Instituto Tecnológico de Culiacán

Ingeniería en Sistemas Computacionales

Materia:

Inteligencia Artificial.

Tarea:

SVM Radial.

Alumno:

Flores Saldaña Martin Alejandro

Docente:

Jose Mario Rios Felix

Grupo:

18:00-19:00

1. Introducción al Problema

Las SVM estándar (lineales) funcionan encontrando un "hiperplano" que separa dos clases de datos. Sin embargo, en el mundo real, los datos raramente son separables linealmente. A menudo tienen estructuras complejas, como anillos concéntricos o grupos irregulares.

Aquí es donde entra el Kernel RBF (Radial Basis Function). Este permite a la SVM clasificar datos que no se pueden separar con una línea recta, proyectándolos matemáticamente a una dimensión superior.

2. El "Kernel Trick" y la Función Radial

La magia del RBF reside en el "Truco del Kernel" (Kernel Trick). En lugar de calcular costosas coordenadas en un espacio de dimensiones superiores, la SVM calcula la similitud entre dos puntos directamente en el espacio original usando una fórmula matemática.

La Fórmula Matemática

El kernel RBF calcula la similitud entre dos puntos de datos, x y x', basándose en la distancia euclidiana entre ellos. La fórmula es:

$$K(x, x') = \exp\left(-\gamma\left||x - x'|\right|^2\right)$$

Donde:

- $\left||x - x'|\right|^2$: Es la distancia euclidiana al cuadrado entre dos puntos.
- γ (Gamma): Es un parámetro que controla la "anchura" de la campana gaussiana (explicado más abajo).
- exp: Es la función exponencial.

Intuición: El valor del Kernel varía entre 0 y 1. Si dos puntos están muy cerca, la distancia es casi 0 y el resultado es cercano a 1 (son muy similares). Si están lejos, el resultado tiende a 0.

3. Hiperparámetros Críticos

El éxito de un modelo SVM con RBF depende casi enteramente del ajuste de dos hiperparámetros: C y Gamma (γ).

A. Gamma (γ)

Define hasta dónde llega la influencia de un solo ejemplo de entrenamiento.

- Gamma Bajo: La "campana" es ancha. Los puntos alejados tienen influencia. El límite de decisión es suave y generalizado.
- Gamma Alto: La "campana" es estrecha. Solo los puntos muy cercanos importan. El límite de decisión se ajusta mucho a los datos de entrenamiento, creando "islas" alrededor de los puntos.
    - Riesgo: Un gamma muy alto causa Overfitting (sobreajuste).

B. C (Regularización)

Controla el compromiso entre tener un límite de decisión suave y clasificar correctamente todos los puntos de entrenamiento.

- C Bajo: Permite errores de clasificación en el entrenamiento para buscar un margen más amplio y simple.
- C Alto: Intenta clasificar todos los puntos de entrenamiento correctamente a toda costa, resultando en un margen estrecho y complejo.

4. Ventajas y Desventajas del RBF

Ventajas

- Versatilidad: Puede modelar límites de decisión no lineales muy complejos.
- Pocos Parámetros: Solo requiere ajustar C y γ para obtener buenos resultados.
- Robustez: Funciona muy bien en espacios de alta dimensión.

Desventajas

- Escalabilidad: Es computacionalmente costoso en datasets muy grandes ($O(n^2)$ o $O(n^3)$).
- Caja Negra: Es difícil interpretar geométricamente qué está haciendo el modelo en dimensiones altas.

- Sensibilidad: Es muy sensible a datos no escalados (requiere normalización previa).

5. Nota Importante: Escalado de Características

Dado que el kernel RBF se basa en la distancia euclidiana, es fundamental que todas las características (features) estén en la misma escala.

Ejemplo: Si tienes una variable "Edad" (0-100) y otra "Salario" (1000-100000), la distancia estará dominada totalmente por el salario, haciendo que la edad sea irrelevante para el modelo. Siempre debes aplicar `StandardScaler` o `MinMaxScaler` antes de usar SVM RBF.

6. Ejemplo de Implementación (Python/Scikit-Learn)

```python
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

# 1. Crear el pipeline con escalado y el modelo
# kernel='rbf' es el default, pero lo ponemos explícito.
modelo = make_pipeline(
    StandardScaler(),
    SVC(kernel='rbf', C=1.0, gamma='scale')
)

# 2. Entrenar
# X_train son tus datos, y_train tus etiquetas
modelo.fit(X_train, y_train)

# 3. Predecir
predicciones = modelo.predict(X_test)
```