Support Vector Machine (SVM) Complete Cheatsheet

1. Problem Setup

Goal: Find a hyperplane that separates two classes with maximum margin

Training Data:

- Dataset: $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$
- $x_i \in \mathbb{R}^d$ (feature vectors)
- $y_i \in \{-1, +1\}$ (class labels)

Hyperplane Equation:

```
w·x + b = 0
```

- w: weight vector (normal to hyperplane)
- b: bias term
- x: input vector

Decision Function:

```
f(x) = sign(w·x + b)
```

---

2. Linear SVM - Hard Margin

Geometric Margin

Distance from point x to hyperplane:

```
distance = |w·x + b| / ||w||
```

Margin (γ): Distance from hyperplane to nearest point

```
γ = min(|w·xᵢ + b| / ||w||) for all i
```

Optimization Problem (Primal Form)

Maximize margin:

```
max (1/||w||)  ⟺  min (1/2)||w||²
```

Subject to constraints:

```
yᵢ(w·xᵢ + b) ≥ 1  for all i = 1,...,n
```

This ensures all points are correctly classified with margin $\geq 1/||w||$

Complete Primal Problem:

```
min w,b  (1/2)||w||²

subject to: yᵢ(w·xᵢ + b) ≥ 1, i = 1,...,n
```

---

3. Linear SVM - Soft Margin

Why Soft Margin?

- Data may not be perfectly separable
- Allow some misclassifications
- Introduces slack variables ξ_i

Slack Variables

```
ξᵢ ≥ 0  (amount of constraint violation for point i)
```

Interpretation:

- $\xi_i = 0$: point is correctly classified
- $0 < \xi_i < 1$: point is inside margin but correctly classified
- $\xi_i \geq 1$: point is misclassified

Optimization Problem

Primal Form:

```
min w,b,ξ  (1/2)||w||² + C·Σξᵢ

subject to:
  yᵢ(w·xᵢ + b) ≥ 1 - ξᵢ  for all i
  ξᵢ ≥ 0  for all i
```

C (Regularization Parameter):

- Large C: prioritize correct classification (small margin, low bias, high variance)
- Small C: prioritize large margin (allow misclassifications, high bias, low variance)

---

4. Dual Formulation (Lagrangian)

Lagrangian Function

Primal variables: w, b, ξ
Dual variables (Lagrange multipliers): α, μ

```
L(w,b,ξ,α,μ) = (1/2)||w||² + C·Σξᵢ - Σαᵢ[yᵢ(w·xᵢ + b) - 1 + ξᵢ] - Σμᵢξᵢ
```

KKT Conditions

Stationarity:

```
∂L/∂w = 0  ⟹  w = Σαᵢyᵢxᵢ
∂L/∂b = 0  ⟹  Σαᵢyᵢ = 0
∂L/∂ξᵢ = 0  ⟹  αᵢ = C - μᵢ
```

Primal Feasibility:

```
yᵢ(w·xᵢ + b) ≥ 1 - ξᵢ
ξᵢ ≥ 0
```

Dual Feasibility:

```
αᵢ ≥ 0
μᵢ ≥ 0
```

Complementary Slackness:

```
αᵢ[yᵢ(w·xᵢ + b) - 1 + ξᵢ] = 0
μᵢξᵢ = 0
```

Dual Problem

```
```

```
max α  Σαᵢ - (1/2)ΣΣαᵢαⱼyᵢyⱼ(xᵢ·xⱼ)

subject to:
  Σαᵢyᵢ = 0
  0 ≤ αᵢ ≤ C  for all i
```

Why use dual?

- Easier to solve for high-dimensional data
- Enables kernel trick
- Only depends on dot products $x_i \cdot x_j$

---

5. Support Vectors

Classification of points based on α:

1. **Non-support vectors:** $\alpha_i = 0$
   - Correctly classified, outside margin
   - Don't affect decision boundary
2. **Support vectors on margin:** $0 < \alpha_i < C$
   - Exactly on margin boundary
   - $\xi_i = 0$, $y_i(w \cdot x_i + b) = 1$
3. **Support vectors inside margin/misclassified:** $\alpha_i = C$
   - Inside margin or misclassified
   - $\xi_i > 0$

Computing w and b:

```
w = Σ(αᵢyᵢxᵢ)  [sum over support vectors]

b = yₛ - w·xₛ  [for any support vector xₛ with 0 < αₛ < C]
```

Better (averaged):

```
b = (1/Nₛᵥ) Σ[yₛ - w·xₛ]  [average over all support vectors with 0 < α < C]
```

---

6. Kernel Trick

Motivation

Transform data to higher-dimensional space where it's linearly separable

Feature Mapping:

```
ϕ: ℝᵈ → ℝᴰ  (D >> d)
```

Kernel Function:

```
K(xᵢ, xⱼ) = ϕ(xᵢ)·ϕ(xⱼ)
```

Computes dot product in high-dimensional space without explicit transformation!

