Building and Evaluating SVMs for CIFAR-10 Multi-Label Classification

Aimilia Palaska 10453 - aimiliapm@ece.auth.gr
Academic Supervisor: Anastasios Tefas

December 23, 2024

1 Introduction

This report is created as part of the Neural Networks & Deep Learning course in Electrical and Computer Engineering department of Aristotle University of Thessaloniki. The developed code in python, along with detailed comments, can be found in the following GitHub Repository [1].

Support Vector Machines (SVMs) are widely used for solving real-world classification problems due to their robust theoretical foundation and ability to handle complex decision boundaries. This report focuses on understanding the behavior of SVMs and their kernels through practical experiments.

A dataset was chosen, processed, and visualized to serve as a benchmark for SVM evaluation. Various kernel functions, including linear, polynomial, radial basis function (RBF), and sigmoid, were implemented and tested to investigate their impact on classification performance. Additionally, to understand the optimization process inherent to SVM training, Quadratic Programming (QP) solvers were explored, albeit with limited success due to computational constraints.

2 Dataset Overview

2.1 Data Format and Structure

The CIFAR-10 dataset [2], originally introduced by Alex Krizhevsky, is a widely used benchmark in machine learning and computer vision, consisting of 60,000 color images categorized into 10 distinct classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each image is 32x32 pixels in size and contains a single object from one of these classes, making it a relatively low-resolution dataset ideal for testing image classification models and algorithms. In Figure 1, a grid of 16 random samples, along with their respective label, is presented for the purpose of visualization. Split into 50,000 training images and 10,000 test images, CIFAR-10 offers a balanced and challenging dataset that facilitates evaluation and comparison of model performance across a broad range of supervised learning techniques.

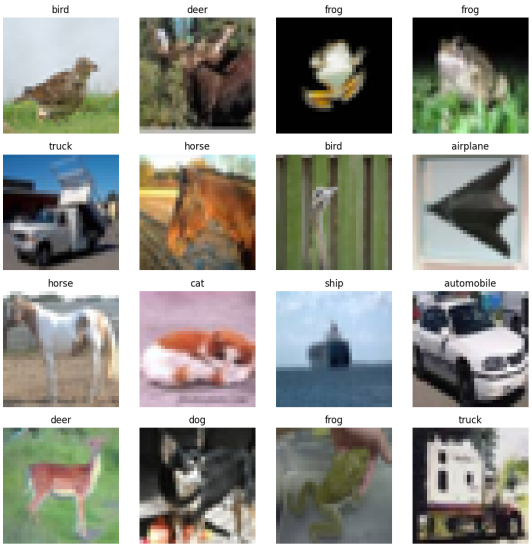


Figure 1: Random samples of the CIFAR-10 dataset and their respective labels.

The CIFAR-10 dataset has a uniform distribution of samples across its 10 classes, each class containing 6,000 images, which eliminates the need for downsampling and ensures that there is no inherent bias towards any particular class during training. This can be seen in Figure 2, where the ten classes are plotted with the corresponding number of instances in the labels array.

2.2 Preprocessing

Since CIFAR-10 is a clean dataset without duplicates or missing values, it requires no preprocessing for data integrity.

However, as the images have pixel values ranging from 0 to 255, normalization is essential for training in order to standardize these values to facilitate more stable and effective learning. Two common normalization methods are applied: z-score normalization and min-max normalization. In z-score normalization, each pixel value x is transformed using $x' = \frac{x-\mu}{\sigma}$, where μ is the mean pixel value, and σ is the standard deviation of the pixel distribution. This method centers the values around 0 with a standard deviation of

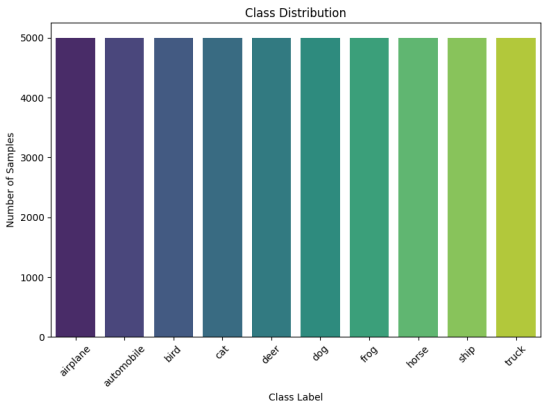


Figure 2: Class distribution of the CIFAR-10 dataset.

