Lecture 6: k-Nearest Neighbors and Support Vector Machines

Luu Minh Sao Khue

Distance-based and margin-based classification — intuition, mathematics, kernels, and visualization of decision boundaries.

1. Learning Objectives

After this lecture, you will be able to:

- Explain the intuition behind instance-based (k-NN) and margin-based (SVM) learning.
- Derive distance metrics and understand their effects on classification.
- Explain how k and data dimensionality influence k-NN.
- Understand SVM formulation: hyperplane, margin, support vectors.
- Distinguish hard-margin and soft-margin SVM and interpret the regularization parameter C.
- Apply kernel trick for nonlinear decision boundaries.
- Implement and visualize k-NN and SVM classifiers in Python.

2. Intuition

k-Nearest Neighbors (k-NN) classifies a new sample by comparing it with the most similar points in the training set. The idea: "Tell me who your neighbors are, and I'll tell you who you are." It is a simple but powerful **non-parametric** method — it learns nothing explicitly until a query arrives.

Support Vector Machine (SVM) aims to find a decision boundary that separates classes with the *maximum margin*. It chooses the hyperplane that is farthest from the nearest data points (support vectors), which leads to good generalization.

Analogy: k-NN uses *local voting* for classification; SVM finds the *global optimal boundary*.

3. k-Nearest Neighbors (k-NN)

3.1 Basic Definition

Given a dataset $\{(x_i, y_i)\}_{i=1}^n$ with $x_i \in \mathbb{R}^p$ and $y_i \in \{1, \ldots, K\}$, the k-NN prediction for a new point x is:

$$\hat{y} = \arg\max_c \sum_{i \in N_k(x)} \mathbb{I}(y_i = c),$$

where $N_k(x)$ is the set of indices of the k closest training samples to x, and $\mathbb{I}$ is the indicator function.

3.2 Distance Metrics

To define "closest," we use a distance function $d(x_i, x_j)$:

- **Euclidean distance:**

$$d_E(x_i, x_j) = \sqrt{\sum_{l=1}^{p} (x_{il} - x_{jl})^2}.$$

- **Manhattan (L1) distance:**

$$d_M(x_i, x_j) = \sum_{l=1}^{p} |x_{il} - x_{jl}|.$$

- **Minkowski distance (general form):**

$$d_p(x_i, x_j) = \left(\sum_{l=1}^{p} |x_{il} - x_{jl}|^p \right)^{1/p}.$$

The choice of distance depends on feature scaling; features must be standardized to avoid dominance by large-valued features.

3.3 Effect of k and Curse of Dimensionality

- Small k: low bias, high variance — sensitive to noise.
- Large k: high bias, low variance — smoother decision boundary.

Curse of dimensionality. As dimensionality p increases, all points become far apart:

$$\frac{\max d(x_i, x_j) - \min d(x_i, x_j)}{\min d(x_i, x_j)} \to 0.$$

Distances lose meaning, making k-NN less effective in high dimensions. Dimensionality reduction or feature selection helps mitigate this.

3.4 Weighted k-NN

Instead of equal votes, weight neighbors inversely by distance:

$$\hat{y} = \arg\max_c \sum_{i \in N_k(x)} w_i \mathbb{I}(y_i = c), \quad w_i = \frac{1}{d(x, x_i)^2}.$$

4. Support Vector Machines (SVM)

4.1 Linear SVM for Binary Classification

We seek a hyperplane:

$$w^T x + b = 0,$$

that separates the data with the largest possible **margin**. For each sample (x_i, y_i), $y_i \in \{-1, +1\}$.

The signed distance from x_i to the hyperplane is:

$$\frac{y_i(w^T x_i + b)}{\|w\|}.$$

4.2 Hard-Margin SVM

We maximize the margin subject to perfect separation:

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|^2, \\ \text{s.t.} \quad & y_i(w^T x_i + b) \geq 1, \ \forall i. \end{aligned}$$

The margin width is $2/\|w\|$.

Support vectors. Points satisfying $y_i(w^T x_i + b) = 1$ lie on the margin boundaries and define the decision surface.

