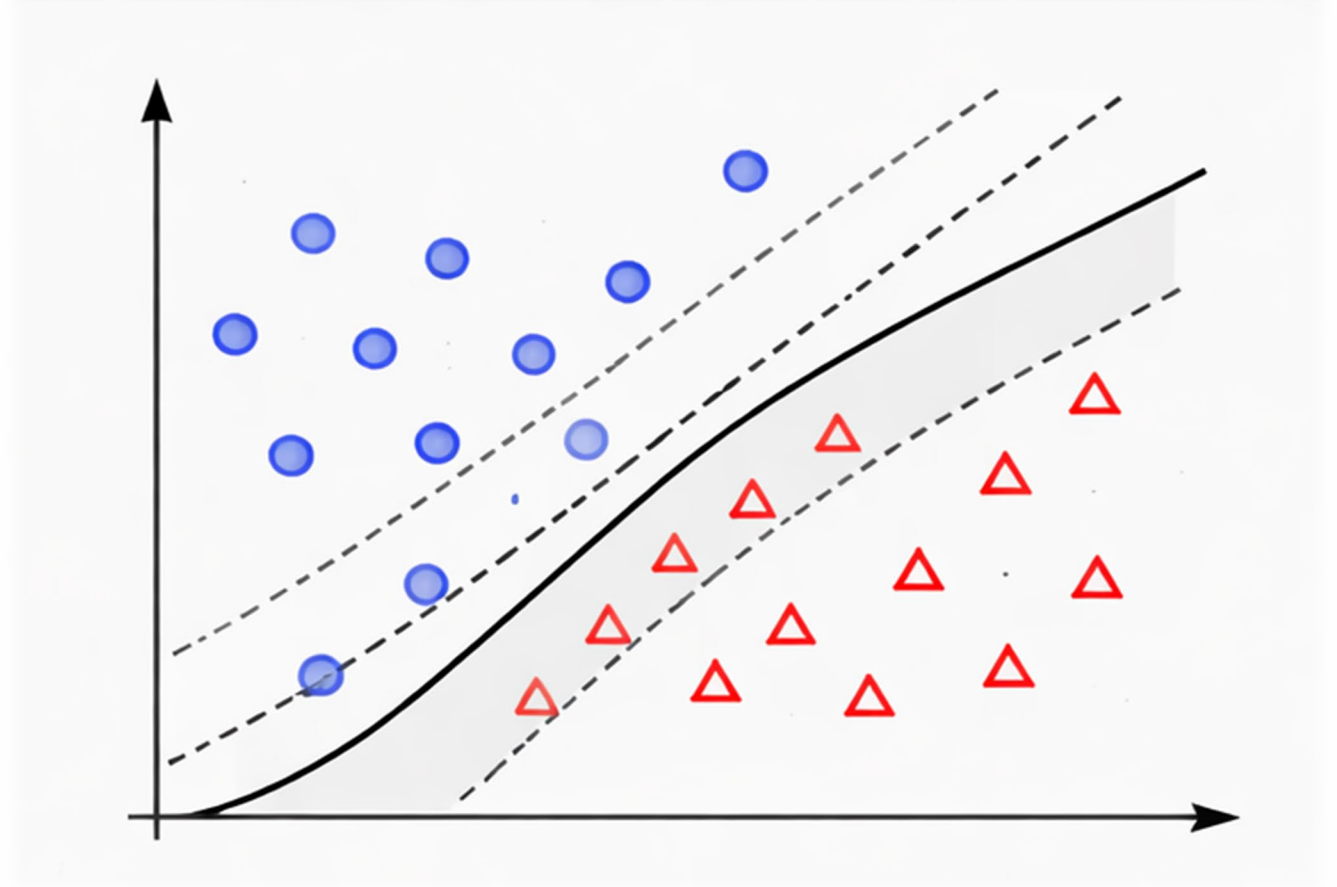

BINARY CLASSIFICATION OF HANDWRITTEN DIGITS USING SUPPORT VECTOR MACHINES

FARAH CHAUDHRY

COSC6349

12/19/2025

1. Introduction

Recognizing handwritten digits is a classic problem in computer vision and pattern recognition. While it is relatively easy for a human to distinguish between handwritten digits, creating a computer that can accurately classify handwritten digits requires a well-thought-out way to represent the digit data as images and use appropriate machine learning algorithms to learn how to classify the digit images accurately. The project presented in this paper focuses on a binary classification task where the goal is to classify handwritten numbers 2 and 3.

There are two main reasons for choosing this task. The first is that digits 2 and 3 share a lot of similar visual characteristics, including curves and strokes, making it a more difficult task to separate them from other digits than if we chose two other distinct digits. The second reason is that by limiting the problem to two classes of images, the behavior of the classifier can be studied in more detail, especially with respect to the decision boundary and classification error.

Support Vector Machines (SVMs) are well-suited for solving this problem because SVMs use the method of separating the classes of input data by creating a decision boundary that is maximized in terms of the distance between the two classes. Because image data can be numerically represented (through the pixel values of each image), SVM provides the right framework for separating classes of images based on their numerical features. In this paper, both linear SVM models and non-linear SVMs using Radial Basis Function (RBF) kernels are used to see how the complexity of the model affects the performance of classification.

2. Dataset Description

This project uses a dataset, the MNIST handwritten digits, which is one of the more common and widely used datasets in assessing image classification techniques. The MNIST dataset consists of a collection of grayscale images of handwritten numbers ranging from 0 to 9. Each image is characterized by its pixel intensity values, which represent the amount of brightness or darkness in the picture. These pixel intensity values can be converted into numerical representations (i.e., feature vectors), which makes it possible to process them using machine learning algorithms.

