Support Vector Machines (SVMs)

Eyuel Nigussie

Support Vector Machines (SVMs)

Support Vector Machines (SVMs) are a powerful supervised machine learning algorithm used for classification and regression tasks. SVMs are designed to find the optimal boundary, called a **hyperplane**, that separates classes of data with the largest possible margin (distance to the nearest points). This ensures robust and accurate predictions on new data. They can handle both linearly separable and non-linearly separable data through a technique called the *kernel trick*.

Key Concepts of Support Vector Machines

- **Hyperplane:** A decision boundary separating different classes in feature space.
- **Support Vectors:** The closest data points to the hyperplane, crucial for determining the hyperplane and margin in SVM.
- **Margin:** The distance between the hyperplane and the support vectors. SVM aims to maximize this margin for better classification performance.
- **Kernel:** A function that maps data to a higher-dimensional space, enabling SVM to handle non-linearly separable data.
- **Hard Margin:** A maximum-margin hyperplane that perfectly separates the data without misclassifications.
- **Soft Margin:** Allows some misclassifications by introducing slack variables, balancing margin maximization and misclassification penalties when data is not perfectly separable.

Maximum Margin Classifier (Hard Margin) in SVMs

The Maximum Margin Classifier (MMC) is the simplest form of a Support Vector Machine (SVM) used for binary classification. It aims to find a hyperplane that perfectly separates two classes of data with the widest possible margin. MMC assumes the data is **linearly separable**, meaning there exists a hyperplane that can separate the classes without any misclassification.

Deriving the Margin Formula

1. Defining the Hyperplanes

In a linear Support Vector Machine (SVM), the decision boundary is a **hyperplane** defined by:

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

where:

- $\mathbf{w}$ is the weight vector (normal to the hyperplane),
- $\mathbf{x}$ is a data point,
- b is the bias term.

The two **parallel hyperplanes** that define the margin pass through the **support vectors** and are given by:

$$\mathbf{w} \cdot \mathbf{x} + b = 1$$

$$\mathbf{w} \cdot \mathbf{x} + b = -1$$

These are chosen so that the closest points to the decision boundary (the support vectors) satisfy these equalities. The SVM seeks to **maximize the distance** between these two hyperplanes.

2. Distance Between the Margin Hyperplanes

To find the distance between the two parallel hyperplanes, we:

1. Pick a point $\mathbf{x}_1$ on the first hyperplane:

$$\mathbf{w} \cdot \mathbf{x}_1 + b = 1$$

2. Measure its perpendicular distance to the second hyperplane:

$$\mathbf{w} \cdot \mathbf{x} + b = -1$$

3. General Distance Formula

The perpendicular distance from a point $\mathbf{x}_0$ to a hyperplane $\mathbf{w} \cdot \mathbf{x} + b_{\text{plane}} = 0$ is given by:

$$d = \frac{|\mathbf{w} \cdot \mathbf{x}_0 + b_{\text{plane}}|}{\|\mathbf{w}\|}$$

where $\|\mathbf{w}\|$ is the Euclidean norm of $\mathbf{w}$.

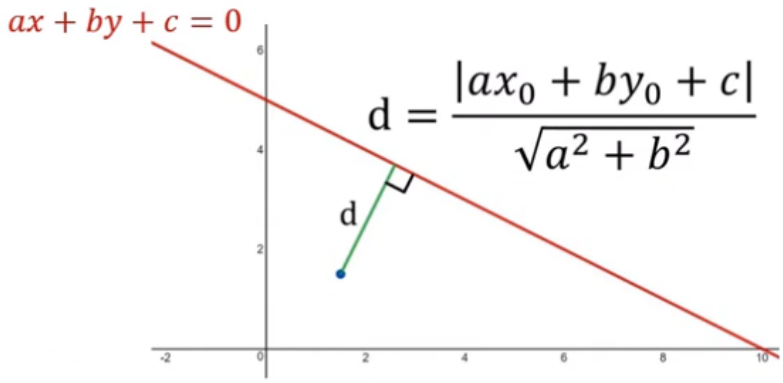

Figure 1: The Formula for the Distance from a Point to a Line

4. Applying the Formula

For our case:

