# Module 6: Linear Classification

Machine Learning Course

## Contents

**1 Introduction to Linear Classification** **3**
1.1 Overview of Classification . . . 3
1.2 Topics Covered . . . 3
1.3 Applications . . . 3

**2 Linear Decision Boundaries** **3**
2.1 Mathematical Formulation . . . 3
2.2 Making Predictions . . . 4
2.3 Correctness Condition . . . 4

**3 Loss Function for Classification** **4**
3.1 Defining the Loss . . . 4
3.2 Gradient of the Loss . . . 5

**4 The Perceptron Algorithm** **5**
4.1 Algorithm Description . . . 5
4.2 Connection to Stochastic Gradient Descent . . . 5
4.3 The Perceptron in Action . . . 5
4.4 Perceptron Convergence Theorem . . . 6

**5 The Hard-Margin Support Vector Machine** **6**
5.1 The Margin Concept . . . 6
5.2 The Learning Problem . . . 6
5.3 Maximizing the Margin . . . 7
5.4 The Optimization Problem . . . 7
5.5 Support Vectors . . . 8

**6 Example: Iris Dataset** **8**
6.1 Dataset Description . . . 8
6.2 Binary Classification Example . . . 9

**7 The Soft-Margin SVM** **9**
7.1 Handling Non-linearly Separable Data . . . 9
7.2 Slack Variables . . . 9
7.3 The Optimization Problem . . . 10

**8 Parameter Selection for SVMs** **10**
8.1 The Role of Parameter $C$ . . . 10
8.2 Practical Implications . . . 10

**9 Sentiment Analysis Example** **11**
9.1 Dataset Description . . . 11
9.2 Example Sentences . . . 11
9.3 Feature Representation . . . 12

**10 Experimental Results with Different $C$ Values** **12**
10.1 Performance Metrics . . . 12
10.2 Results Table . . . 12
10.3 Analysis of Results . . . 13

**11 Cross-Validation for Parameter Tuning** **13**
11.1 The Cross-Validation Procedure . . . 13
11.2 Cross-Validation Results . . . 14

**12 Interpreting the Trade-offs** **14**
12.1 Bias-Variance Trade-off . . . 14
12.2 Margin vs. Classification Accuracy . . . 15
12.3 Model Complexity . . . 15

**13 Practical Considerations** **15**
13.1 Feature Scaling . . . 15
13.2 Handling Large Datasets . . . 16
13.3 Kernel Selection . . . 16
13.4 Multi-class Classification . . . 16

**14 Summary and Key Takeaways** **16**
14.1 Linear Classification Approaches . . . 16
14.2 The Importance of Parameter Tuning . . . 17
14.3 Practical Guidelines . . . 17
14.4 Future Directions . . . 17

# 1 Introduction to Linear Classification

## 1.1 Overview of Classification

Classification is a fundamental task in machine learning where the goal is to assign discrete labels to data points. In binary classification, we aim to distinguish between two classes, typically labeled as $+1$ and $-1$. Linear classification is one of the simplest yet powerful approaches to this problem, using a linear decision boundary to separate the classes.

## 1.2 Topics Covered

This module explores several key aspects of linear classification:

1. Linear decision boundaries for binary classification
2. The Perceptron algorithm
3. Maximizing the margin (Support Vector Machines)
4. The soft-margin SVM for non-separable data

## 1.3 Applications

Linear classification methods have numerous applications across various domains:

- Text categorization (e.g., spam detection, sentiment analysis)
- Image recognition (e.g., simple object detection)
- Medical diagnosis (e.g., disease classification based on symptoms)
- Financial analysis (e.g., credit approval)

# 2 Linear Decision Boundaries

## 2.1 Mathematical Formulation

In binary classification, we have:

- Data points $x \in \mathbb{R}^d$ (feature vectors in $d$-dimensional space)
- Labels $y \in \{-1, +1\}$ (binary class labels)

A linear classifier is defined by:

- A weight vector $w \in \mathbb{R}^d$
- A bias term $b \in \mathbb{R}$

The decision boundary is the hyperplane defined by:

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

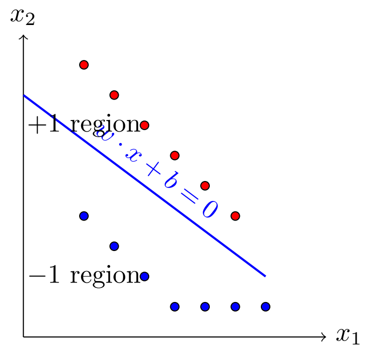

Figure 1: Example of a linear decision boundary separating two classes

### 2.2 Making Predictions

For a new data point $x$, the predicted label is:

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

### 2.3 Correctness Condition

A linear classifier correctly classifies a point $(x, y)$ if and only if:

$$y(w \cdot x + b) > 0 \tag{3}$$

This elegant formulation combines both cases:

- When $y = +1$, we need $w \cdot x + b > 0$
- When $y = -1$, we need $w \cdot x + b < 0$

## 3 Loss Function for Classification

### 3.1 Defining the Loss

To train a linear classifier, we need a loss function that quantifies how "wrong" our model is on a given example. A natural loss function for classification is:

$$L(w, b; x, y) = \begin{cases} 0 & \text{if } y(w \cdot x + b) > 0 \text{ (correct classification)} \\ -y(w \cdot x + b) & \text{if } y(w \cdot x + b) \leq 0 \text{ (incorrect classification)} \end{cases} \tag{4}$$

This loss function has several desirable properties:

- Zero loss for correctly classified points
- Positive loss for incorrectly classified points
- The loss increases as the point gets further on the wrong side of the boundary

### 3.2 Gradient of the Loss

For stochastic gradient descent, we need the gradient of the loss function:

$$\nabla_w L(w,b;x,y) = \begin{cases} 0 & \text{if } y(w\cdot x+b) > 0 \\ -yx & \text{if } y(w\cdot x+b) \leq 0 \end{cases} \tag{5}$$

$$\frac{\partial L}{\partial b}(w,b;x,y) = \begin{cases} 0 & \text{if } y(w\cdot x+b) > 0 \\ -y & \text{if } y(w\cdot x+b) \leq 0 \end{cases} \tag{6}$$

## 4 The Perceptron Algorithm

### 4.1 Algorithm Description

The Perceptron is a simple yet powerful algorithm for learning linear classifiers. It iteratively updates the model parameters based on misclassified points.

**The Perceptron Algorithm**

```
1. Initialize $w = 0$ and $b = 0$
2. Repeat until convergence:
    (a) For each training example $(x, y)$:
        i. If $y(w \cdot x + b) \leq 0$ (point is misclassified):
            A. $w = w + yx$
            B. $b = b + y$
```

### 4.2 Connection to Stochastic Gradient Descent

The Perceptron update rule can be derived from stochastic gradient descent on the loss function defined earlier:

$$w = w - \eta \nabla_w L(w,b;x,y) \tag{7}$$

$$b = b - \eta \frac{\partial L}{\partial b}(w,b;x,y) \tag{8}$$

With learning rate $\eta = 1$ and for misclassified points where $y(w \cdot x + b) \leq 0$:

$$w = w - (-yx) = w + yx \tag{9}$$

$$b = b - (-y) = b + y \tag{10}$$

### 4.3 The Perceptron in Action

The Perceptron algorithm has been shown to work well on linearly separable data. For example, on a dataset with 85 linearly separable points, the Perceptron successfully finds a decision boundary that correctly classifies all points.

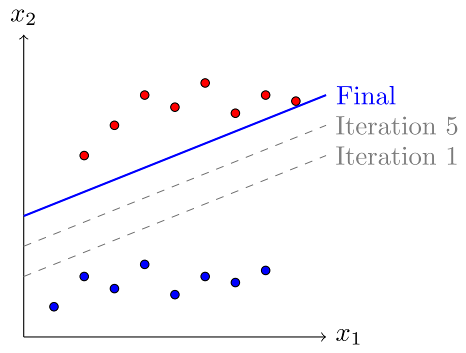

Figure 2: Illustration of Perceptron convergence on linearly separable data. The decision boundary evolves over iterations until it perfectly separates the classes.