Dual Problem with Kernels

```
max α  Σαᵢ - (1/2)ΣΣαᵢαⱼyᵢyⱼK(xᵢ,xⱼ)

subject to:
  Σαᵢyᵢ = 0
  0 ≤ αᵢ ≤ C
```

Decision Function with Kernels

```
f(x) = sign(ΣαᵢyᵢK(xᵢ,x) + b)
```

---

7. Common Kernel Functions

1. Linear Kernel

```
K(xᵢ,xⱼ) = xᵢ·xⱼ
```

- Use when data is linearly separable
- Equivalent to no kernel
- Fast computation

2. Polynomial Kernel

```
K(xᵢ,xⱼ) = (γ·xᵢ·xⱼ + r)ᵈ
```

- γ: kernel coefficient (default: 1/n_features)
- r: independent term (default: 0)
- d: degree (default: 3)
- Use for non-linear boundaries with polynomial shape

Special case (Homogeneous):

```
K(xᵢ,xⱼ) = (xᵢ·xⱼ)ᵈ
```

3. Radial Basis Function (RBF/Gaussian) Kernel ⭐

```
K(xᵢ,xⱼ) = exp(-γ||xᵢ - xⱼ||²)
```

- γ: kernel coefficient (default: 1/n_features)
- Most popular kernel
- Can handle highly non-linear boundaries
- Maps to infinite-dimensional space

γ parameter effect:

- Large γ: narrow Gaussian, complex decision boundary (high variance)
- Small γ: wide Gaussian, smooth decision boundary (high bias)

4. Sigmoid Kernel

```
K(xᵢ,xⱼ) = tanh(γ·xᵢ·xⱼ + r)
```

- Similar to neural network activation
- Not always positive semi-definite (can cause issues)

5. Custom Kernels

Mercer's Condition: Kernel must satisfy

```
∫∫K(x,x')g(x)g(x')dxdx' ≥ 0  for all g
```

---

8. Multi-Class Classification

SVMs are binary classifiers. For multi-class:

One-vs-Rest (OvR)

- Train K binary classifiers (K = number of classes)
- Classifier k: class k vs all others
- Predict: class with highest decision function value

Decision:

```
ŷ = argmax(wₖ·x + bₖ)
      k
```

One-vs-One (OvO)

- Train K(K-1)/2 binary classifiers
- One for each pair of classes
- Predict: class that wins most pairwise comparisons (voting)

---

9. SVM Regression (SVR)

ε-insensitive Loss

Goal: Find function f(x) = w·x + b such that $|y_i - f(x_i)| \leq \varepsilon$

ε-tube: No penalty for errors within ±ε

Optimization Problem

```
min w,b,ξ,ξ*  (1/2)||w||² + C·Σ(ξᵢ + ξᵢ*)

subject to:
  yᵢ - (w·xᵢ + b) ≤ ε + ξᵢ
  (w·xᵢ + b) - yᵢ ≤ ε + ξᵢ*
  ξᵢ, ξᵢ* ≥ 0
```

Dual Form:

```
max α,α*  -ε·Σ(αᵢ + αᵢ*) + Σyᵢ(αᵢ - αᵢ*) - (1/2)ΣΣ(αᵢ - αᵢ*)(αⱼ - αⱼ*)K(xᵢ,xⱼ)

subject to:
  Σ(αᵢ - αᵢ*) = 0
  0 ≤ αᵢ, αᵢ* ≤ C
```

Prediction:

```
f(x) = Σ(αᵢ - αᵢ*)K(xᵢ,x) + b
```

10. Key Formulas Summary

Training (Dual Optimization)

```
max α  Σαᵢ - (1/2)ΣΣαᵢαⱼyᵢyⱼK(xᵢ,xⱼ)
s.t.   Σαᵢyᵢ = 0, 0 ≤ αᵢ ≤ C
```

Weight Vector

```
w = Σαᵢyᵢϕ(xᵢ)  [in feature space]
```

Bias Term

```
b = (1/Nₛᵥ)Σ[yₛ - ΣαᵢyᵢK(xᵢ,xₛ)]  [average over support vectors]
```

Prediction

```
f(x) = sign(ΣαᵢyᵢK(xᵢ,x) + b)
```

Margin

```
margin = 2/||w||
```

---

11. Hyperparameter Tuning

C (Regularization)

Effect:

- ↑ C: Lower bias, higher variance (harder margin, fit training data closely)
- ↓ C: Higher bias, lower variance (softer margin, better generalization)

Typical values: 0.1, 1, 10, 100

γ (RBF Kernel)

Effect:

- ↑ γ: Lower bias, higher variance (more complex boundary)

- ↓ γ: Higher bias, lower variance (smoother boundary)

