Minor in AI

Support Vector Machines Demystified

1 From Gardens to Algorithms: Iris Flower

Imagine you're handed three types of iris flowers (Setosa, Versicolor, Virginica) and asked to classify them using only two measurements: **sepal length** and **sepal width**. This real-world problem mirrors the classic Iris dataset, where machines learn to distinguish species through pattern recognition.

Why SVM?

Traditional rulers fail when data isn't linearly separable. SVMs act like **smart flexible rulers** that can bend their decision boundaries using kernel tricks. The RBF (Radial Basis Function) kernel is particularly powerful—it transforms the 2D plane into a higher-dimensional space where separation becomes possible.

2 The Science Behind the Magic

2.1 Gamma: The Neighborhood Watch Parameter

In the context of the RBF kernel, γ determines how "local" or "global" the influence of each training point is when shaping the decision boundary.

- **High gamma ($\gamma = 1.0$) – *Microscope mode*** Each point's influence is sharply localized. Only very close neighbors contribute meaningfully to the kernel value, leading to a wiggly boundary that can overfit small fluctuations.
- **Low gamma ($\gamma = 0.01$) – *Satellite view*** Each point's influence spreads broadly across the feature space. Distant points still have non-negligible effect, producing a smoother, more global decision surface.

Real–World Analogy: Drop a pebble in a pond and observe the ripples. Gamma controls how far those ripples travel. A high gamma is like a thick, viscous fluid where ripples die out quickly (only very close points "feel" each other). A low gamma is like water where ripples travel far, connecting distant regions.

$$\text{Influence between two points} \propto e^{-\gamma \times (\text{distance})^2}$$

Here, "distance" is the straight-line (Euclidean) separation between two points in your feature space.

2.2 RBF Kernel Deep Dive

The Radial Basis Function (RBF) kernel computes similarity $K(x_i, x_j)$ between any two feature vectors x_i and x_j as:

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right)$$

Let us unpack each component:

- $\|x_i - x_j\|^2$: The squared Euclidean distance

  $$\|x_i - x_j\|^2 = (x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots$$

  This measures how far apart the two points are in the original space.

- $-\gamma$: A scaling factor in the exponent A larger γ makes the exponent more negative for a given distance, causing the similarity to drop off more rapidly.
- $\exp(\cdot)$: The exponential function Transforms the negative distance into a similarity score between 0 and 1:

$$K(x_i, x_j) \to \begin{cases} 1, & \text{if } x_i = x_j \quad \text{(zero distance)} \\ 0, & \text{as } \|x_i - x_j\| \to \infty \end{cases}$$

Interpretation: The RBF kernel effectively creates a high-dimensional feature space where each original point x_i is represented by a "bump" centered at that point. The height of the bump at another point x_j is exactly $K(x_i, x_j)$. SVM then finds a linear hyperplane in this transformed space, which corresponds to a non-linear boundary back in the original space.

With an appropriate choice of γ (and regularization C), the RBF kernel equips SVM with the flexibility to carve out complex decision regions—much like a sculptor shaping clay into intricate forms.

Spreadsheet Insight

In our lecture demo, we calculated pairwise distances between 3 points:

Points	Squared Distance	RBF Value (=0.1)
A-B	2.0	$e^{-0.1*2.0} \approx 0.8187$
A-C	5.0	$e^{-0.1*5.0} \approx 0.6065$

This creates a **similarity matrix** that guides boundary creation.