### 4.4 Perceptron Convergence Theorem

A fundamental result for the Perceptron algorithm:

**Perceptron Convergence Theorem**
If the training data is linearly separable, then:

- The Perceptron algorithm will find a linear classifier with zero training error
- It will converge within a finite number of steps

However, the Perceptron algorithm has limitations:

- It only works for linearly separable data
- It may find any linear separator that works, not necessarily the "best" one
- The solution depends on the order of processing the training examples

This raises an important question: Among all possible linear separators, which one should we choose?

## 5 The Hard-Margin Support Vector Machine

### 5.1 The Margin Concept

The margin of a linear classifier is the distance from the decision boundary to the nearest training point. A classifier with a large margin is likely to generalize better to unseen data.

### 5.2 The Learning Problem

Given training data $\{(x^{(i)}, y^{(i)}) : i = 1, \ldots, n\} \subset \mathbb{R}^d \times \{-1, +1\}$, we want to find $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ such that:

$$y^{(i)}(w \cdot x^{(i)} + b) > 0 \quad \text{for all } i = 1, \ldots, n \tag{11}$$

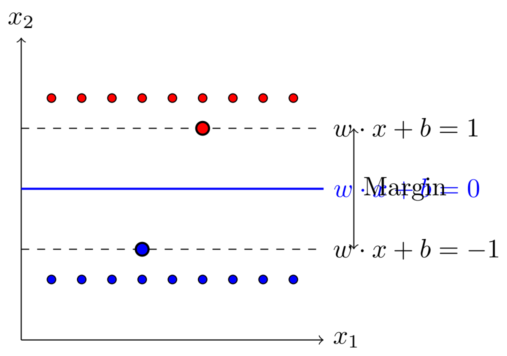

Figure 3: Illustration of the margin in a linearly separable dataset. The support vectors (highlighted) lie exactly on the margin boundaries.

By scaling $w$ and $b$, we can equivalently require:

$$y^{(i)}(w \cdot x^{(i)} + b) \geq 1 \quad \text{for all } i = 1, \ldots, n \tag{12}$$

This formulation ensures that all points are not only correctly classified but also at least a certain distance from the decision boundary.

## 5.3 Maximizing the Margin

The margin $\gamma$ of a linear classifier is the distance from the decision boundary to the nearest training point. For a linear classifier with $\|w\| = 1$, this distance is given by $|w \cdot x + b|$.

With our constraint $y^{(i)}(w \cdot x^{(i)} + b) \geq 1$, the margin is $\gamma = 1/\|w\|$.

Therefore, maximizing the margin is equivalent to minimizing $\|w\|$, or equivalently, minimizing $\|w\|^2$ (which is differentiable everywhere).

## 5.4 The Optimization Problem

The hard-margin SVM optimization problem is:

**Hard-Margin SVM Optimization Problem**

$$\min_{w \in \mathbb{R}^d, b \in \mathbb{R}} \|w\|^2 \tag{13}$$

$$\text{subject to: } y^{(i)}(w \cdot x^{(i)} + b) \geq 1 \quad \text{for all } i = 1, 2, \ldots, n \tag{14}$$

This is a convex quadratic optimization problem with linear constraints, which means:

- It has a unique global minimum
- It can be solved efficiently using standard optimization techniques
- Duality theory provides insights into the structure of the solution

### 5.5 Support Vectors

The solution to the SVM optimization problem has an interesting property: most training points have no influence on the decision boundary. The only points that matter are those that lie exactly on the margin, i.e., points for which $y^{(i)}(w \cdot x^{(i)} + b) = 1$. These points are called support vectors.

The optimal weight vector can be expressed as a linear combination of the support vectors:

$$w = \sum_{i=1}^{n} \alpha_i y^{(i)} x^{(i)} \tag{15}$$

where $\alpha_i \geq 0$ are Lagrange multipliers that are non-zero only for support vectors.

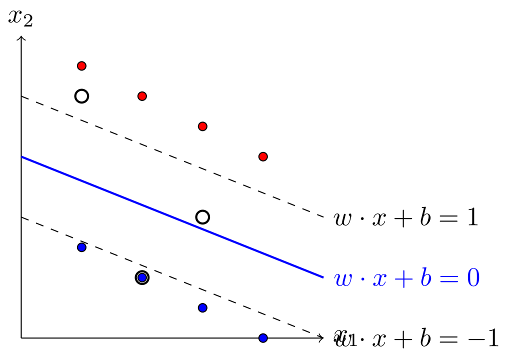

Figure 4: Support vectors are the points that lie exactly on the margin boundaries. The decision boundary is determined entirely by these points.