4.3 Soft-Margin SVM

If perfect separation is impossible, introduce slack variables $\xi_i \geq 0$:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i, \\ \text{s.t.} \quad & y_i(w^T x_i + b) \geq 1 - \xi_i. \end{aligned}$$

The regularization parameter $C > 0$ controls the trade-off:

- Large C: prioritize small training error (less regularization).
- Small C: allow more violations for a wider margin.

4.4 Dual Form and Kernel Trick

The dual optimization problem is:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j), \\ \text{s.t.} \quad & \sum_{i} \alpha_i y_i = 0, \quad 0 \le \alpha_i \le C. \end{aligned}$$

Here $K(x_i, x_j) = x_i^T x_j$ for linear SVM. Replacing it with nonlinear kernels maps inputs to a higher-dimensional feature space:

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j).$$

Common kernels:

- Linear: $K(x_i, x_j) = x_i^T x_j$
- Polynomial: $K(x_i, x_j) = (x_i^T x_j + c)^d$
- RBF (Gaussian): $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$

The decision function for new input x:

$$f(x) = \text{sign}\left(\sum_{i} \alpha_i y_i K(x_i, x) + b \right).$$

4.5 Geometric Interpretation

The margin is the distance between two parallel hyperplanes:

$$H_1 : \ w^T x + b = 1, \quad H_2 : \ w^T x + b = -1.$$

SVM seeks w minimizing $\|w\|$ while keeping all samples outside the margin. Only support vectors (points on H_1 or H_2) determine the boundary; non-support vectors can be removed without changing the solution.

5. Bias–Variance Trade-off

Model	Bias	Variance
k-NN (small k)	Low	High
k-NN (large k)	High	Low
SVM (large C)	Low	High
SVM (small C)	High	Low

Cross-validation helps choose k, kernel, γ, and C for optimal generalization.

6. Python Implementation: k-NN and SVM

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

# Generate 2D data for visualization
X, y = make_classification(n_samples=200, n_features=2,
                           n_redundant=0, n_informative=2,
                           n_clusters_per_class=1, random_state=42)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# k-NN
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn.fit(X_train, y_train)

# SVM with RBF kernel
svm = SVC(kernel='rbf', C=1.0, gamma='scale')
svm.fit(X_train, y_train)

print("k-NN accuracy:", knn.score(X_test, y_test))
print("SVM accuracy:", svm.score(X_test, y_test))

# Decision boundary visualization
def plot_boundary(model, title):
    xx, yy = np.meshgrid(np.linspace(-3,3,200), np.linspace(-3,3,200))
    Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.3)
    plt.scatter(X_train[:,0], X_train[:,1], c=y_train, edgecolor='k')
    plt.title(title)
    plt.show()

plot_boundary(knn, "k-NN Decision Boundary (k=5)")
plot_boundary(svm, "SVM Decision Boundary (RBF kernel)")
```

7. Practical Tips

- Always standardize features for both k-NN and SVM.
- Tune hyperparameters (k, C, γ) with cross-validation.
- Use distance-weighted k-NN for noisy data.

- Use linear SVM for high-dimensional sparse data (e.g., text).
- RBF kernel handles complex nonlinear boundaries.
- For large datasets, approximate SVM (e.g., linear SVC) for efficiency.

8. Summary

- k-NN: non-parametric, relies on distance metric and k choice.
- Sensitive to feature scale and dimensionality.
- SVM: margin-based, robust, works well with kernel trick.
- Regularization parameter C balances margin width and misclassifications.
- Both models benefit from cross-validation and feature scaling.
- Visualization helps interpret decision boundaries.

9. Exercises

1. Compute Euclidean and Manhattan distances between two 3D points.
2. Implement k-NN from scratch using NumPy.
3. Plot classification accuracy versus k for k-NN.
4. Derive the margin width for an SVM with $\|w\| = 4$.
5. Explain how the parameter C affects bias and variance.
6. Visualize SVM with linear and RBF kernels on a 2D dataset.
7. Compare performance of k-NN and SVM on the Iris dataset.