- The point $\mathbf{x}_1$ lies on $\mathbf{w} \cdot \mathbf{x} + b = 1$.
- The second hyperplane is $\mathbf{w} \cdot \mathbf{x} + b = -1$, which can be rewritten as:

$$\mathbf{w} \cdot \mathbf{x} + (b + 1) = 0$$

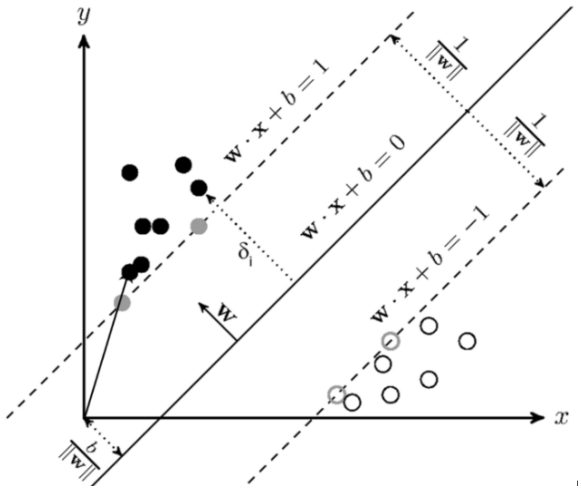

Figure 2: Illustration of a Support Vector Machine (SVM) Classifier and the Maximum Margin Hyperplane

Substituting into the distance formula:

$$d = \frac{|\mathbf{w} \cdot \mathbf{x}_1 + (b + 1)|}{\|\mathbf{w}\|}$$

Since $\mathbf{w} \cdot \mathbf{x}_1 + b = 1$, we have:

$$d = \frac{|1+1|}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$$

5. The Margin Formula

The total margin M between the two hyperplanes is:

$$M = \frac{2}{\|\mathbf{w}\|}$$

Maximizing the margin is equivalent to minimizing $\|\mathbf{w}\|$.

6. Hard Margin SVM Optimization Problem

The hard-margin SVM can be formulated as:

$$\min_{\mathbf{w},b} \ \frac{1}{2}\|\mathbf{w}\|^2$$

subject to:

$$y_i \left(\mathbf{w}^T \mathbf{x}_i + b\right) \geq 1, \quad \forall i$$

where $y_i \in \{-1, +1\}$ are the class labels.

Support Vector Classifier (Soft Margin) in SVMs

The **Support Vector Classifier (SVC)**, also called the *soft-margin SVM*, extends the **Maximum Margin Classifier (MMC)** for binary classification. While the MMC assumes perfectly linearly separable data, the SVC allows certain points to be inside the margin or even misclassified. This flexibility is crucial for handling noisy or overlapping datasets.

The SVC works by **maximizing the margin** while **penalizing classification errors**:

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

subject to:

$$y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i$$

where:

- ξ_i are **slack variables** allowing margin violations,
- $C > 0$ is the **regularization parameter**.

Slack Variables

The slack variables ξ_i describe margin violations:

$$\begin{cases} \xi_i = 0 & \text{Correctly classified, outside/on margin} \\ 0 < \xi_i \leq 1 & \text{Inside margin but correctly classified} \\ \xi_i > 1 & \text{Misclassified} \end{cases}$$

The C Parameter

The regularization parameter C controls the trade-off between margin width and classification accuracy:

$$\begin{cases} \text{Large } C: & \text{Less tolerance for violations, narrower margin (risk of overfitting)} \\ \text{Small } C: & \text{More tolerance for violations, wider margin (better generalization)} \end{cases}$$

SVC with Non-Linear Kernels

For datasets that are not linearly separable, the SVC uses the **kernel trick** to map data into a higher-dimensional space $\Phi(\mathbf{x})$ where a linear separator exists. In the original space, this results in a **non-linear decision boundary**.

Kernel Definition: Instead of explicitly computing $\Phi(\mathbf{x})$, we compute the dot product in the higher-dimensional space directly:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^T \Phi(\mathbf{x}_j)$$

The RBF Kernel

The **Radial Basis Function (RBF) kernel** is defined as:

$$K(\mathbf{x}, \mathbf{y}) = e^{-\gamma \|\mathbf{x}-\mathbf{y}\|^2}$$

where:

- If $\mathbf{x} = \mathbf{y}$, $K(\mathbf{x}, \mathbf{y}) = 1$.
- If $\mathbf{x}$ and $\mathbf{y}$ are far apart, $K(\mathbf{x}, \mathbf{y}) \to 0$.
- Small γ: Similarity decays slowly (wider influence).
- Large γ: Similarity decays quickly (localized influence).

The RBF kernel implicitly maps data into an infinite-dimensional feature space, which can be seen via its Taylor expansion. To understand this, we can expand the exponential function using the Taylor series:

$$e^z = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \dots$$

Substituting $z = -\gamma\|\mathbf{x} - \mathbf{y}\|^2$ from the RBF kernel, we get:

$$K(\mathbf{x}, \mathbf{y}) = 1 - \gamma\|\mathbf{x} - \mathbf{y}\|^2 + \frac{(\gamma\|\mathbf{x} - \mathbf{y}\|^2)^2}{2!} - \dots$$

This expansion shows that the RBF kernel corresponds to a dot product in an infinite-dimensional space, where the transformation $\Phi(\mathbf{x})$ represents the data in that space.