## 6 Example: Iris Dataset

### 6.1 Dataset Description

The Iris dataset is a classic dataset in machine learning, collected by the botanist Edgar Anderson and made famous by the statistician Ronald Fisher. It contains measurements of 150 iris flowers from three different species:

- Iris setosa
- Iris versicolor
- Iris virginica

For each flower, four measurements were taken:

- Sepal length
- Sepal width
- Petal length
- Petal width

## 6.2 Binary Classification Example

For simplicity, we can consider a binary classification problem using only two of the species (setosa and versicolor) and two of the features (sepal width and petal width).

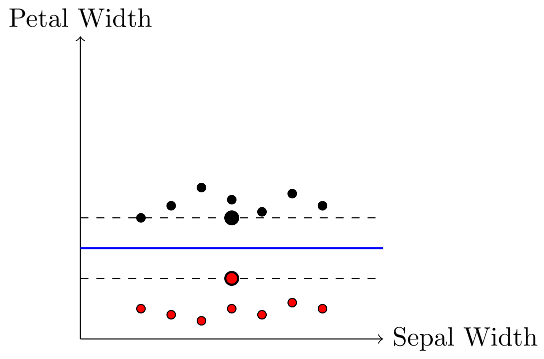

Figure 5: SVM classification of Iris setosa (red circles) and Iris versicolor (black triangles) using sepal width and petal width features.

In this example, the data is linearly separable, and the SVM finds the maximum-margin linear classifier. The support vectors are the points that lie exactly on the margin boundaries.

# 7 The Soft-Margin SVM

## 7.1 Handling Non-linearly Separable Data

The hard-margin SVM assumes that the data is linearly separable. However, in real-world scenarios, data is often not perfectly separable due to noise or outliers. To handle such cases, we can use a soft-margin SVM, which allows for some misclassifications.

## 7.2 Slack Variables

We introduce slack variables $\xi_i \geq 0$ for each training point, which measure the degree of misclassification:

$$y^{(i)}(w \cdot x^{(i)} + b) \geq 1 - \xi_i \quad \text{for all } i = 1, 2, \ldots, n \tag{16}$$

The interpretation of $\xi_i$ is:

- $\xi_i = 0$: The point is correctly classified and on or beyond the margin
- $0 < \xi_i \leq 1$: The point is correctly classified but within the margin
- $\xi_i > 1$: The point is misclassified

### 7.3 The Optimization Problem

The soft-margin SVM optimization problem is:

**Soft-Margin SVM Optimization Problem**

$$\min_{w\in\mathbb{R}^d, b\in\mathbb{R}, \xi\in\mathbb{R}^n} \|w\|^2 + C\sum_{i=1}^{n}\xi_i \tag{17}$$

$$\text{subject to: } y^{(i)}(w\cdot x^{(i)} + b) \geq 1 - \xi_i \quad \text{for all } i = 1, 2, \ldots, n \tag{18}$$

$$\xi_i \geq 0 \quad \text{for all } i = 1, 2, \ldots, n \tag{19}$$

The parameter $C > 0$ controls the trade-off between maximizing the margin and minimizing the classification error. A larger $C$ places more emphasis on correctly classifying all training points, while a smaller $C$ places more emphasis on maximizing the margin.

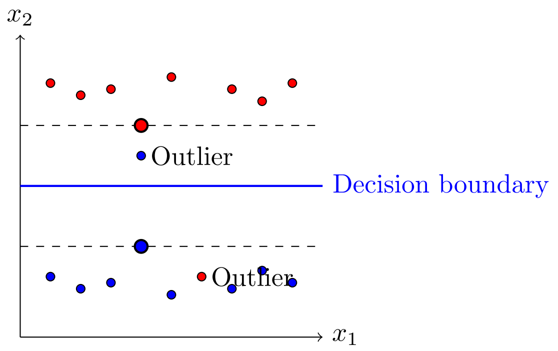

Figure 6: Soft-margin SVM allows for some misclassifications to handle outliers or non-linearly separable data.

