Support Vector Machines: Mathematical Foundation and Implementation

DSC 255: Machine Learning Fundamentals

May 10, 2025

1 Introduction

Support Vector Machines (SVMs) are powerful supervised learning models used for classification and regression tasks. Developed by Vladimir Vapnik and colleagues in the 1990s, SVMs have become one of the most robust and versatile machine learning algorithms. This document provides a comprehensive explanation of SVMs, covering their mathematical foundations, optimization techniques, implementation details, and practical applications.

2 Mathematical Foundation

2.1 Linear Classification and Maximum Margin

The core idea behind SVMs is to find the optimal hyperplane that separates different classes of data points with the maximum margin. For a binary classification problem in a d-dimensional feature space, this hyperplane is defined by:

$$w \cdot x + b = 0 \tag{1}$$

where:

- $w \in \mathbb{R}^d$ is the weight vector perpendicular to the hyperplane
- $b \in \mathbb{R}$ is the bias term
- $x \in \mathbb{R}^d$ is a data point in the feature space

The classification rule is given by:

$$f(x) = \text{sign}(w \cdot x + b) = \begin{cases} +1 & \text{if } w \cdot x + b \geq 0 \\ -1 & \text{if } w \cdot x + b < 0 \end{cases} \tag{2}$$

2.2 Margin and Support Vectors

The margin of an SVM is defined as the perpendicular distance between the decision boundary and the closest data points from each class. These closest points are called support vectors.

For a linearly separable dataset, we can scale w and b such that the functional margin of the support vectors is 1:

$$y_i(w \cdot x_i + b) = 1 \tag{3}$$

for the support vectors x_i. The geometric margin is then given by $\frac{1}{\|w\|}$. To maximize this margin, we need to minimize $\|w\|$, or equivalently, minimize $\frac{1}{2}\|w\|^2$.

3 Hard-Margin SVM

3.1 Optimization Problem

For linearly separable data, the hard-margin SVM optimization problem is formulated as:

$$\min_{w,b} \frac{1}{2}\|w\|^2 \tag{4}$$

$$\text{subject to } y_i(w \cdot x_i + b) \geq 1, \quad \forall i \in \{1, 2, \ldots, n\} \tag{5}$$

This is a convex quadratic programming problem with linear constraints.

3.2 Lagrangian Formulation

To solve this optimization problem, we introduce Lagrange multipliers $\alpha_i \geq 0$ for each constraint:

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(w \cdot x_i + b) - 1] \tag{6}$$

Taking partial derivatives with respect to w and b and setting them to zero:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \tag{7}$$

$$\Rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{8}$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{9}$$

$$\Rightarrow \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{10}$$

3.3 Dual Problem

Substituting these back into the Lagrangian, we get the dual optimization problem:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \tag{11}$$

$$\text{subject to } \alpha_i \geq 0, \quad \forall i \in \{1, 2, \ldots, n\} \tag{12}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{13}$$

3.4 Karush-Kuhn-Tucker (KKT) Conditions

The KKT conditions provide necessary and sufficient conditions for optimality:

$$\alpha_i [y_i(w \cdot x_i + b) - 1] = 0, \quad \forall i \in \{1, 2, \ldots, n\} \tag{14}$$

$$\alpha_i \geq 0, \quad \forall i \in \{1, 2, \ldots, n\} \tag{15}$$

$$y_i(w \cdot x_i + b) - 1 \geq 0, \quad \forall i \in \{1, 2, \ldots, n\} \tag{16}$$

From the first condition, we can see that either $\alpha_i = 0$ or $y_i(w \cdot x_i + b) = 1$. The points with $\alpha_i > 0$ are the support vectors, which lie exactly on the margin boundaries.

4 Soft-Margin SVM

4.1 Handling Non-Linearly Separable Data

In practice, data is often not linearly separable. To handle such cases, we introduce slack variables $\xi_i \geq 0$ that allow for some misclassification:

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \forall i \in \{1, 2, \ldots, n\} \tag{17}$$

4.2 Optimization Problem

The soft-margin SVM optimization problem becomes:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \tag{18}$$

$$\text{subject to } y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \forall i \in \{1, 2, \ldots, n\} \tag{19}$$

$$\xi_i \geq 0, \quad \forall i \in \{1, 2, \ldots, n\} \tag{20}$$

where $C > 0$ is a regularization parameter that controls the trade-off between maximizing the margin and minimizing the classification error.

4.3 Dual Problem

The dual problem for soft-margin SVM is similar to the hard-margin case, but with an additional constraint:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) \tag{21}$$

$$\text{subject to } 0 \leq \alpha_i \leq C, \quad \forall i \in \{1, 2, \ldots, n\} \tag{22}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{23}$$

4.4 Effect of the Regularization Parameter C

The parameter C controls the trade-off between maximizing the margin and minimizing the classification error:

- Large C: The optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of classifying all training points correctly. This can lead to overfitting.
- Small C: The optimization will look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. This can lead to underfitting.