1. Alternatively, in min-max normalization, the pixel value x is scaled as $x' = \frac{x-min}{max-min}$, setting the range between 0 and 1, where min and max are the minimum and maximum pixel values (0 and 255, respectively).

3 Baseline Models

3.1 Algorithm Overview

Baseline models provide a reference point, allowing for a meaningful comparison to assess the neural network's accuracy and improvement over simpler methods. For this purpose, k-nearest neighbors and nearest centroid classifiers were implemented, as they are straightforward, widely understood algorithms for classification with minimal development complexity.

K-Nearest Neighbor (KNN) The k-nearest neighbors (KNN) algorithm is a simple, non-parametric method used for classification tasks, relying on the concept of feature similarity. For a given input sample, KNN identifies the k nearest samples in the dataset and assigns the input to the class most common among those neighbors. By varying k, the algorithm can be fine-tuned for different levels of sensitivity to local patterns in the data. Although computationally intensive, especially for large datasets, KNN offers a straightforward way to predict labels based purely on proximity to labeled samples, making it a useful baseline.

Nearest Centroid (NC) The nearest centroid classifier, on the other hand, simplifies the classification process by representing each class with a single prototype—its centroid. In this approach, the centroid of each class is calculated by averaging the feature values of all samples within that class. To classify a new sample, the algorithm assigns it to the class with the closest centroid. This method is computationally efficient since it reduces each class to a single point, making it faster than KNN while still capturing basic patterns within the data. However, it may struggle with complex class boundaries, as it assumes each class can be effectively represented by a central point.

3.2 Distance Calculation

For the classification of the CIFAR-10 dataset, four methods of calculating the distance between the images were implemented, for experimental purposes.

- **Pair-wise distance**: This approach calculates the absolute difference between corresponding pixels of two images, summing up these differences to get an overall measure of similarity. Pair-wise distance captures the pixel-level discrepancies, making it sensitive to small changes but computationally light.
- **Mean squared error (MSE)**: MSE calculates the average of the squared differences between pixel values of two images, represented as $MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2$ where x and y are the pixel values of the two images. This method penalizes larger differences more heavily, making it useful for detecting significant deviations, though it can be sensitive to minor changes.
- **Cosine distance**: This method measures the angle between two image vectors in high-dimensional space rather than their absolute pixel values. Calculated as $1 - \frac{x \cdot y}{\|x\|\|y\|}$, cosine distance captures differences in orientation rather than intensity. This approach is often effective for comparing patterns in images while disregarding brightness variations.
- **Structural similarity (SSIM)**: Structural similarity [3] aims to measure perceptual differences by comparing local patterns of pixel intensities, focusing on luminance, contrast, and structure. SSIM values range from -1 to 1, with values closer to 1 indicating greater similarity. This method aligns more closely with human visual perception, making it valuable for detecting nuanced changes in image structure.

4 Evaluation Metrics

To evaluate model performance in this classification task, four core metrics are employed: accuracy, precision, recall, and F-score. These metrics offer distinct insights: accuracy, as the primary metric, measures the overall correctness of predictions, calculated as the ratio of correctly predicted instances to the total number of instances. Precision quantifies the model's ability to avoid false positives, while recall assesses its ability to capture true positives. The F-score, as

a division of precision and recall, gives a balanced measure that is especially useful for imbalanced data scenarios.

Although test accuracy remains the most significant indicator of model performance, precision, recall, and F-score offer additional insights into robustness, potential overfitting, and class imbalances. These metrics will help us better understand how the model performs beyond simple accuracy when evaluating Neural Networks.