## 8 Parameter Selection for SVMs

### 8.1 The Role of Parameter $C$

The parameter $C$ in the soft-margin SVM formulation controls the trade-off between maximizing the margin and minimizing the classification error:

- Small $C$: Places more emphasis on maximizing the margin, even if it means allowing more training errors
- Large $C$: Places more emphasis on minimizing training errors, potentially at the cost of a smaller margin

### 8.2 Practical Implications

The choice of $C$ has significant practical implications:

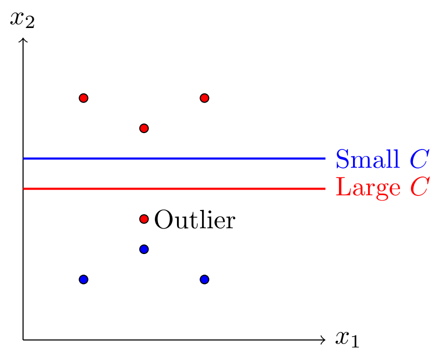

Figure 7: Effect of parameter $C$ on the decision boundary. A small $C$ (blue line) allows the outlier to be misclassified to maintain a larger margin. A large $C$ (red line) adjusts the boundary to correctly classify the outlier, resulting in a smaller margin.

- **Overfitting vs. Underfitting**: Large $C$ values can lead to overfitting, while small $C$ values might result in underfitting
- **Number of Support Vectors**: Larger $C$ values typically result in fewer support vectors
- **Generalization Performance**: The optimal $C$ value balances training accuracy and generalization to unseen data

## 9 Sentiment Analysis Example

### 9.1 Dataset Description

To illustrate the importance of parameter selection, we'll examine a sentiment analysis dataset:

- **Source**: Reviews from Amazon, Yelp, and IMDB
- **Labels**: Binary classification (positive or negative sentiment)
- **Representation**: Bag-of-words with a vocabulary of 4500 words
- **Size**: 2500 training sentences, 500 test sentences

### 9.2 Example Sentences

The dataset contains sentences like:

- ”Needless to say, I wasted my money.” (Negative)
- ”He was very impressed when going from the original battery to the extended battery.” (Positive)
- ”I have to jiggle the plug to get it to line up right to get decent volume.” (Negative)
- ”Will order from them again!” (Positive)

## 9.3 Feature Representation

The bag-of-words representation transforms each sentence into a high-dimensional vector:

- Each dimension corresponds to a word in the vocabulary
- The value in each dimension represents the presence or frequency of the word in the sentence
- This creates a sparse vector representation (most entries are zero)

$$\begin{array}{c} \text{"Will order from them again!"} \\ \Big\downarrow \\ [0, 0, ..., \underset{\substack{\uparrow \\ \text{"will"}}}{1}, 0, ..., \underset{\substack{\uparrow \\ \text{"order"}}}{1}, 0, ..., \underset{\substack{\uparrow \\ \text{"from"}}}{1}, 0, ..., \underset{\substack{\uparrow \\ \text{"again"}}}{1}, 0, ...] \end{array}$$

Figure 8: Bag-of-words representation of a sentence. Each word in the vocabulary corresponds to a dimension in the feature vector.

# 10 Experimental Results with Different $C$ Values

## 10.1 Performance Metrics

For each value of $C$, we measure three key metrics:

- **Training error**: Percentage of misclassified training examples
- **Test error**: Percentage of misclassified test examples
- **Number of support vectors**: Points that influence the decision boundary

## 10.2 Results Table

| **C** | **Training Error (%)** | **Test Error (%)** | **# Support Vectors** |
|---|---|---|---|
| 0.01 | 23.72 | 28.4 | 2294 |
| 0.1 | 7.88 | 18.4 | 1766 |
| 1 | 1.12 | 16.8 | 1306 |
| 10 | 0.16 | 19.4 | 1105 |
| 100 | 0.08 | 19.4 | 1035 |
| 1000 | 0.08 | 19.4 | 950 |

Table 1: Performance metrics for different values of parameter $C$ on the sentiment analysis dataset

