Supervised Learning: Support Vector Machines

Linear and Non-linear SVM, Kernel Trick, Margin Optimization

Sarwan Ali

Department of Computer Science
Georgia State University

Understanding Support Vector Machines

Today's Learning Journey

What are Support Vector Machines?

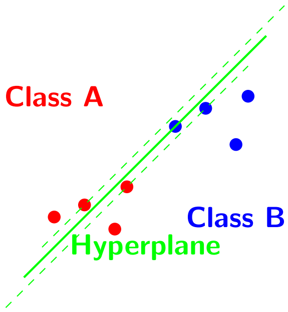
Support Vector Machines (SVMs) are powerful supervised learning algorithms for:

- Classification (primary use)
- Regression (SVR)

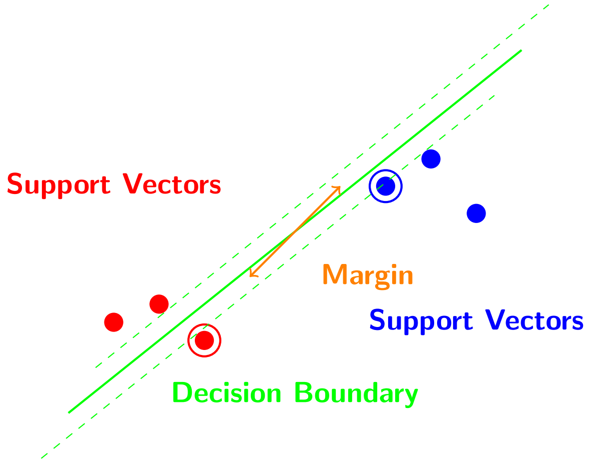
Key Idea: Find the optimal hyperplane that separates classes with maximum margin

Why SVMs?

- Effective in high-dimensional spaces
- Memory efficient
- Versatile (different kernel functions)



The Margin Concept



Margin: Distance between the decision boundary and the closest data points from each class
Support Vectors: The data points that lie closest to the decision boundary

Linear SVM: Mathematical Formulation

Goal: Find hyperplane $\mathbf{w}^T\mathbf{x} + b = 0$ that maximizes the margin

Distance from point to hyperplane:

$$d = \frac{|\mathbf{w}^T\mathbf{x} + b|}{||\mathbf{w}||}$$

For linearly separable data:

- Class +1: $\mathbf{w}^T\mathbf{x}_i + b \geq +1$
- Class -1: $\mathbf{w}^T\mathbf{x}_i + b \leq -1$

Margin width: $\frac{2}{||\mathbf{w}||}$

Optimization Problem:

$$\text{maximize} \quad \frac{2}{||\mathbf{w}||} \tag{1}$$

$$\text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad \forall i \tag{2}$$

Primal Optimization Problem

Equivalent formulation (easier to solve):

$$\text{minimize} \quad \frac{1}{2}||\mathbf{w}||^2 \tag{3}$$

$$\text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad \forall i \tag{4}$$

For non-separable data (Soft Margin):

$$\text{minimize} \quad \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i \tag{5}$$

$$\text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \tag{6}$$

$$\xi_i \geq 0, \quad \forall i \tag{7}$$

Where:

- ξ_i are slack variables (allow misclassification)
- C is the regularization parameter (trade-off between margin and errors)

Dual Optimization Problem

Using Lagrange multipliers, we get the dual form:

$$\text{maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \tag{8}$$

$$\text{subject to} \quad \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{9}$$

$$0 \leq \alpha_i \leq C, \quad \forall i \tag{10}$$

Solution:

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

Decision function:

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

Key insight: Only support vectors have $\alpha_i > 0$

Motivation for Kernels

Problem: What if data is not linearly separable?

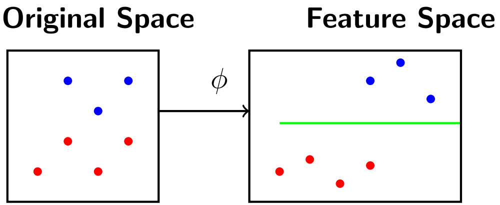
Solution: Map data to higher-dimensional space where it becomes linearly separable

Mapping function:

$$\phi : \mathbb{R}^d \to \mathbb{R}^D$$

where $D >> d$

Problem: Computing $\phi(\mathbf{x})$ explicitly can be expensive or impossible



The Kernel Trick

Key Insight: In the dual formulation, we only need dot products $\mathbf{x}_i^T \mathbf{x}_j$

Kernel Function:

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

Dual form with kernels:

$$\text{maximize} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{11}$$

Decision function:

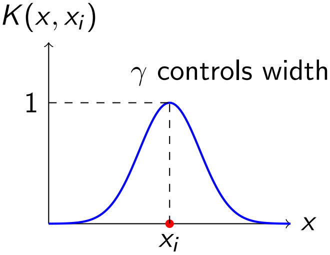
$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \right)$$

Advantage: We can work in infinite-dimensional spaces without explicitly computing $\phi(\mathbf{x})$!

