataset to compare the performance of our implementation, yielding an accuracy of 0.982 and 0.930 for the validation and testing datasets respectively. This accuracy is exactly the same as our ID3 implementation.

The resulting tree was the following:

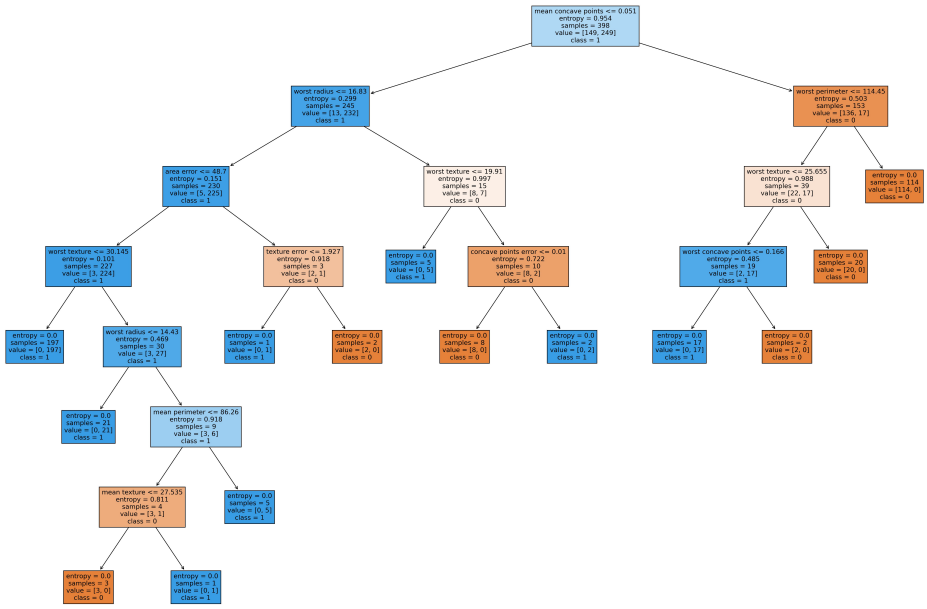

Figure 8: Decision Tree built with Scikit-learn on the Breast Cancer dataset.