Minor in AI

SVM: Classifying the Unclassifiable

1 The Magic of Separation: Spam or Ham?

Imagine your phone buzzing with a new message: "Congrats! You won $1,000,000. Click here to claim!" Is this legitimate or spam? This is exactly what Support Vector Machines (SVM) can solve! SVMs are powerful algorithms that find optimal boundaries between different categories of data.

Consider email classification:

- **Problem:** Separate genuine emails (ham) from spam
- **Challenge:** Messages don't come with clear labels
- **SVM Solution:** Finds the best dividing line using key examples

But what about more complex patterns? Like distinguishing between galaxy spirals in astronomy? Traditional straight-line classifiers fail here. This is where SVM's true power emerges - it can find curved boundaries in complex data landscapes.

2 The Geometry of Separation

2.1 Core Idea: Maximum Margin

SVM creates the widest possible "safety buffer" between classes:

$$\text{Margin Width} = \frac{2}{\|\mathbf{w}\|}$$

where $\mathbf{w}$ is the weight vector. Maximizing this margin improves generalization.

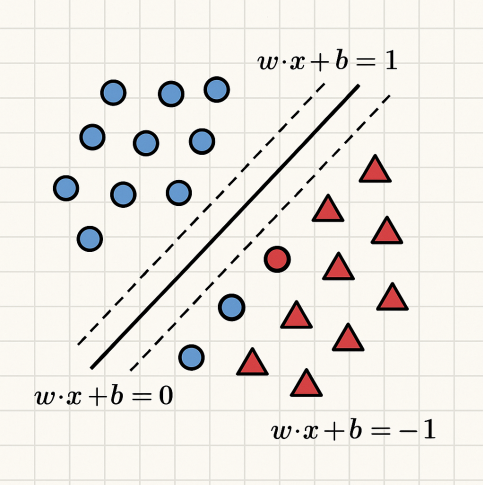

Figure 1: SVM creates the widest possible separation between classes

2.2 Support Vectors: The Heroes

These are the critical data points that "support" the decision boundary:

- Lie closest to the separating hyperplane
- Determine the margin's position and orientation
- If removed, the decision boundary changes

2.3 The Kernel Trick: Seeing in Higher Dimensions

When data isn't linearly separable (like our spiral galaxy data), SVM uses kernel functions to project data into higher dimensions:

$$K(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)$$

Common kernels:

Kernel	Use Case
Linear	Simple separations
Polynomial	Curved boundaries
RBF (Radial Basis Function)	Complex patterns like spirals

3 Galaxy Classification: A Cosmic Case Study

3.1 Creating Cosmic Data

We generate synthetic spiral data mimicking galaxy patterns:

Listing 1: Generating Spiral Data

```
import numpy as np

def generate_spiral(n_points, noise=0.5):
    n = np.sqrt(np.random.rand(n_points)) * 780 * (2*np.pi)/360
    d1x = -np.cos(n)*n + np.random.rand(n_points) * noise
    d1y = np.sin(n)*n + np.random.rand(n_points) * noise
    # ... (full code from lecture_code.py)
    return X, y

X, y = generate_spiral(200, noise=0.5)
```

This creates two interleaved spirals (Class 0 = blue, Class 1 = red) with controlled noise.

3.2 Why Linear Models Fail

Traditional logistic regression creates a straight boundary that fails spectacularly:

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X, y)
# Visualize poor performance
```

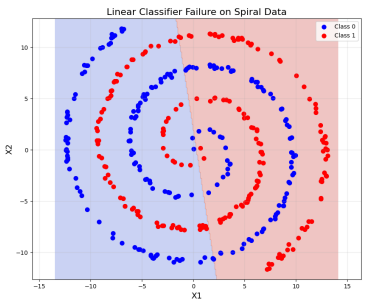

Figure 2: Linear classifier fails on spiral data

3.3 RBF Kernel to the Rescue

We project 2D data into 3D using Radial Basis Function:

```
def rbf_lift(X, gamma=0.1, center=np.array([0,0])):
    diff = X - center
    squared_dist = np.sum(diff**2, axis=1)
    z = np.exp(-gamma * squared_dist)
    return z
```

The transformation creates a 3D landscape where classes become separable:

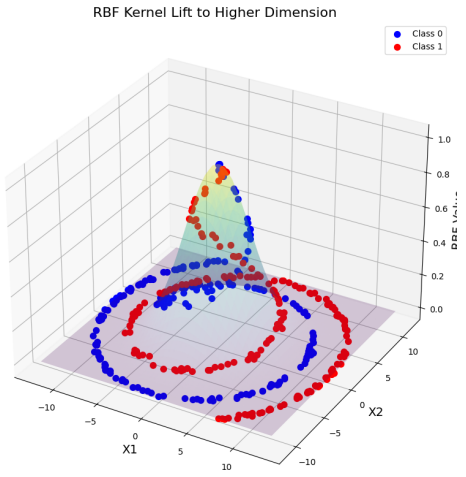

Figure 3: Spiral data lifted to 3D via RBF kernel

3.4 SVM in Action

Implement SVM with RBF kernel and visualize results:

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

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
clf = SVC(kernel='rbf', gamma='auto')
clf.fit(X_scaled, y)

# Visualize decision boundaries and support vectors
```

Key parameters:

- `gamma`: Controls influence range of single points
- `C`: Regularization parameter (balance accuracy vs. simplicity)

4 Real-World Applications

4.1 SMS Spam Detection

Practical implementation with TF-IDF vectorization:

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC

# Load SMS dataset
vectorizer = TfidfVectorizer(stop_words='english')
X_train_tfidf = vectorizer.fit_transform(X_train)

clf = SVC(kernel='linear')
clf.fit(X_train_tfidf, y_train)
# Achieves >98% accuracy!
```

4.2 Diabetes Prediction

Using the Pima Indians Diabetes Database:

```python
from sklearn.svm import SVC

# Load diabetes data
clf = SVC(kernel='rbf')
clf.fit(X_train, y_train)

# Compare kernel performance:
# Linear: 70% accuracy
# RBF: 77% accuracy
```

Pro Tip: Kernel Selection

Start simple:
1. Try linear kernel first
2. If accuracy low, switch to RBF
3. Tune gamma and C parameters
4. Visualize decision boundaries

5 Key Takeaways: The SVM Advantage

1. **Margin Maximization:** Creates robust decision boundaries resistant to noise
2. **Kernel Magic:** Handles non-linear data through dimension lifting
3. **Support Vectors:** Only critical points affect the model (memory efficient)
4. **Versatility:** Works with images, text, biological data, astronomy
5. **Parameter Tuning:** Gamma and C dramatically affect performance

Remember

"Support Vector Machines find the widest possible road between classes, using the most critical data points as guardrails."

6 SVM Exploration Kit

Continue experimenting with:

- **Datasets:** - SMS Spam Collection: `https://archive.ics.uci.edu/dataset/228/sms+spam+collection` - Pima Indians Diabetes: `https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database`
- **Interactive Tools:** - SVM Playground: `https://www.cs.cmu.edu/~svmplay/` - Kernel Visualization: `https://scikit-learn.org/stable/auto_examples/svm/plot_iris_svc.html`