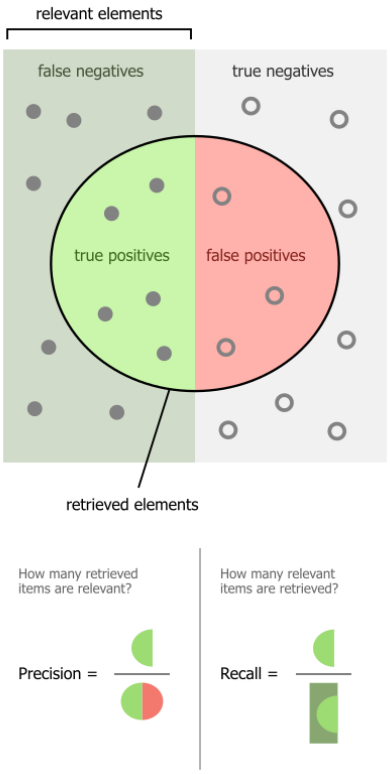


Figure 3: Precision and recall formula visualized for better understanding. Image by Wallber on Wikipedia [4].

In this baseline analysis, simpler models like k-nearest neighbor (KNN) and nearest centroid are expected to perform modestly, as they lack the complexity to capture the full structure of image data. However, these baseline metrics are essential as they establish a performance threshold. This threshold serves as a benchmark that the neural network model, with its greater complexity and adaptability, is expected to surpass, demonstrating its suitability for CIFAR-10 classification.

5 Preliminary Results with Baseline Models

This section presents the performance of the K-Nearest Neighbors (KNN) and Nearest Centroid (NC) algorithms on the CIFAR-10 dataset, using the predefined metrics. Results are analyzed for each model with a discussion on the strengths and limitations observed, setting expectations for the neural network's anticipated performance.

	Z-score		Min-Max	
	$k=1$	$k=3$	$k=1$	$k=3$
Accuracy	0,3548	0,3287	0,3539	0,3303
Precision	0,4118	0,4283	0,4112	0,4304
Recall	0,3548	0,3287	0,3539	0,3303
F-score	0,3503	0,318	0,3495	0,3192

Table 1: Performance of KNN model for two normalization methods, Z-score and Min-Max. It is observed that the performance of the models differs slightly, indicating similar behavior of the KNN algorithm for these normalization methods.

Comparison of Normalization Methods: To assess the impact of normalization on model performance, the KNN algorithm was tested with two common normalization techniques: Z-score and Min-Max. Results are summarized in Table 1, using pairwise distance metrics and evaluating performance at $k = 1$ and $k = 3$. Both normalization techniques performed similarly across most metrics, suggesting limited impact on the model's effectiveness. Given these similarities, Z-score normalization was selected for subsequent experiments, as it provided slightly more stable results overall.

Effect of Different k-values in KNN: Table 3 compares the KNN model's performance for $k = 1$ and $k = 3$, using several distance metrics. Generally, models with $k = 1$ achieved higher accuracy than those with $k = 3$, indicating that a closer, single nearest neighbor may yield better classification results in this dataset. However, despite this advantage, none of the tested configurations achieved accuracy scores above 50%, with the lowest performance observed in models using cosine similarity. This highlights a limitation in the baseline models' ability to classify image features accurately.

Comparison of KNN and Nearest Centroid (NC): A comparison of KNN and NC models in Table 2 reveals a trade-off between execution time and accuracy. While the NC algorithm demonstrated faster computation times, it generally delivered lower accuracy scores compared to KNN, except in the cosine similarity case. This indicates that NC may be a viable option when computational efficiency is prioritized, but its reduced accuracy makes it less ideal for more accurate classifications.

Distance Metric Performance Analysis: It is observed that the Mean Squared Error (MSE) and Pairwise metrics produced similar results, as expected, given their calculation similarity (MSE incorporates squared differences compared to Pairwise). Among all tested metrics, the Structural similarity metric emerged as the most

Method	Distance	Accuracy	Precision	Recall	F-score
KNN	Pair	0.3548	0.4118	0.3548	0.3503
	MSE	0.3548	0.4118	0.3548	0.3503
	Cosine	0.0992	0.1096	0.0992	0.0963
	Structural	0.407	0.4178	0.407	0.4055
NC	Pair	0.2784	0.2882	0.2784	0.2551
	MSE	0.2784	0.2882	0.2784	0.2551
	Cosine	0.2853	0.2931	0.2853	0.2696
	Structural	0.1955	0.219	0.1955	0.1626

Table 2: Comparison of k-nearest-neighbor ($k = 1$) and nearest centroid models for four distance calculation methods.

