SVM - Support Vector Machine Fundamentos Matemáticos e Implementação em Java para Projetos Profissionais

O Support Vector Machine (SVM) é um dos algoritmos mais sofisticados e eficazes para problemas de classificação e regressão.

Ao buscar o hiperplano que maximiza a separação entre classes, o SVM oferece precisão elevada, sendo robusto mesmo em contextos de alta dimensionalidade.

Este artigo explora detalhadamente a teoria do SVM, cobrindo desde a matemática subjacente até a implementação prática em Java com código didático e recursos em português.

O principal objetivo do SVM é encontrar o **hiperplano ótimo** que separa duas classes de dados no espaço multidimensional. Esse hiperplano é determinado por um conjunto de pontos conhecidos como **vetores de suporte**, que são os pontos mais próximos da linha de separação e que definem a largura da margem.

Hiperplano Separador e Margem

O hiperplano é uma superfície que divide o espaço de características e é definido matematicamente pela seguinte fórmula:

w·x+b=0w \cdot x + b = 0w·x+b=0

Onde:

- www é o vetor de pesos (parâmetros do modelo),
- xxx representa o vetor de características (atributos) do dado,
- bbb é o termo de viés, que ajusta o deslocamento do hiperplano.

A margem MMM entre as duas classes é definida como:

M=2∥w∥M = \frac{2}{\|w\|}M=∥w∥2

Essa margem representa a distância entre os vetores de suporte mais próximos e o hiperplano.

O objetivo do SVM é maximizar MMM, ou seja, maximizar a separação entre as classes.

Otimização do Hiperplano

Para maximizar a margem, o SVM resolve um problema de **otimização convexa** que minimiza ∥w∥\|w\|∥w∥ sujeita às restrições de que todos os pontos sejam corretamente classificados:

yi(w·xi+b)≥1,∀iy_i (w \cdot x_i + b) \geq 1, \quad \forall iyi(w·xi+b)≥1,∀i

Onde:

- yiy_iyi é o rótulo da classe de xix_ixi (1 ou -1),
- xix_ixi são as amostras de treinamento.

Esse problema é resolvido por métodos de otimização, como os **Multiplicadores de Lagrange**, para encontrar os valores ótimos de www e bbb.

Kernel Trick (Truque do Kernel)

Quando os dados não são linearmente separáveis, o SVM recorre ao **truque do kernel** para mapear os dados para um espaço de maior dimensionalidade, onde se torna possível encontrar um hiperplano separador.

As principais funções de kernel são:

- **Linear**: Útil para dados linearmente separáveis.

  K(xi,xj)=xi·xjK(x_i, x_j) = x_i \cdot x_jK(xi,xj)=xi·xj

- **Polinomial**: Útil para dados que exibem uma relação polinomial.

  K(xi,xj)=(xi·xj+c)dK(x_i, x_j) = (x_i \cdot x_j + c)^dK(xi,xj)=(xi·xj+c)d

- **RBF (Radial Basis Function)**: Recomendado para dados não linearmente separáveis.

  K(xi,xj)=exp⁡(−γ∥xi−xj∥2)K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)K(xi,xj)=exp(−γ∥xi−xj∥2)

Configuração do SVM no Java
Passo a Passo

Para ilustrar a aplicação do SVM em Java, usaremos a biblioteca Smile, que fornece classes e métodos úteis para implementar e treinar modelos de machine learning, incluindo o SVM.