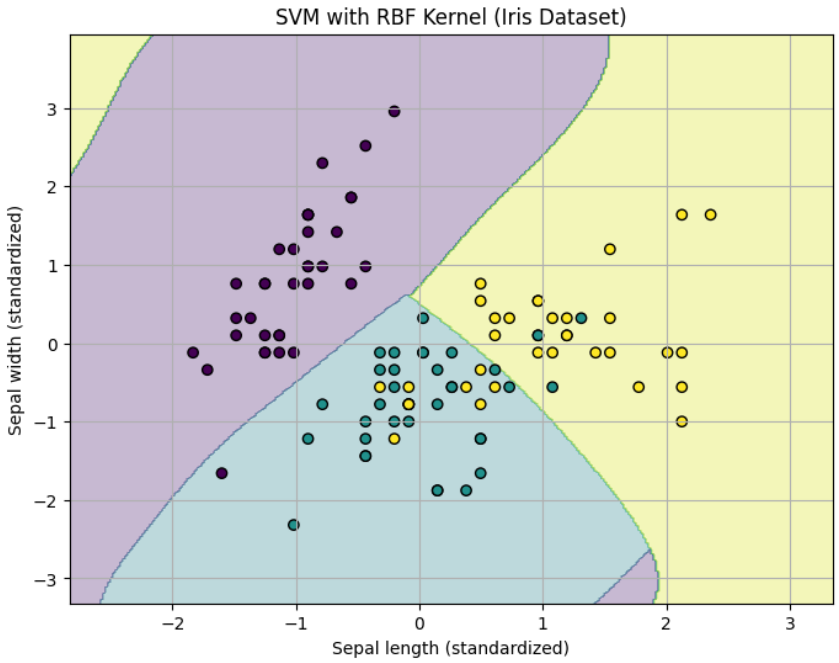
3 Hands-On: Classifying Flowers

Listing 1: Iris Classification Code

```
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler

# Load data
iris = datasets.load_iris()
X = iris.data[:, :2]  # Sepal features only
y = iris.target

# Standardization is crucial!
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Build RBF-SVM model
model = SVC(kernel='rbf', gamma='auto')
model.fit(X_scaled, y)

# Visualize decision boundaries
plot_decision_boundary(X_scaled, y, model)
```

Step-by-Step Explanation:

1. **Feature Selection**: Using only 2 features (sepal length/width) enables 2D visualization
2. **Standardization**: SVMs are distance-based—scaling prevents dominance by large-valued features
3. **Gamma='auto'**: Lets sklearn choose $\gamma = 1/(\text{n\_features} \times \text{X.var}())$
4. **Decision Boundary Plot**: Shows regions where model predicts each class



Complex RBF boundaries separating three iris classes

4 When Penguins Challenge Machines

The Palmer Penguins dataset introduces new real-world hurdles for our SVM classifier:

- **Missing values**: We must clean the data before training.
- **Overlapping classes**: Different species may share similar measurements.
- **Biological variation**: Natural variability requires robust decision boundaries.

Listing 2: Penguin Classification with SVM

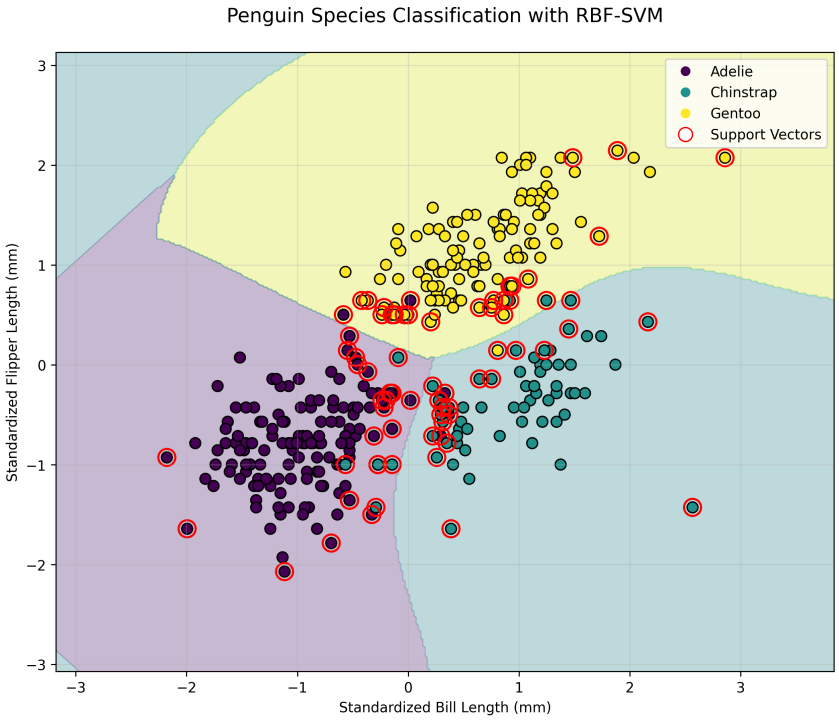
```
# 1. Load and clean the dataset
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
import matplotlib.pyplot as plt