KNN	Distance	Accuracy	Precision	Recall	F-score
$k = 1$	Pair	0.3548	0.4118	0.3548	0.3503
	MSE	0.3548	0.4118	0.3548	0.3503
	Cosine	0.0992	0.1096	0.0992	0.0963
	Structural	0.407	0.4178	0.407	0.4055
$k = 3$	Pair	0.3287	0.4283	0.3287	0.318
	MSE	0.3287	0.4283	0.3287	0.318
	Cosine	0.09	0.112	0.09	0.0812
	Structural	0.374	0.4031	0.374	0.372

Table 3: Comparison of k-nearest-neighbor models for $k = 1$ and $k = 3$ neighbors.

effective for this classification problem, yielding the highest accuracy scores. The best-performing model overall was the KNN model with Z-score normalization, $k = 1$, and the Structural distance calculation.

6 Experimental Setup

6.1 MLP baseline with Hinge-Loss

To establish an additional baseline, a Multi-Layer Perceptron (MLP) classifier was trained using the Hinge loss function. The MLP architecture includes an input layer corresponding to the feature space, one or more hidden layers with non-linear activations, and an output layer that predicts the class scores. The Hinge loss function, commonly used in Support Vector Machines (SVMs), is adapted for neural networks to penalize incorrect classifications with a margin. For a batch of N data-points, hinge loss is defined as:

$$L = \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - y_i \cdot \hat{y_i})$$

where $y_i \in -1, 1$ is the true label of the i data point and $\hat{y}$ is the predicted value.

Comparing SVMs with an MLP trained on the same loss function is crucial to evaluate the impact of model complexity and representation learning on performance. The MLP, implemented using `NumPy` [5], minimizes dependencies to better understand its fundamental operations. The training process utilized gradient descent for optimization, with parameters such as learning rate and number of epochs tuned experimentally. This comparison helps highlight the trade-offs between simpler models like SVMs and more flexible, data-driven architectures like MLPs.

6.2 SVMs

The main focus of this report is a Support Vector Machine (SVM) classifier implemented from scratch, without leveraging external libraries apart from `NumPy` [5]. Custom kernel functions were implemented to extend the SVM's functionality, allowing for both linear and non-linear separations (Polynomial, RBF & Sigmoid). To benchmark the custom SVM implementation, the SVM module from the `Sklearn` library [6] was employed. `Sklearn` provides a robust and efficient implementation of SVMs, supporting various kernels and regularization parameters. The library's flexibility allowed for training and evaluating SVM models with linear, polynomial, sigmoid and radial basis function (RBF) kernels. This comparison helped validate the correctness and performance of the scratch implementation and provided insights into practical considerations such as scalability and parameter tuning.

6.3 Kernels

A brief description of each kernel type:

- **Linear Kernel:** The linear kernel is the simplest form, defined as:

$$K(x, x') = x^T x'$$

where x and x' are feature vectors. This kernel assumes that the data is linearly separable in the original feature space.

- **Polynomial Kernel:** The polynomial kernel allows for non-linear decision boundaries and is defined as:

$$K(x, x') = (\gamma x^T x' + r)^d$$

where d is the degree of the polynomial (set to 3 in this report), γ is a scaling parameter, and r is a constant. This kernel captures interactions between features of a specific degree.

- **Radial Basis Function (RBF) Kernel:** The RBF kernel is a popular choice for non-linear problems, defined as:

$$K(x, x') = \exp(-\gamma \|x - x'\|^2)$$

where γ determines the influence of a single training example. It effectively maps data into a higher-dimensional space where a linear separator can be applied.

- **Sigmoid Kernel:** The sigmoid kernel, inspired by neural networks, is given by:

$$K(x, x') = \tanh(\gamma x^T x' + r)$$

where tanh is the hyperbolic tangent function. This kernel introduces non-linearity but is less commonly used due to its sensitivity to parameter tuning.