5 Kernel Methods

5.1 The Kernel Trick

SVMs can be extended to handle non-linear decision boundaries by using the kernel trick. The idea is to map the original feature space to a higher-dimensional space where the data becomes linearly separable:

$$\Phi : \mathbb{R}^d \to \mathbb{R}^D, \quad D > d \tag{24}$$

Instead of computing this mapping explicitly, we use a kernel function that computes the inner product in the transformed space:

$$K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) \tag{25}$$

5.2 Common Kernel Functions

Some commonly used kernel functions include:

- Linear kernel: $K(x_i, x_j) = x_i \cdot x_j$
- Polynomial kernel: $K(x_i, x_j) = (x_i \cdot x_j + c)^d$
- Radial Basis Function (RBF) kernel: $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$
- Sigmoid kernel: $K(x_i, x_j) = \tanh(\kappa x_i \cdot x_j + c)$

5.3 Dual Problem with Kernels

The dual problem with kernels becomes:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \tag{26}$$

$$\text{subject to } 0 \leq \alpha_i \leq C, \quad \forall i \in \{1, 2, \ldots, n\} \tag{27}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{28}$$

6 Implementation in Python

6.1 Using scikit-learn

The scikit-learn library provides a convenient implementation of SVMs through the `SVC` class:

```python
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = datasets.load_iris()
X = iris.data[:, [0, 2]]  # Features 0 and 2
y = iris.target
mask = (y == 1) | (y == 2)  # Labels 1 and 2
X = X[mask]
y = y[mask]

# Train an SVM with linear kernel
svm = SVC(kernel='linear', C=1.0)
svm.fit(X, y)

# Predict on the training data
y_pred = svm.predict(X)
accuracy = accuracy_score(y, y_pred)
print(f"Training accuracy: {accuracy:.4f}")

# Get the number of support vectors
n_support_vectors = svm.n_support_.sum()
print(f"Number of support vectors: {n_support_vectors}")
```

Listing 1: SVM Implementation with scikit-learn

6.2 Visualizing the Decision Boundary

For two-dimensional data, we can visualize the decision boundary:

```python
def plot_decision_boundary(X, y, svm):
    # Create a mesh grid
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                         np.arange(y_min, y_max, 0.02))

    # Predict on the mesh grid
    Z = svm.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # Plot the decision boundary
    plt.contourf(xx, yy, Z, alpha=0.3)

    # Plot the data points
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k')

    # Highlight support vectors
    plt.scatter(svm.support_vectors_[:, 0], svm.support_vectors_[:, 1],
                s=100, linewidth=1, facecolors='none', edgecolors='k')

    plt.xlabel('Feature 0')
    plt.ylabel('Feature 2')
    plt.title('SVM Decision Boundary')
    plt.show()

# Visualize the decision boundary
plot_decision_boundary(X, y, svm)
```

Listing 2: Visualizing the SVM Decision Boundary

7 Analyzing the Effect of the Regularization Parameter C

7.1 Grid Search for Optimal C

We can perform a grid search to find the optimal value of C:

```python
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000]}

# Create a grid search object
grid_search = GridSearchCV(SVC(kernel='linear'), param_grid, cv=5)
grid_search.fit(X, y)

# Print the best parameter
print(f"Best C: {grid_search.best_params_['C']}")
print(f"Best cross-validation score: {grid_search.best_score_:.4f}")
```

Listing 3: Grid Search for Optimal C

7.2 Comparing Different Values of C

We can also manually compare different values of C:

```python
c_values = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
results = []

for c in c_values:
    svm = SVC(kernel='linear', C=c)
    svm.fit(X, y)

    # Predict on training data
    y_pred = svm.predict(X)
    training_error = 1 - accuracy_score(y, y_pred)

    # Get number of support vectors
    n_support_vectors = svm.n_support_.sum()

    results.append({
        'C': c,
        'training_error': training_error,
        'n_support_vectors': n_support_vectors
    })

# Print results
print("\n{:<10} {:<20} {:<20}".format('C', 'Training Error', 'Support Vectors'))
print("-" * 50)
for result in results:
    print("{:<10.3f} {:<20.4f} {:<20}".format(
        result['C'],
        result['training_error'],
        result['n_support_vectors']
    ))
```

Listing 4: Comparing Different Values of C

8 Limitations and Extensions

8.1 Limitations

SVMs have several limitations:

- They do not directly provide probability estimates.
- They can be memory-intensive and computationally expensive for large datasets.
- The choice of kernel and regularization parameters can significantly affect performance.
- They are sensitive to the choice of kernel parameters.

8.2 Extensions

Several extensions have been proposed to address these limitations:

- Probabilistic SVMs: Use techniques like Platt scaling to obtain probability estimates.
- Multi-class SVMs: Extend binary SVMs to handle multiple classes using one-vs-one or one-vs-rest strategies.
- Structured SVMs: Handle structured output spaces.
- Online SVMs: Process data in an online manner for large-scale learning.

9 Conclusion

Support Vector Machines are powerful and versatile machine learning algorithms with strong theoretical foundations. They excel in high-dimensional spaces and are effective even when the number of dimensions exceeds the number of samples. By understanding the mathematical principles behind SVMs and their implementation details, practitioners can effectively apply them to a wide range of classification and regression tasks.

References

[1] Vapnik, V. N. (1995). The nature of statistical learning theory. Springer.

[2] Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine learning, 20(3), 273-297.

[3] Schölkopf, B., & Smola, A. J. (2002). Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press.