The dataset used comes from a public archive of the **MNIST dataset available on Kaggle**, where each of the training images and training labels, as well as test images, all have their own file. Only the images of the digit "2" and images of the digit "3" were chosen for use within this analysis, thus creating a binary classification problem. This simplification allows the performance of the classifier to be analyzed more clearly, as only two-digit classes are considered.

All pixels within each image were combined into a single one-dimensional array of pixel intensity values, with each pixel representing a separate feature. The dataset is divided into two sections. The training portion of the dataset is used to construct a model that learns to classify handwritten digits based on labeled examples. The trained model is then evaluated on a separate, previously unseen test dataset to assess its generalization performance.

3. Methodology

The application of supervised learning techniques is employed in this work to classify images of handwritten digits. Each number image can be numerically represented by taking the values of the pixel intensities (brightness) of each pixel, which enables mathematical processing by machine learning methods. Because SVMs calculate distances, it is critical that the input data has been pre-processed and features scaled specifically for the model training phase.

An MNIST image contains a 28 × 28 grid of grayscale pixels. Each grid of pixels is then converted into a single, one-dimensional array of 784-pixel values.

```
# Load MNIST data

train_images_path = "data/train-images.idx3-ubyte"
train_labels_path = "data/train-labels.idx1-ubyte"
test_images_path  = "data/t10k-images.idx3-ubyte"
test_labels_path  = "data/t10k-labels.idx1-ubyte"

X_train = load_mnist_images(train_images_path)
y_train = load_mnist_labels(train_labels_path)

X_test = load_mnist_images(test_images_path)
y_test = load_mnist_labels(test_labels_path)
```

```
# Filter digits 2 and 3

train_mask = (y_train == 2) | (y_train == 3)
test_mask  = (y_test == 2) | (y_test == 3)

X_train = X_train[train_mask]
y_train = y_train[train_mask]

X_test = X_test[test_mask]
y_test = y_test[test_mask]
```

Figure 1. MNIST data loading and preprocessing.
The dataset is loaded from IDX files, filtered to include only digits 2 and 3, and reshaped into 784-dimensional feature vectors.