6.4 Quadratic Programming

Quadratic Programming (QP) methods were utilized to solve the SVM optimization problem directly. The QP formulation focuses on minimizing a convex quadratic objective function subject to linear constraints, which corresponds to the dual form of the SVM problem and can be expressed as:

$$\min_{\alpha} \frac{1}{2}\alpha^T Q \alpha - \mathbf{1}^T \alpha, \; 0 \leq \alpha_i \leq C, \; \sum_i \alpha_i y_i = 0$$

where Q is a matrix derived from the kernel, α are the Lagrange multipliers, C is the regularization parameter, and y_i are the labels.

A QP solver was employed to determine the support vector coefficients and validate the theoretical foundation of the SVM approach, using the `cvxopt` [7] library. Testing in this section was minimal due to the time and memory constraints of running a QP optimization problem on a large dataset such as CIFAR-10.

7 Results

The repository is structured as follows; the whole functionality is implemented in the `svm` module, separated into `svm.ready_models` and `svm.scratch` according to the implementation. The extra baseline model is located inside the `baselines` module and all the main functions, showcasing examples of experiments, are in the `source` directory. Finally, extensive results and plots can be found in the corresponding directories, outside of the `source` module.

Regarding the computational cost of the trainings and fittings, the SVMs from scratch took 0.1 to 1 sec per iteration, with fluctuations due to the kernel functions. Most SVMs were tested for 10 iterations, meaning 10 seconds maximum per classifier. The one-vs-one strategy for the multi-label classification trains 45 classifiers, resulting in around 8 minutes to fit all of them. With the computational cost of loading and saving data as well as extracting the final metrics, these tests did not last longer than 15 minutes. It should be noted that through more extensive hyperparameter tuning and potential integration of a cross validation mechanism this duration could be greatly affected. The ready models through `Sklearn` averaged in 2-3 hours each, following the same multi-label classification approach but including more iterations and optimization techniques.

7.1 MLP baseline with Hinge-loss

The results presented in Table 4 highlight the effect of tuning hyperparameters on the performance of the MLP classifier trained with the Hinge loss function. Experiment 1, using a higher learning rate of 0.1 and a shorter training duration of 30 epochs, achieved an accuracy of 20.62% with relatively lower precision (22.55%) and F-score (13.99%). In contrast, Experiment 2, which employed a smaller learning rate of 0.01 and extended training for 100 epochs, showed a modest improvement across all metrics, achieving an accuracy of 22.88%, a precision of 22.70%, and an F-score of 16.36%. Reduced learning rates allowed the MLP to converge more effectively, albeit the overall performance remains suboptimal.

7.2 Multi-label Classification Approaches

This section evaluates two common strategies for multi-label classification: **One-vs-One (1v1)** and **One-vs-All (1vA)**. These approaches decompose the multi-label problem into simpler binary classification tasks, enabling the use of standard binary classifiers like the Multi-Layer Perceptron (MLP) trained with Hinge loss.

	Exp. 1	Exp. 2
Learning Rate	0.1	0.01
Epochs	30	100
Accuracy	0.2062	0.2288
Precision	0.2255	0.2270
Recall	0.2062	0.2288
F-score	0.1399	0.1636

Table 4: Baseline results of the MLP with hinge loss function implementation. The multi-class strategy follows the one-vs-one approach with a voting system.

Strategy	LR	Lambda	Accuracy
1v1	0.001	0.01	0.3089
	0.001	0.001	0.3141
	0.0001	0.001	0.3957
1vA	0.001	0.001	0.1539
	0.001	0.001	0.2136
	0.0001	0.01	0.2310

Table 5: Performance of One-vs-One (1v1) and One-vs-All (1vA) strategies with varying learning rates and regularization parameters.

The One-vs-One (1v1) strategy involves creating a binary classifier for every pair of classes. Given C classes, this results in $\binom{C}{2}$ classifiers. Each classifier is trained to distinguish between a specific pair of classes. For prediction, the class that receives the majority of votes from the pairwise classifiers is selected. Formally, the predicted class y for a given input x can be expressed as:

$$y = \arg\max_i \sum_{j \neq i} f_{ij}(x)$$

where $f_{ij}(x)$ is the decision function of the classifier distinguishing class i from class j.