Common Kernel Functions

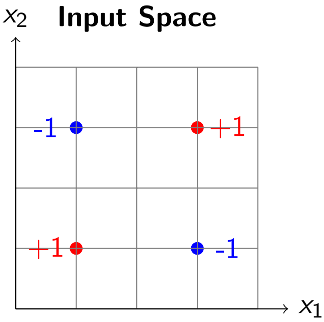
Kernel	Formula	Use Case
Linear	$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$	Linearly separable data
Polynomial	$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + 1)^d$	Polynomial decision boundaries
RBF (Gaussian)	$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\|\mathbf{x}_i - \mathbf{x}_j\|\|^2\right)$	Complex, non-linear patterns
Sigmoid	$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\alpha \mathbf{x}_i^T \mathbf{x}_j + c)$	Neural network-like

RBF Kernel Visualization:

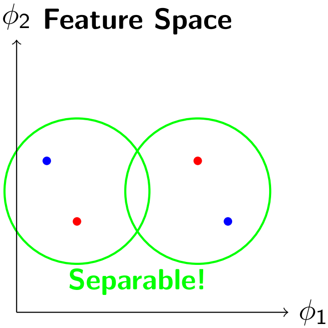


Non-linear SVM Example

XOR Problem: Not linearly separable in 2D



After RBF Kernel transformation:



Key Points:

- RBF kernel creates local decision boundaries
- Each support vector creates a "bump" in feature space
- Final decision boundary is combination of all bumps

Hyperparameter Tuning

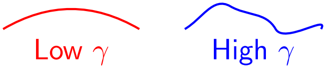
Key Hyperparameters:

1. Regularization Parameter (C):

- Small C: Wider margin, more misclassification (underfitting)
- Large C: Narrower margin, less misclassification (overfitting)

2. RBF Kernel Parameter (γ):

- Small γ: Smooth decision boundary (underfitting)
- Large γ: Complex decision boundary (overfitting)

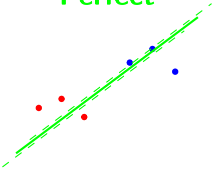


Hard Margin vs Soft Margin

Hard Margin SVM:

- No misclassification allowed
- Only works for linearly separable data
- Optimization: $\min \frac{1}{2}||\mathbf{w}||^2$
- Constraint: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

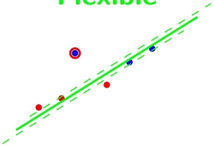
Perfect



Soft Margin SVM:

- Allows some misclassification
- Works for non-separable data
- Optimization: $\min \frac{1}{2}||\mathbf{w}||^2 + C\sum \xi_i$
- Constraint: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$

Flexible



Understanding the Margin

Geometric Margin:

$$\gamma_i = \frac{y_i(\mathbf{w}^T \mathbf{x}_i + b)}{||\mathbf{w}||}$$

Functional Margin:

$$\hat{\gamma}_i = y_i(\mathbf{w}^T \mathbf{x}_i + b)$$

Margin Optimization Strategy:

1. Maximize the minimum margin: $\max \min_i \gamma_i$
2. Normalize so that minimum functional margin $= 1$
3. This leads to: $\min ||\mathbf{w}||^2$ subject to $y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1$

Why maximize margin?

- Better generalization (statistical learning theory)
- Unique solution
- Robust to small perturbations

SVM Algorithm Summary

Training Phase:

1. Choose kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$
2. Solve dual optimization problem to find α_i
3. Identify support vectors (where $\alpha_i > 0$)
4. Calculate bias term b using support vectors

Prediction Phase:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{\text{support vectors}} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right)$$

Computational Complexity:

- Training: $O(n^2)$ to $O(n^3)$ depending on solver
- Prediction: $O(n_{sv} \cdot d)$ where n_{sv} is number of support vectors
- Memory: $O(n_{sv})$ (only need to store support vectors)

Advantages and Disadvantages

Advantages:

- Effective in high dimensions
- Memory efficient
- Versatile (different kernels)
- Global optimum guaranteed
- Works well with small datasets
- Good generalization

Disadvantages:

- Slow on large datasets
- Sensitive to feature scaling
- No probabilistic output
- Choice of kernel and parameters
- Poor performance on noisy data
- Doesn't handle missing values

When to use SVMs:

- High-dimensional data (text classification, gene analysis)
- Small to medium datasets
- Clear margin of separation exists
- Need for interpretable support vectors

Implementation Tips

Data Preprocessing:

- Scale features: Use StandardScaler or MinMaxScaler
- Handle missing values: Impute or remove
- Feature selection: Remove irrelevant features

Hyperparameter Tuning:

- Use Grid Search or Random Search
- Cross-validation for model selection
- Start with RBF kernel, then try others
- Typical ranges: $C \in [0.1, 1, 10, 100]$, $\gamma \in [0.001, 0.01, 0.1, 1]$

Python Libraries:

- scikit-learn: `SVC` for classification, `SVR` for regression
- `LIBSVM`: Fast C++ implementation with Python bindings
- `sklearn.model_selection`: For hyperparameter tuning