```

```
penguins = sns.load_dataset("penguins")      # Load Palmer Penguins
penguins = penguins.dropna()                 # Remove rows with missing
    values

# 2. Select features and encode labels
X = penguins[['bill_length_mm', 'flipper_length_mm']]
y = LabelEncoder().fit_transform(penguins['species'])
# species     integers 0, 1, 2

# 3. Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# 4. Scale features to zero mean & unit variance
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)

# 5. Train an  R B F kernel  SVM
#    gamma='scale'     1 / (n_features * var(X_train))
model = SVC(kernel='rbf', gamma='scale', C=1.0)
model.fit(X_train_scaled, y_train)

# 6. Visualize support vectors on training data
plt.figure(figsize=(6,6))
# Plot all training points (faded by species)
plt.scatter(X_train_scaled[:,0], X_train_scaled[:,1],
            c=y_train, cmap='viridis', alpha=0.5, edgecolors='k')

# Overlay support vectors with bold circles
plt.scatter(model.support_vectors_[:,0], model.support_vectors_[:,1],
            s=120, facecolors='none',
            edgecolors='red', linewidths=2,
            label='Support Vectors')

plt.xlabel('Bill Length (standardized)')
plt.ylabel('Flipper Length (standardized)')
plt.title('SVM Support Vectors on Penguin Features')
plt.legend()
plt.grid(True)
plt.show()
```

Key Differences from Iris Dataset

- **Data Cleaning**: We drop missing values with `.dropna()`, ensuring no invalid rows are used.
- **Categorical Encoding**: Penguin species are converted from strings to integers (0, 1, 2) via `LabelEncoder`.
- **Feature Scaling**: Standardization ($\mu = 0$, $\sigma = 1$) is critical for SVM to treat each feature equally.
- **Gamma = 'scale'**: Automatically sets $\gamma = 1/(n\_features \times \text{Var}(X))$, a sensible default for most problems.

- **Support Vectors**: These are the critical training points that lie exactly on the margin boundaries and define the classifier.



SVM decision boundaries and support vectors for penguin species classification

Interpreting Support Vectors

- **Position**: Always lie on or between the two margin boundaries.
- **Significance**: Removing any single support vector will alter the optimal hyperplane.
- **Visual Clue**: In our plot, support vectors are marked by bold red circles.

5 Expert-Level Tips

5.1 When to Use the RBF Kernel

- **Pros**:
  - Can carve out highly complex, non-linear decision boundaries.
  - Automatically adapts to clusters of varying shapes and densities.
- **Cons**:
  - More computationally intensive than linear or low-degree polynomial kernels.
  - Requires careful tuning of γ and C to avoid overfitting or underfitting.
- **Sweet Spot**: Medium-sized datasets (up to 10,000 samples) where clear but non-linear clusters exist, and training time remains practical.

5.2 Gamma in Practice

Gamma Value	Decision Boundary	Risk / Behavior
$\gamma > 1$	Highly wiggly, tight around each point	Overfit: memorizes noise and outliers
Optimal ($\approx$ "scale" or tuned)	Smooth, well-curved margins	Good trade-off: generalizes to new data
$\gamma < 0.01$	Almost linear, broad margins	Underfit: misses subtle patterns

5.3 Debugging SVM Performance

1. **Always scale features.** Use `StandardScaler` or `MinMaxScaler` so all dimensions contribute equally.
2. **Begin with `gamma='scale'`.** This default adapts γ to $\frac{1}{n\_features \times \text{Var}(X)}$.
3. **If overfitting:**
   - Decrease γ to smooth the boundary.
   - Increase C to allow a wider margin (more slack).
4. **If underfitting:**
   - Increase γ for tighter local decision regions.
   - Consider a polynomial kernel (`kernel='poly'`) with degree 2–3.
5. **Monitor support vectors.** Too many support vectors often indicate a complex (potentially overfit) model.

6 Key Takeaways

SVM Essentials

- **Kernel Choice**: `rbf` for complex patterns; `linear` for simple, high-dimensional data.
- **Gamma Wisdom**: Start with `gamma='scale'` or automated heuristics before manual tuning.
- **Visual Reality**: RBF creates "warped" boundaries that can be hard to interpret—always validate on a hold-out set.
- **Data Preparation**: Handle missing values, encode categorical labels, and standardize all features.

"SVM with RBF is like teaching AI origami - it folds feature space to separate classes!"