The One-vs-All (1vA) strategy trains C classifiers, where each classifier distinguishes a single class from all others. During inference, the class with the highest confidence score from its respective classifier is chosen:

$$y = \arg\max_i f_i(x)$$

where $f_i(x)$ is the decision function of the classifier for class i.

To assess the performance of these strategies, the conducted experiments include various learning rates (LR) and regularization strengths (λ). Additionally, it was noted experimentally that the Hinge loss stabilized relatively quickly, typically within 10–20 iterations, regardless of the strategy. An example of this can be observed in Figure 4.

The results, summarized in Table 5, indicate that the 1v1 strategy consistently outperforms 1vA. For example, with a learning rate of 0.0001 and a regularization parameter of 0.001, the

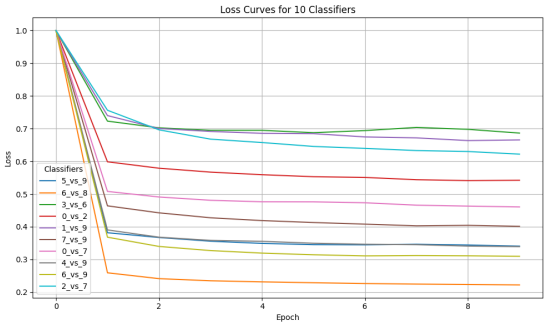


Figure 4: Loss function throughout ten iterations for ten randomly selected classifiers using the implemented from scratch SVM with linear kernel, learning rate 0.0001 and lambda parameter 0.01.

1v1 approach achieved an accuracy of 39.57%, whereas the best accuracy for 1vA under the same conditions was only 23.10%. This discrepancy highlights the strength of 1v1 in handling imbalanced or complex decision boundaries, likely due to its focus on pairwise comparisons which results in data balance across classes.

Another important observation is that the individual evaluation of the classifiers showed great performance (around 70% on average), especially in the 1vA strategy, but diminished accuracy in the general multi-label classification. This could indicate the models adapting to classify most samples as the 'all' class in the 1vA strategy, mainly due to the data imbalance. Choosing a different decision condition or voting system as well as downsampling in 1vA cases could potentially improve these results.

Based on these findings, the One-vs-One (1v1) strategy will be the default approach in all subsequent experiments discussed in this report, given its superior performance and reliability in the evaluations.

7.3 T-SNE visualization

To enhance visualization of the kernels implemented from scratch in the `svm.scratch` module, t-SNE (t-distributed Stochastic Neighbor Embedding) [8] is employed to gain insights into feature representations and class distributions learned by the different kernels implemented from scratch.

T-SNE is a non-linear dimensionality reduction technique commonly used to visualize high-dimensional data in two or three dimensions. It works by modeling pairwise similarities between data points in the high-dimensional space and preserving these relationships in the lower-dimensional embedding. T-SNE excels at revealing local clusters and patterns within the data, making it particularly useful for exploring class separability or structure in datasets. However, it is sensitive to hyperparameters such as perplexity

	LibLinear	Linear	Polynomial	RBF	Sigmoid
Accuracy	0.3756	0.3750	0.5246	0.5437	0.0828
Precision	0.3677	0.3769	0.5294	0.5425	0.0367
Recall	0.3756	0.3750	0.5246	0.5437	0.0828
F-score	0.3688	0.3748	0.5255	0.5421	0.0371

Table 6: Performance comparison of different SVM kernels using scikit-learn and LibLinear implementations. Results indicate the superior performance of non-linear kernels (Polynomial and RBF) compared to linear ones.