Python Implementation Example

```
from sklearn import datasets
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score
# Load dataset
X, y = datasets.make_classification(n_samples=1000, n_features=2,
                                    n_redundant=0, n_informative=2,
                                    random_state=42)
# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)
# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Hyperparameter tuning
param_grid = {'C': [0.1, 1, 10, 100],
              'gamma': ['scale', 'auto', 0.001, 0.01, 0.1, 1],
              'kernel': ['rbf', 'poly', 'sigmoid']}
svm = SVC()
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)
# Best model
best_svm = grid_search.best_estimator_
y_pred = best_svm.predict(X_test_scaled)
print(f"Best parameters: {grid_search.best_params_}")
print(f"Accuracy: {accuracy_score(y_test, y_pred):.3f}")
```

Multi-class SVM

Problem: SVMs are inherently binary classifiers

Solutions for Multi-class Classification:

1. **One-vs-Rest (OvR):**
   - Train k binary classifiers (one per class)
   - Class i vs. all other classes
   - Prediction: Choose class with highest confidence score
2. **One-vs-One (OvO):**
   - Train $\binom{k}{2} = \frac{k(k-1)}{2}$ binary classifiers
   - Each pair of classes
   - Prediction: Majority voting
3. **Directed Acyclic Graph (DAG-SVM):**
   - Hierarchical approach using OvO classifiers
   - More efficient prediction than standard OvO

Support Vector Regression (SVR)

Goal: Find function $f(x) = w^T\phi(x) + b$ that deviates from targets y_i by at most ε

ε-insensitive loss:

$$L_\varepsilon(y, f(x)) = \begin{cases} 0 & \text{if } |y - f(x)| \leq \varepsilon \\ |y - f(x)| - \varepsilon & \text{otherwise} \end{cases}$$

Optimization Problem:

$$\text{minimize} \quad \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \tag{12}$$

$$\text{subject to} \quad y_i - w^T\phi(x_i) - b \leq \varepsilon + \xi_i \tag{13}$$

$$w^T\phi(x_i) + b - y_i \leq \varepsilon + \xi_i^* \tag{14}$$

$$\xi_i, \xi_i^* \geq 0 \tag{15}$$

Applications: Time series forecasting, function approximation, financial modeling

Custom Kernels

Kernel Requirements (Mercer's Theorem):

- Must be symmetric: $K(x_i, x_j) = K(x_j, x_i)$
- Must be positive semi-definite
- Kernel matrix must have non-negative eigenvalues

Domain-Specific Kernels:

String Kernels: For text and biological sequences

$$K_{spectrum}(s_1, s_2) = \sum_{u \in \Sigma^k} \phi_u(s_1)\phi_u(s_2)$$

Graph Kernels: For structured data

$$K_{walk}(G_1, G_2) = \sum_{i,j} [\lambda^{-1}(I - \lambda A_1)]_{ij} [\lambda^{-1}(I - \lambda A_2)]_{ij}$$

Composite Kernels:

- Addition: $K(x_i, x_j) = K_1(x_i, x_j) + K_2(x_i, x_j)$
- Multiplication: $K(x_i, x_j) = K_1(x_i, x_j) \cdot K_2(x_i, x_j)$

SVM vs Other Classifiers

Aspect	SVM	Logistic Reg.	Random Forest	Neural Net
Interpretability	Medium	High	Medium	Low
Training Speed	Slow	Fast	Fast	Variable
Prediction Speed	Fast	Fast	Fast	Fast
Memory Usage	Low	Low	High	Variable
Overfitting Risk	Low	Medium	Low	High
Feature Scaling	Required	Recommended	Not needed	Required
High Dimensions	Excellent	Good	Poor	Good
Non-linear Data	Excellent	Poor	Excellent	Excellent
Probabilistic Output	No	Yes	Yes	Yes
Hyperparameter Tuning	Critical	Simple	Moderate	Complex

Decision Guide:

- **Use SVM when:** High dimensions, clear separation, small-medium datasets
- **Use Logistic Regression when:** Need probabilities, linear relationships
- **Use Random Forest when:** Mixed data types, feature interactions
- **Use Neural Networks when:** Very large data, complex patterns

Real-World Applications

Text Classification:

- Email spam detection
- Sentiment analysis
- Document categorization
- News article classification

Image Recognition:

- Face recognition
- Handwritten digit recognition
- Medical image analysis
- Object detection

Bioinformatics:

- Gene classification
- Protein structure prediction
- Cancer diagnosis
- Drug discovery

Finance:

- Credit scoring
- Fraud detection
- Algorithmic trading
- Risk assessment

Key Takeaways

Support Vector Machines

Core Concepts:

- Maximum Margin: Find optimal separating hyperplane
- Support Vectors: Only boundary points matter
- Kernel Trick: Handle non-linear data efficiently
- Dual Formulation: Transform to simpler optimization

Success Factors:

- Proper feature scaling
- Appropriate kernel selection
- Careful hyperparameter tuning
- Understanding data characteristics

Remember: SVMs excel in high-dimensional spaces but require careful preprocessing and parameter selection for optimal performance.

Questions?

❓ Discussion ❓

Next Topic: k-Nearest Neighbors: Distance metrics, curse of dimensionality

Department of Computer Science
Georgia State University