The Kernel Trick: In SVM optimization, we never need the explicit coordinates of points in the high-dimensional space. Instead, we only require their dot product:

$$\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

where $\Phi(\mathbf{x})$ is the transformation into the high-dimensional space, and $\cdot$ denotes the dot product. The kernel trick allows us to compute this dot product directly using the kernel function:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$$

This equals the dot product $\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ in the infinite-dimensional space without ever computing $\Phi(\mathbf{x})$ explicitly.

Why This Is Efficient: The kernel trick is computationally efficient because it:

- Avoids explicitly creating infinite-dimensional feature vectors.
- Works with a simple formula in the original feature space.
- Saves time (no massive computations) and memory (no huge datasets).

SVM Optimization Problem

Primal Form (Hard Margin):

$$\min_{\mathbf{w},b} \ \frac{1}{2}\|\mathbf{w}\|^2$$

subject to:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \quad \forall i$$

Dual Form: We introduce Lagrange multipliers $\alpha_i \geq 0$ and form:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1]$$

Setting derivatives to zero:

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \quad \Rightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

$$\frac{\partial L}{\partial b} = 0 \quad \Rightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Dual Optimization Problem:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

subject to:

$$\alpha_i \geq 0, \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

In the dual, the data only appears as dot products $(\mathbf{x}_i \cdot \mathbf{x}_j)$, so we can replace:

$$\mathbf{x}_i \cdot \mathbf{x}_j \longrightarrow K(\mathbf{x}_i, \mathbf{x}_j)$$

This allows non-linear decision boundaries without explicitly computing $\Phi(\mathbf{x})$.

Final Classifier in Dual Form

Once α is found:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right)$$

Only **support vectors** (where $\alpha_i > 0$) contribute to the sum.

Key Takeaway:

- Primal: Works in original feature space, explicitly uses $\mathbf{w}$ and b.
- Dual: Works with dot products, enables kernels, and focuses only on support vectors.

Overview of Implementation

The following implementation compares three approaches for MNIST digit classification:

1. **CNN trained end-to-end with a softmax classifier:** The CNN learns both feature extraction and classification in a single training process.
2. **SVM trained directly on raw pixel data:** The SVM attempts to classify digits without any feature learning, relying only on the original pixel values.
3. **CNN feature extraction + SVM classifier:** The CNN is used to learn useful features from the images, and these extracted features are then passed to an SVM for classification.

This setup allows evaluation of:

- Performance gain from deep learning feature extraction vs. direct classification.
- Whether combining CNN-learned features with SVM improves accuracy.

Main Steps Performed

1. Data Loading & Preprocessing

- Load the MNIST dataset.
- Normalize pixel values to $[0, 1]$.

2. CNN Model Training

- Train a CNN on MNIST with a softmax output layer.

3. SVM on Raw Pixels

- Flatten each 28×28 image into a 784-dimensional vector.
- Standardize features.
- Train an SVM with an RBF kernel directly on raw pixel values.

4. CNN as Feature Extractor + SVM Classifier

- Build a CNN without the final softmax layer to act as a feature extractor.
- Train the CNN on MNIST.
- Extract feature vectors from the CNN.
- Train an SVM on the extracted features.

5. Result Comparison

- Compare the classification accuracy of the three approaches.

Performance Comparison Summary

The following table presents the key performance metrics for the three models evaluated on the MNIST dataset.