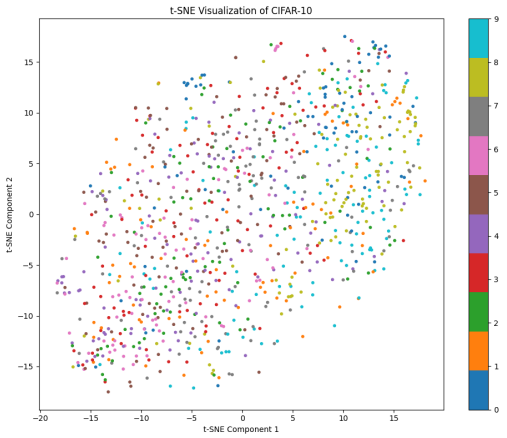


Figure 5: T-SNE visualization of downsampled CIFAR-10 dataset.

and learning rate, which can affect the resulting visualization.

In Figure 5 a 2-dimensional visualization of the whole CIFAR-10 dataset, downsampled to 1000 equally distributed samples for clarity, is represented with different colors indicating the 10 different classes. However, there are no apparent geometrical dependencies for the separation of the classes in this representation.

Figure 6 showcases the projected decision boundaries of the SVMs using different kernel functions (linear, polynomial, RBF and sigmoid) plotted over a t-SNE-reduced feature space. For the visualization, the classifier between labels 0 and 1 was chosen for simplicity, as all other classifiers had similar results with slightly fluctuating performance.

The first image illustrates the decision boundary with a linear kernel, which generates a linear hyperplane to separate the two classes. The boundary is rigid, and as evident from the graph, it struggles to adapt to the non-linear structure of the data, resulting in visible overlaps and misclassifications. The second image represents the polynomial kernel, which introduces non-linearity by incorporating polynomial combinations of the input features. This allows the decision boundary to better capture the underlying complexity of the data, evident from the smoother but intricate contours. Lastly, the RBF and Sigmoid kernel graph exhibit the most flexibility, producing highly non-linear boundaries. Their adaptability to local variations in the data is apparent from the tightly curved regions, which effectively separate the two classes in complex scenarios. These visualizations highlight the trade-off between kernel flexibility and generalization, with Sigmoid being the most expressive and linear being the most constrained.

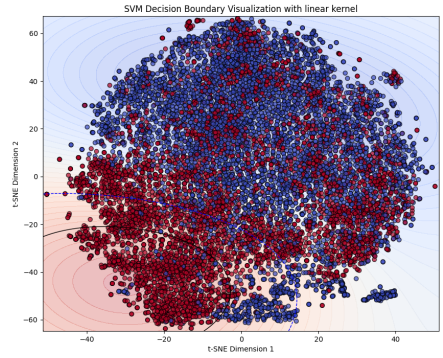
7.4 Kernel Comparison & Quadratic Programming

The kernels were implemented from scratch in the `svm.scratch` module. However, due to the time complexity of QP solvers and kernel computation, the tests were conducted on a fraction of the dataset. This resulted in suboptimal performance and limited insight into the models' capabilities.

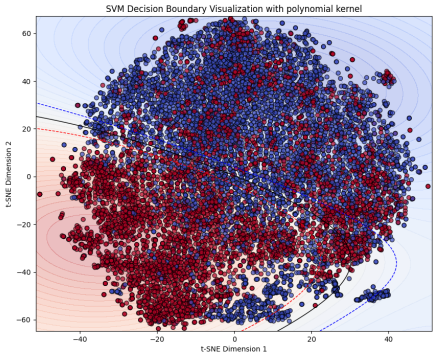
As a comparison, the library used was `scikit-learn` SVM implementation, which provided significantly better results due to its optimized methods and scalability. The results, presented in Table 6, highlight the performance of the kernels:

- The RBF kernel achieved the best performance with an accuracy of 54.37% and F-score of 54.21%.
- The Polynomial kernel performed slightly worse but still outperformed the Linear kernel, with an accuracy of 52.46%.
- The Linear kernel, using both `LibSVM` and `LibLinear`, achieved comparable accuracy around 37.5%.
- The Sigmoid kernel achieved the lowest score, despite the somewhat promising visualization in the previous paragraph.