Dependências do Projeto

Adicione a dependência do Smile ao arquivo pom.xml do seu projeto Maven:

```
<dependency>
    <groupId>com.github.haifengl</groupId>
    <artifactId>smile-core</artifactId>
    <version>2.6.0</version>
</dependency>
```

Estrutura da Classe SVM

Criamos a classe ClassificadorSVM para definir e treinar o modelo SVM, a seguir está a implementação com variáveis, métodos e objetos em português.

```
import smile.classification.SVM;
import smile.math.kernel.GaussianKernel;

public class ClassificadorSVM {
    private SVM<double[]> modelo;

    // Método para treinar o modelo com kernel Gaussiano (RBF)
    public void treinarModelo(double[][] dados, int[] rotulos, double sigma, double custo) {
        // Definindo o kernel RBF com sigma e o parâmetro de regularização custo
        modelo = new SVM<>(new GaussianKernel(sigma), custo);
        modelo.learn(dados, rotulos); // Treinamento do modelo
        modelo.finish(); // Finalização do treinamento
    }

    // Método para prever a classe de novos dados
    public int preverClasse(double[] novoDado) {
        return modelo.predict(novoDado);
    }
}
```

Neste código:

- **modelo**: Instância da classe SVM com um kernel RBF.
- **treinarModelo**: Método que configura o kernel com um valor de σ\sigmaσ para definir a influência dos pontos, e o parâmetro de custo (regularização).
- **preverClasse**: Método para prever a classe de um novo dado com base no modelo treinado.

Parâmetros de Treinamento

- **sigma**: Define a largura do kernel RBF. Valores pequenos de sigma geram regiões de influência mais próximas dos vetores de suporte.
- **custo**: Parâmetro de regularização que controla o balanceamento entre maximizar a margem e minimizar o erro de classificação. Valores mais altos de custo tornam o modelo mais rígido.

Exemplo de Uso da Classe

Para usar o ClassificadorSVM, forneça dados e rótulos para o treinamento e faça previsões com novos dados:

```
public class ExemploUsoSVM {
    public static void main(String[] args) {
        double[][] dados = {
            {1.5, 2.0}, {2.0, 2.5}, {3.0, 3.5}, {5.5, 6.5}, {6.0, 6.0}
        };
        int[] rotulos = {1, 1, -1, -1, 1};

        // Inicializando o classificador
        ClassificadorSVM classificador = new ClassificadorSVM();
        classificador.treinarModelo(dados, rotulos, 0.5, 1.0);

        // Predição para um novo ponto
        double[] novoDado = {4.0, 5.0};
        int predicao = classificador.preverClasse(novoDado);

        System.out.println("Classe prevista para o ponto [4.0, 5.0]: " + predicao);
    }
}
```

Neste exemplo:

- O ClassificadorSVM é treinado com dados de exemplo, e então utilizado para prever a classe de um novo ponto.

Avaliação do Modelo

Para avaliar o desempenho do modelo SVM, podemos utilizar métricas como **acurácia**, **precisão**, e **recall**, dependendo do problema específico. No exemplo abaixo, calculamos a acurácia:

```
import smile.validation.Accuracy;

public double calcularAcuracia(int[] rotulos, double[][] dados) {
    return Accuracy.of(rotulos, modelo.predict(dados));
}
```

Esta função compara os rótulos verdadeiros com as previsões do modelo e calcula a proporção de previsões corretas.

Boas Práticas para o uso de SVM

1. **Pré-processamento de Dados**: Escale as variáveis (normalização ou padronização) para evitar que uma característica tenha mais influência que outras.
2. **Escolha do Kernel**: Se os dados não são linearmente separáveis, escolha um kernel adequado (como RBF).
3. **Ajuste de Hiperparâmetros**: Use validação cruzada para otimizar os valores de σ\sigmaσ e do custo.
4. **Atenção ao Overfitting**: Parâmetros de regularização muito altos podem fazer o modelo ajustar-se aos dados de treino, reduzindo a generalização.

O Support Vector Machine é uma técnica poderosa e flexível para classificação e regressão.

Este artigo abordou desde os fundamentos matemáticos do SVM, passando pelos principais conceitos e pela implementação em Java, até dicas de uso em produção.

Em problemas de alta dimensionalidade e com dados complexos, o SVM continua sendo uma escolha confiável e altamente eficaz para profissionais de machine learning.

EducaCiência FastCode para a comunidade