Formula: $\gamma = 1/(2\sigma^2)$

Typical values: 0.001, 0.01, 0.1, 1

Selection Strategy

Use **Grid Search** or **Random Search** with **Cross-Validation**

Common ranges:

```
param_grid = {
    'C': [0.1, 1, 10, 100, 1000],
    'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
    'kernel': ['rbf', 'poly', 'sigmoid']
}
```

---

12. Computational Complexity

Training

- **Primal:** O(nd) per iteration
- **Dual:** O(n^2 to n^3) depending on solver
  - n: number of samples
  - d: number of features

For large n: Use linear SVM or approximations

Prediction

```
O(nₛᵥ · d)
```

- n_{sv}: number of support vectors
- Typically $n_{sv} << n$

---

13. Advantages vs Disadvantages

☑ Advantages

1. **Effective in high dimensions** (d > n)
2. **Memory efficient** (only stores support vectors)
3. **Versatile** (different kernels)
4. **Robust** to outliers (when C is small)
5. **Global optimum** (convex optimization)
6. **Good generalization** with proper tuning

❌ Disadvantages

1. **Slow for large datasets** ($O(n^2)$ to $O(n^3)$)
2. **No probability estimates** (requires Platt scaling)
3. **Sensitive to feature scaling**
4. **Kernel/parameter selection** can be difficult
5. **Black box** with non-linear kernels (interpretability)
6. **Not suitable for large n** (use linear SVM or logistic regression)

---

14. Feature Scaling

Critical: SVMs are sensitive to feature scales

Standardization (Recommended)

```
x' = (x - μ) / σ
```

- Mean = 0, Std = 1
- Preserves outliers

Normalization

```
x' = (x - min) / (max - min)
```

- Range [0, 1]
- Affected by outliers

Rule: Always scale features before training!

---

15. Implementation Tips

Scikit-learn Example

```python
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train SVM
svm = SVC(kernel='rbf', C=1.0, gamma='scale')
svm.fit(X_train_scaled, y_train)
```

```python
# Hyperparameter tuning
param_grid = {'C': [0.1, 1, 10], 'gamma': [0.001, 0.01, 0.1]}
grid = GridSearchCV(SVC(), param_grid, cv=5)
grid.fit(X_train_scaled, y_train)

# Access support vectors
n_support = svm.n_support_  # number per class
support_vectors = svm.support_vectors_
support_indices = svm.support_

# Prediction
y_pred = svm.predict(X_test_scaled)
```

---

16. When to Use SVM

✅ Use SVM when:

- High-dimensional data (text classification, genomics)
- Clear margin of separation exists
- More features than samples
- Need robust model against outliers
- Non-linear decision boundaries (with kernels)

❌ Avoid SVM when:

- Very large datasets (n > 100,000)
- Need probability estimates (use logistic regression)
- Need interpretability (use decision trees)
- Data has significant noise/overlapping classes
- Computational resources are limited

---

17. Common Applications

1. **Text Classification** (spam detection, sentiment analysis)
2. **Image Recognition** (face detection, handwriting recognition)
3. **Bioinformatics** (protein classification, gene expression)
4. **Medical Diagnosis** (cancer classification)
5. **Time Series Prediction** (with SVR)
6. **Financial Forecasting**
7. **Remote Sensing** (land cover classification)

---

18. Important Notes

Probability Estimates

SVM doesn't naturally output probabilities. Use **Platt Scaling:**

```
P(y=1|x) = 1 / (1 + exp(A·f(x) + B))
```

- Requires additional cross-validation
- Enable with `probability=True` in sklearn

Class Imbalance

- Use **class_weight='balanced'** parameter
- Or manually set: `class_weight={0: w₀, 1: w₁}`
- Adjusts C: `Cᵢ = C × wᵢ`

Kernel Selection Guidelines

1. Start with **linear** kernel (baseline)
2. Try **RBF** if linear doesn't work well
3. Use **polynomial** for specific problem knowledge
4. Avoid **sigmoid** (rarely useful)

Quick Reference Card

Concept	Formula
Hyperplane	$w \cdot x + b = 0$
Primal Problem	$\min (1/2)\|\|w\|\|^2 + C \cdot \Sigma\xi_i$
Dual Problem	$\max \Sigma\alpha_i - (1/2)\Sigma\Sigma\alpha_i\alpha_j y_i y_j K(x_i,x_j)$
Decision Function	$f(x) = \text{sign}(\Sigma\alpha_i y_i K(x_i,x) + b)$
Margin	$2/\|\|w\|\|$
RBF Kernel	$K(x,x') = \exp(-\gamma\|\|x-x'\|\|^2)$
Support Vectors	Points where $0 < \alpha_i \leq C$

Remember:

- Always scale features
- Tune C and γ carefully
- Use cross-validation
- Consider linear SVM first for large datasets