The individual pixel values alone do not explicitly describe any shape or stroke and therefore do not provide enough information in and of themselves to allow a classifier to differentiate between digit classes; however, collectively, they form an identifiable pattern that can be analyzed by a classifier.

Feature scaling is performed on the pixel values after they have been identified as features.

```
# Scale features (very important for SVM)

scaler = StandardScaler()

# Fit ONLY on training data, then transform both
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Figure 2. Feature scaling using StandardScaler.
Scaling is applied to normalize pixel intensity values, ensuring fair distance computations during SVM training.

The goal of scaling is to allow pixel features to have a common numerical range so that the numeric value (magnitude) of a feature does not dominate the distance computations. The importance of the scaling procedure is greater for SVMs than for many other algorithms, because SVMs define decision boundaries based upon the distances between data points in the feature space.

In the current research project, two distinct SVM Models were applied. The first SVM Model is a Linear Kernel Model (i.e., a linear classifier) which attempts to separate the digit classes using a Linear Decision Boundary (LDB). This is used as a baseline model, providing insight into the degree to which each class of digits may be separated using a simple boundary.

Conversely, the second SVM Model, a RBF Kernel Model, is capable of creating a Non-Linear Decision Boundary (NDB); therefore, the RBF Kernel Model is able to capture the more complex interrelationships found in the data by constructing a new

feature set by implicitly mapping the original input features into a higher dimensional Space.

Both SVM Models are trained and evaluated on the same Training Dataset and Testing Dataset for purposes of fair comparison. The Performance of both SVM Models in Terms of Correctly Classifying Digit Classes was assessed by using two methodologies: 1) using Classification Accuracy, and 2) Creating Confusion Matrices. The use of a Classification Accuracy score will indicate the overall performance of each SVM Model, while the production of Confusion Matrices will provide more specific information regarding the types of classification errors experienced by each SVM Model.

4. Experimental Setup

The dataset was pre-processed and feature-scaled, then split into two subsets, training and testing, before training any of the Support Vector Machine models. The training set contains handwritten images of digits labeled “2” or “3” and is used exclusively for training the Support Vector Machine classifiers. The test set is an unseen dataset and is only used to evaluate the performance of the classifiers built using the training data.

To ensure a fair comparison between the two Support Vector Machine classifiers, both were trained using identical datasets and pre-processing techniques. The first classifier uses a linear kernel, which attempts to separate the two-digit classes along a straight line (decision boundary) in the feature space. The second classifier uses the radial basis function (RBF) kernel, which has a non-linear boundary (decided boundary) and greater flexibility. The only difference in model parameters (excluding kernel types) among the two classifiers was used in this study to ensure that any difference in classifier performance would be based on model complexity rather than differences in experimental setup or methodology.

The same feature scaling was applied to both the training and the test datasets using the same scaling parameters obtained during the training data preparation process. This process prevents leakage of test dataset information into the results of the training classifiers, thereby maintaining the dimensionality of both the training and test datasets.

5. Results

Classification accuracy and confusion matrices were used to evaluate the performance of the models used in this study.

```
Model training complete.

--- Linear SVM Results ---
Accuracy: 0.9740450538687562
Confusion Matrix:
[[1001   31]
 [  22  988]]
```