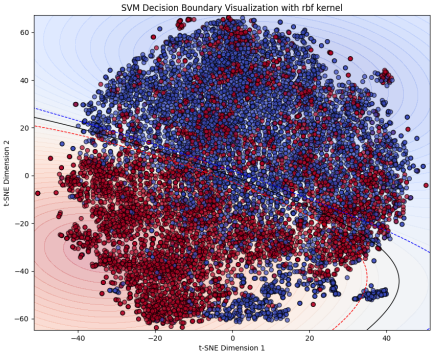
These findings confirm the suitability of non-linear kernels like RBF and Polynomial for capturing complex patterns in the data, while other models are more limited in this context. This model had the best overall performance for the CIFAR-10 dataset, with the confusion matrix of the results being shown in Figure 7.



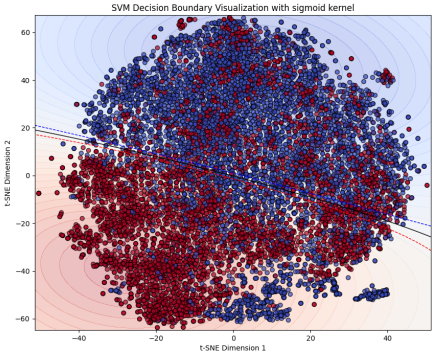
(a) Linear



(b) Polynomial (Degree 3)



(c) RBF



(d) Sigmoid

Figure 6: Visualization of the samples, decision boundary and maximum margins for the 0-vs-1 classifier through T-SNE for various kernel configurations.

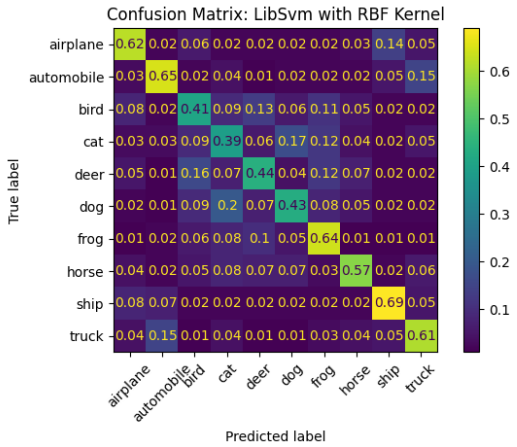


Figure 7: Confusion matrix of the best performing SVM model.

8 Conclusion

The exploration of Support Vector Machines (SVMs) for the CIFAR-10 classification task demonstrated the significant potential of non-linear kernels, particularly the radial basis function (RBF) and polynomial kernels, in capturing complex patterns inherent in high-dimensional data. Baseline models like k-nearest neighbors and nearest centroid provided a foundational understanding of the dataset's challenges, emphasizing the need for more sophisticated methods.

The scratch implementation of SVMs showcased an in-depth comprehension of fundamental principles, including kernel functions and quadratic programming, while benchmarking with scikit-learn highlighted the advantages of optimized frameworks. Multi-label classification strategies revealed the superiority of the One-vs-One approach in handling imbalanced classes and complex decision boundaries.

Finally, visualizations using techniques like t-SNE illuminated the interpretive nuances of kernel decision boundaries. These results underline the importance of model selection and hyperparameter tuning in achieving robust and efficient classification, setting the stage for further ad-

vancements in neural network-based methodologies.

Acknowledgments

Results presented in this work have been produced using the Aristotle University of Thessaloniki (AUTh) High Performance Computing Infrastructure and Resources. Additionally, the open-source language model ChatGPT [9] was utilized in parts of these experiments. It generated portions of the code which were then tested and analyzed, as well as enhanced the overall readability and clarity of this report.

References

[1] A. Palaska, "cifar10-nn." https://github.com/emily-palaska/cifar10-nn, 2024.

[2] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep., University of Toronto, 2009.

[3] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, "scikit-image: Image processing in python." https://doi.org/10.7717/peerj.453, 2014.

[4] Wallber, "Precision and recall." https://commons.wikimedia.org/w/index.php?curid=36926283, 2021.

[5] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, *et al.*, "Array programming with NumPy," 2020.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[7] M. S. Andersen, J. Dahl, and L. Vandenberghe, "Cvxopt: A python package for convex optimization." https://cvxopt.org, 2023.

[8] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[9] OpenAI, "Chatgpt." https://chat.openai.com/, 2024.