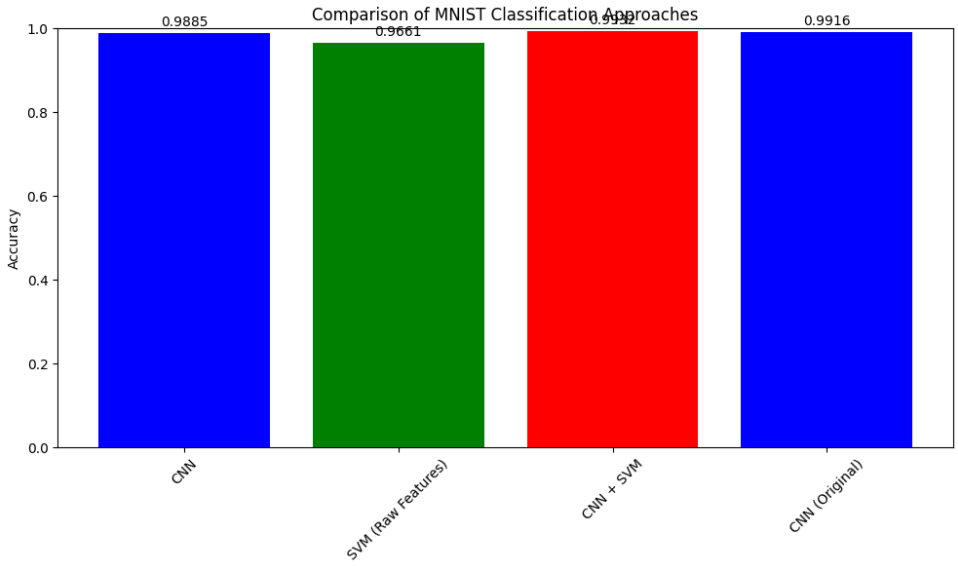

Figure 3: Accuracy comparison of CNN, SVM, and CNN+SVM on the MNIST dataset

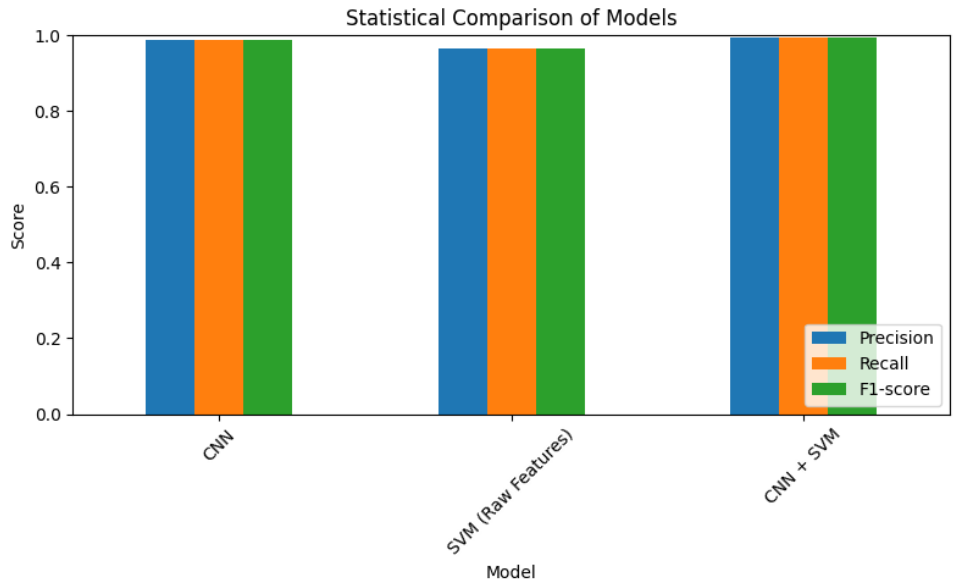

Figure 4: Score comparison of CNN, SVM, and CNN+SVM on the MNIST dataset

Model	Precision	Recall	F1-score
CNN Feature Extraction + SVM	0.9932	0.9932	0.9932
CNN (End-to-End)	0.9887	0.9885	0.9885
SVM (Raw Pixel Features)	0.9663	0.9661	0.9661

Discussion

The results clearly demonstrate the critical role of effective feature extraction in image classification tasks. The performance of the three models highlights a hierarchy of effectiveness:

Best Performer: CNN Feature Extraction + SVM This model achieved the highest scores. Its success stems from a powerful combination: the CNN layers act as a sophisticated feature extractor, learning hierarchical patterns and converting raw pixel data into a highly discriminative, lower-dimensional feature space. The SVM then excels at finding the optimal, max-margin decision boundary within this refined feature space, enhancing the model's generalization capabilities and leading to the best overall performance.

Strong Performer: CNN The standalone CNN also performed exceptionally well, showcasing the inherent power of convolutional neural networks to automatically learn features from images. Its high scores prove that it can effectively capture spatial dependencies and patterns without any manual feature engineering.

Weakest Performer: SVM (Raw Pixel Features) The SVM using raw pixel data performed significantly worse. This is because it lacks the ability to understand the spatial structure of the images. Raw pixel values are highly sensitive to variations in rotation, translation, and stroke thickness, which an SVM alone cannot overcome. This result underscores that a powerful classifier is ineffective without meaningful, well-extracted features.

Conclusion

This study compared three approaches to MNIST digit classification:

1. SVM trained on raw pixel data
2. CNN trained end-to-end
3. CNN feature extraction combined with an SVM classifier

The experiments demonstrate that effective feature extraction is crucial for achieving high accuracy in image classification tasks. While SVMs on raw pixel features struggle due to the lack of spatial understanding, CNNs excel at capturing hierarchical patterns directly from images. Combining CNN-based feature extraction with an SVM classifier yields the best overall performance, confirming that leveraging deep learning for feature representation and classical machine learning for classification can produce superior results.

Colab Notebook

For implementation and code, the Google Colab notebook is here
: **https://colab.research.google.com/drive/1VOUcMop9ZpP4fzSM0RaHpli8nPyR6YiL?us**