```
--- RBF SVM Results ---
Accuracy: 0.9897159647404505
Confusion Matrix:
[[1025    7]
 [  14  996]]
```

Figure 3. Linear SVM classification results

The linear SVM achieved an accuracy of 97.4%, with most misclassifications occurring near ambiguous digit boundaries.

Figure 4. RBF SVM classification results.

The RBF kernel improved accuracy to 98.9%, reducing misclassification errors compared to the linear model.

Classification accuracy is the overall percentage of accurately classified images. Confusion matrices provide a detailed view of how well each digit is classified.

The linear SVM model performed well with respect to classification accuracy. The results indicate that digits '2' and '3' could be separated somewhat using a linear decision boundary. However, some images of handwritten digits had misclassifications, especially when they included ambiguous strokes or forms.

Although classification accuracy for the SVM with the RBF kernel was greater than for the linear model, this improvement indicates that the relationship between pixel values and digit labels is not strictly linear. The nonlinear decision boundary provided by the RBF kernel is more adaptable to variations in handwriting style, thus improving classification accuracy.

The confusion matrices indicated a decrease in the number of misclassified digits for the RBF model as compared to the linear model, thereby demonstrating relative

equal performance of the RBF and SVM with respect to each digit class. Therefore, more flexible models increase classification accuracy with complex images. Complete implementation details are available in the project repository.

6. Discussion

This project demonstrates the importance of selecting the right model when analyzing images. Even though it was fairly accurate at classifying '2' and '3', classification accuracy suffered due to the assumption that data would be situated along a specific linear boundary, as variability exists in the handwritten digits in terms of style, thickness, curvature, and many other factors, making it impossible to draw a linear boundary separating all possibilities.

The RBF SVM performed better than the linear SVM, indicating that the relationship between pixel values and classes is nonlinear in nature. An RBF kernel allows the model to have greater flexibility with its decision boundary and thus enables it to classify handwritten digits more accurately by capturing more subtle differences in handwriting patterns, resulting in more correct classifications. This highlights the advantages of using more complex learning models for high-dimensional image datasets.

Even though these findings suggest better ways of solving the posed problem, there are still limitations associated to the approach chosen for this project. Flattening images into a one-dimensional array of vector values eliminates the spatial relationships between pixels, such as edges or local structures, which may result in less-than-optimal classification. Additionally, the limitation of only two digits in the dataset allows for simplification of the situation and does not represent the actual digit recognition problem you would need to contend with when trying to recognize letters across all ten classifications. In future works, it may be worth looking into feature extraction methods or creating convolutional models that maintain spatial relationships between image pixels.

7. Conclusion

The machine learning technique employed in this concept was Support Vector Machines (SVMs) to classify the handwritten digit information from the MNIST database. By restricting the digits to simply "2" and "3", the problem can be defined

as a classification problem, which enables a better understanding of how the Support Vector Machine performs within these specific ranges. The linear and nonlinear methods of classification ability were tested to provide insight into how the complexity of the decision boundary can affect the performance of classification.

This research has demonstrated that a Support Vector Machine's linear solution is capable of producing a reasonable predictive power; however, using nonlinear SVMs (such as the radial basis function) is more effective for the classification of complex images. Feature scaling is an important component of ensuring accurate distance measurement and, thus, has considerable significance when using Support Vector Machines; therefore, it is necessary to finish this project with an illustration of how classical machine learning techniques apply to image classification problems when combined with proper preprocessing and method selection.

8. Project Repository

The complete implementation, including source code, dataset loading utilities, and experimental results, is publicly available on GitHub:

GitHub Repository:

https://github.com/fch98/cosc6349-final-svm-mnist

This repository contains the full Python implementation, experiment configuration, and output metrics used in this study, allowing for reproducibility and further experimentation.

9. References

- LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324.
- Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273–297.
- Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830.
- Kaggle. MNIST handwritten digit database. https://www.kaggle.com/datasets/oddrationale/mnist-in-csv

Complete Code:

```
"""
Handwritten Digit Classification using Support Vector Machines (SVM)

This script trains and evaluates two SVM models (Linear and RBF)
to classify handwritten digits 2 and 3 from the MNIST dataset.
"""
import numpy as np
import struct

from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix
def load_mnist_images(filepath):
    """
    Load MNIST image data from an IDX file and return it as a NumPy array.
    Each image is flattened into a 1D vector.
    """
    with open(filepath, 'rb') as f:
        magic, num_images, rows, cols = struct.unpack(">IIII", f.read(16))
        images = np.frombuffer(f.read(), dtype=np.uint8)
        images = images.reshape(num_images, rows * cols)
    return images
def load_mnist_labels(filepath):
```