### 10.3 Analysis of Results

Several important trends can be observed from the experimental results:

1. **Training Error**: Decreases monotonically as $C$ increases
   - At $C = 0.01$: High training error (23.72%)
   - At $C = 1000$: Near-zero training error (0.08%)
2. **Test Error**: Initially decreases as $C$ increases, then increases
   - At $C = 0.01$: High test error (28.4%)
   - At $C = 1$: Lowest test error (16.8%)
   - At $C \geq 10$: Test error increases to 19.4%
3. **Number of Support Vectors**: Decreases as $C$ increases
   - At $C = 0.01$: Many support vectors (2294)
   - At $C = 1000$: Fewer support vectors (950)

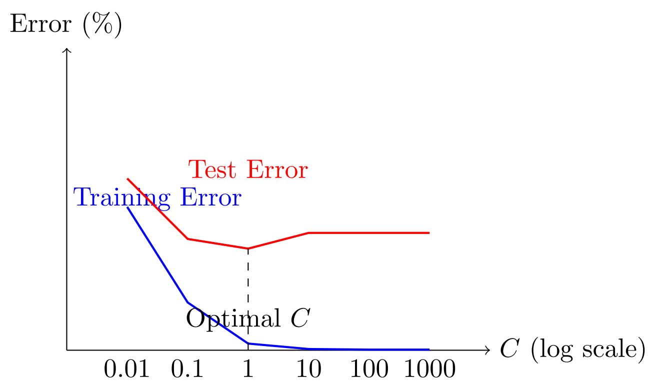

Figure 9: Training and test error as a function of parameter $C$ (log scale). The optimal $C$ value minimizes the test error.

# 11 Cross-Validation for Parameter Tuning

## 11.1 The Cross-Validation Procedure

To find the optimal value of $C$, we can use cross-validation:

1. Divide the training data into $k$ equal folds (e.g., $k = 5$)
2. For each value of $C$:
   (a) For each fold $i$ (from 1 to $k$):

        i. Train an SVM on all folds except fold $i$
        ii. Evaluate the model on fold $i$
    (b) Calculate the average error across all $k$ folds

3. Select the $C$ value with the lowest average error

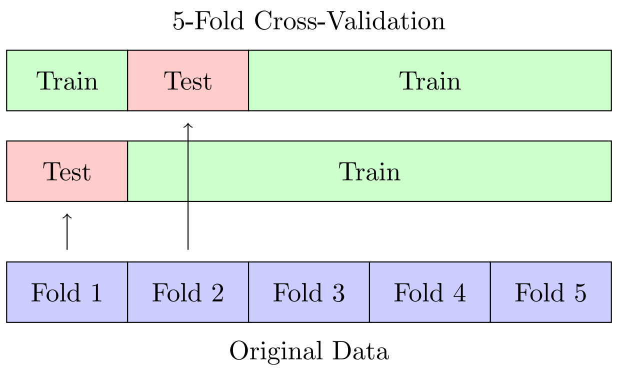

Figure 10: Illustration of 5-fold cross-validation. The data is divided into 5 folds, and each fold serves as the test set once while the remaining folds form the training set.

### 11.2 Cross-Validation Results

For the sentiment analysis dataset, 5-fold cross-validation was performed to select the optimal $C$ value:

- The optimal value was found to be $C = 0.32$
- With this value, the test error was 15.6%

This is better than any of the individual $C$ values tested in the previous experiment, highlighting the importance of proper parameter tuning.

## 12 Interpreting the Trade-offs

### 12.1 Bias-Variance Trade-off

The results illustrate the classic bias-variance trade-off in machine learning:

- **Small $C$ values** (e.g., $C = 0.01$):
    - High bias (underfitting)
    - Simple model with large margin
    - High training and test error

- **Large $C$ values** (e.g., $C = 1000$):
  - High variance (overfitting)
  - Complex model with small margin
  - Low training error but higher test error
- **Optimal $C$ value** (e.g., $C = 0.32$ from cross-validation):
  - Balances bias and variance
  - Moderate margin size
  - Best generalization performance

### 12.2 Margin vs. Classification Accuracy

The parameter $C$ controls the trade-off between margin size and classification accuracy:

- **Small $C$**: Prioritizes larger margins at the expense of training accuracy
- **Large $C$**: Prioritizes training accuracy at the expense of margin size

### 12.3 Model Complexity

The number of support vectors is an indicator of model complexity:

- More support vectors generally mean a more complex model
- As $C$ increases, the number of support vectors decreases
- However, too few support vectors might indicate overfitting to the training data

## 13 Practical Considerations

### 13.1 Feature Scaling

SVMs are sensitive to the scale of the features. It's generally recommended to scale the features before training an SVM:

- **Standardization**: $x' = \frac{x-\mu}{\sigma}$
- **Min-max scaling**: $x' = \frac{x-\min(x)}{\max(x)-\min(x)}$

### 13.2 Handling Large Datasets

Standard SVM implementations have time complexity $O(n^2)$ to $O(n^3)$, where $n$ is the number of training examples. This can be prohibitive for large datasets. Several approaches exist for scaling SVMs to large datasets:

- **Chunking**: Solve the optimization problem in smaller chunks
- **Sequential Minimal Optimization (SMO)**: Optimize two Lagrange multipliers at a time
- **Stochastic gradient descent**: Approximate the SVM solution using SGD
- **Linear SVMs**: For linear kernels, specialized algorithms like LIBLINEAR can scale to millions of examples

### 13.3 Kernel Selection

While linear SVMs are effective for many high-dimensional problems (like text classification), non-linear kernels can be useful for other types of data:

- **Linear kernel**: $K(x, z) = x \cdot z$
- **Polynomial kernel**: $K(x, z) = (x \cdot z + c)^d$
- **Radial Basis Function (RBF) kernel**: $K(x, z) = \exp(-\gamma \|x - z\|^2)$
- **Sigmoid kernel**: $K(x, z) = \tanh(\alpha x \cdot z + c)$

### 13.4 Multi-class Classification

SVMs are inherently binary classifiers, but they can be extended to multi-class problems using several strategies:

- **One-vs-Rest**: Train $k$ binary SVMs, each separating one class from the rest
- **One-vs-One**: Train $\binom{k}{2}$ binary SVMs, one for each pair of classes
- **Direct Multi-class Formulation**: Extend the SVM optimization problem to directly handle multiple classes

## 14 Summary and Key Takeaways

### 14.1 Linear Classification Approaches

We've explored several approaches to linear classification:

- **Perceptron**: A simple, iterative algorithm that finds any linear separator for linearly separable data
- **Hard-margin SVM**: Finds the maximum-margin linear separator for linearly separable data
- **Soft-margin SVM**: Extends the hard-margin SVM to handle non-linearly separable data by allowing some misclassifications

## 14.2 The Importance of Parameter Tuning

The soft-margin SVM parameter $C$ controls the trade-off between margin size and classification accuracy:

- Small $C$ values prioritize larger margins, potentially allowing more training errors
- Large $C$ values prioritize minimizing training errors, potentially at the cost of smaller margins
- The optimal $C$ value balances these trade-offs to achieve the best generalization performance
- Cross-validation is an effective method for selecting the optimal $C$ value

## 14.3 Practical Guidelines

Based on our exploration, here are some practical guidelines for using linear classifiers:

- **Start with a linear SVM**: For many problems, especially high-dimensional ones like text classification, a linear SVM is a good baseline
- **Use cross-validation for parameter tuning**: Test a range of $C$ values (e.g., $10^{-2}$ to $10^3$) and select the one with the best validation performance
- **Monitor both training and test performance**: This helps identify underfitting or overfitting
- **Consider the number of support vectors**: Too many might indicate underfitting, while too few might indicate overfitting
- **Scale features appropriately**: SVMs are sensitive to the scale of the features
- **For non-linear problems**: Consider using kernel SVMs or other non-linear classifiers

## 14.4 Future Directions

Linear classification continues to be an active area of research and development:

- **Scalable algorithms**: Developing more efficient algorithms for large-scale linear classification
- **Online learning**: Adapting linear classifiers for streaming data
- **Deep learning**: Combining linear classifiers with deep neural networks
- **Interpretability**: Enhancing the interpretability of linear classifiers for complex domains