```
    """
    Load MNIST label data from an IDX file and return it as a NumPy array.
    """
    with open(filepath, 'rb') as f:
        magic, num_labels = struct.unpack(">II", f.read(8))
        labels = np.frombuffer(f.read(), dtype=np.uint8)
    return labels

# Load MNIST data

train_images_path = "data/train-images.idx3-ubyte"
train_labels_path = "data/train-labels.idx1-ubyte"
test_images_path  = "data/t10k-images.idx3-ubyte"
test_labels_path  = "data/t10k-labels.idx1-ubyte"

X_train = load_mnist_images(train_images_path)
y_train = load_mnist_labels(train_labels_path)

X_test = load_mnist_images(test_images_path)
y_test = load_mnist_labels(test_labels_path)

print("Training data shape:", X_train.shape)
print("Training labels shape:", y_train.shape)
print("Test data shape:", X_test.shape)
```

```
print("Test labels shape:", y_test.shape)

# Filter digits 2 and 3

train_mask = (y_train == 2) | (y_train == 3)
test_mask  = (y_test == 2) | (y_test == 3)

X_train = X_train[train_mask]
y_train = y_train[train_mask]

X_test = X_test[test_mask]
y_test = y_test[test_mask]

print("Filtered training shape:", X_train.shape)
print("Filtered test shape:", X_test.shape)
print("Unique training labels:", np.unique(y_train))
print("Unique test labels:", np.unique(y_test))

# Convert labels to binary: 2 -> 0, 3 -> 1
y_train = (y_train == 3).astype(int)
y_test  = (y_test == 3).astype(int)

print("After binary conversion - unique y_train:", np.unique(y_train))
print("After binary conversion - unique y_test:", np.unique(y_test))
```

```
# Scale features (very important for SVM)

scaler = StandardScaler()

# Fit ONLY on training data, then transform both
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("Scaling complete.")
print("Scaled training shape:", X_train_scaled.shape)
print("Scaled test shape:", X_test_scaled.shape)

# Train SVM models

# 1) Linear SVM (baseline)
linear_svm = SVC(kernel="linear", C=1.0)
linear_svm.fit(X_train_scaled, y_train)

# 2) RBF SVM (nonlinear)
rbf_svm = SVC(kernel="rbf", C=1.0, gamma="scale")
rbf_svm.fit(X_train_scaled, y_train)

print("Model training complete.")
```

```
# Evaluate models

def evaluate_model(model, X, y, model_name="model"):
    preds = model.predict(X)
    acc = accuracy_score(y, preds)
    cm = confusion_matrix(y, preds)

    print(f"\n--- {model_name} Results ---")
    print("Accuracy:", acc)
    print("Confusion Matrix:")
    print(cm)

    return acc, cm

linear_acc, linear_cm = evaluate_model(linear_svm, X_test_scaled, y_test, "Linear SVM")
rbf_acc, rbf_cm = evaluate_model(rbf_svm, X_test_scaled, y_test, "RBF SVM")

# Save results (connects to report)

import os

os.makedirs("results", exist_ok=True)
```

```
with open("results/metrics.txt", "w") as f:
    f.write("MNIST Digits 2 vs 3 (Binary: 2->0, 3->1)\n\n")

    f.write("Linear SVM\n")
    f.write(f"Accuracy: {linear_acc}\n")
    f.write("Confusion Matrix:\n")
    f.write(str(linear_cm))
    f.write("\n\n")

    f.write("RBF SVM\n")
    f.write(f"Accuracy: {rbf_acc}\n")
    f.write("Confusion Matrix:\n")
    f.write(str(rbf_cm))
    f.write("\n")

print("\nSaved results to: results/metrics.txt")
```