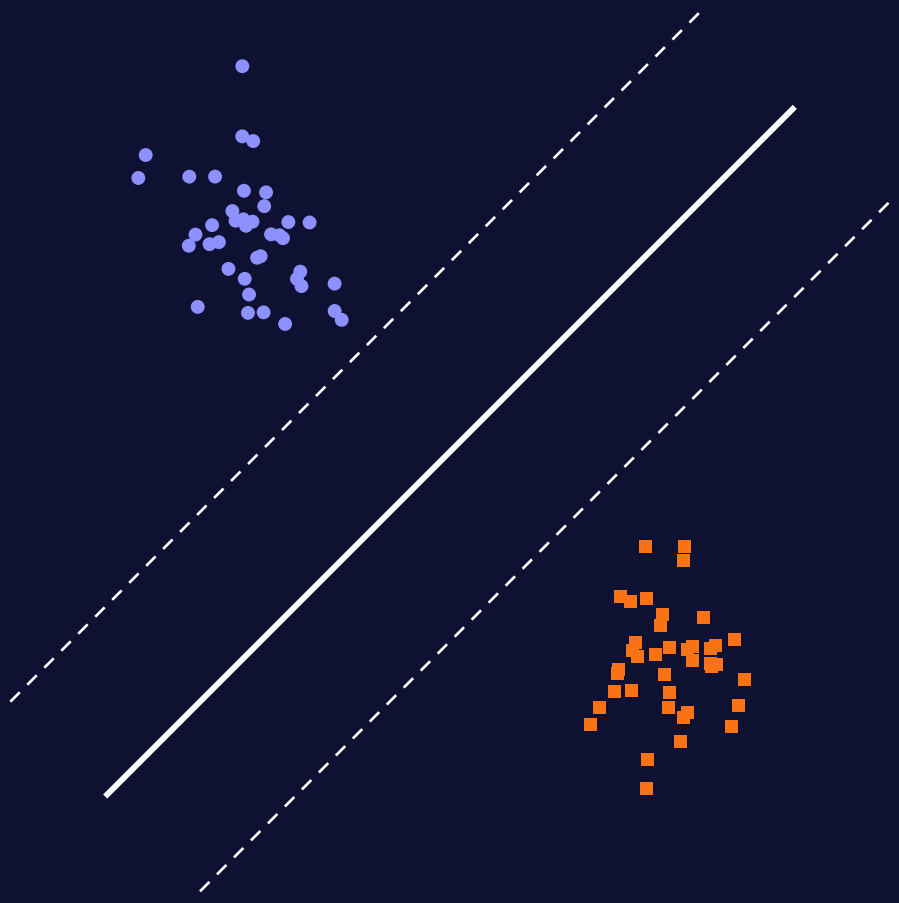
Support Vector Machine

Notes, Python code and interview questions



Code and Debug

codeanddebug.in/blog | YouTube: @codeanddebug

1. The idea

A support vector machine (SVM) separates two classes with the widest possible gap between them. Many lines can split two groups of points; SVM picks the one that stays furthest from both. Picture the widest road between two villages: the middle of the road is the decision boundary and the two edges are the margin.

The points standing on the edges of the road (or inside it, or on the wrong side) are the **support vectors**. They are the only points that decide where the boundary goes. Delete every other training point and the model does not change.

2. Key terms

Term	Meaning
Hyperplane	The decision boundary `w · x + b = 0`. A line in 2D, a plane in 3D.
Score f(x)	`f(x) = w · x + b`. The sign gives the class, the size gives the distance from the boundary.
Margin	The empty band between `f(x) = +1` and `f(x) = −1`. Width = `2 / ||w||`.
Support vectors	Points on or inside the margin. Only they have non-zero weight.
Hard margin	No point may sit inside the margin. Only works for perfectly separable data.
Soft margin	Some points may break the margin, at a price set by C.

3. C and gamma cheat sheet

Setting	What happens	Risk
Small C (e.g. 0.01)	Wide margin, many support vectors, forgives mistakes	Underfitting
Large C (e.g. 100)	Narrow margin, tries to classify every training point	Overfitting noisy data
Small gamma (RBF)	Each point influences a large area, smooth boundary	Underfitting
Large gamma (RBF)	Each point influences a tiny area, wiggly boundary	Overfitting
High C + high gamma	Boundary wraps around single points	Strong overfitting

4. Kernels

Kernel	Formula	Use it when
Linear	`a · b`	Data is roughly linearly separable, or there are many features (text)
Polynomial	`(γ a · b + r)^d`	Feature interactions matter; degree 2 or 3
RBF (Gaussian)	`exp(−γ ||a − b||²)`	Default when you do not know the shape of the data
Sigmoid	`tanh(γ a · b + r)`	Rarely used

The kernel trick: the SVM dual only needs dot products between points. A kernel computes that dot product as if the points were mapped to a higher-dimensional space, without building the new features. A straight boundary in that space looks curved in the original one.

5. The maths in three lines

Hard margin (labels y = +1 or −1):

$$\text{minimise } \tfrac{1}{2}||w||^2 \text{ such that } y_i\,(w \cdot x_i + b) \geq 1 \text{ for every } i$$

Soft margin with hinge loss:

$$\tfrac{1}{2}||w||^2 + C \cdot \Sigma \max(0,\ 1 - y_i\,(w \cdot x_i + b))$$

Dual form, where kernels come in ($0 \leq \alpha_i \leq C$, $\Sigma\ \alpha_i\, y_i = 0$):

$$\text{maximise } \Sigma\ \alpha_i - \tfrac{1}{2}\ \Sigma\ \Sigma\ \alpha_i\ \alpha_j\ y_i\ y_j\ K(x_i, x_j)$$

Prediction: $f(x) = \Sigma\ \alpha_i\ y_i\ K(x_i, x) + b$. Points with $\alpha_i > 0$ are the support vectors. scikit-learn solves the dual with SMO.

6. How to choose kernel, C and gamma

- Scale features with `StandardScaler` inside a `Pipeline`.
- Many features compared with rows (e.g. text): start with a linear kernel or `LinearSVC`.
- Otherwise start with RBF and `gamma="scale"`.
- Grid search C over 0.01 to 100 and gamma over 0.001 to 10 on a log scale.
- Train accuracy much higher than validation: lower gamma or C. Both low: raise them.
- Imbalanced classes: `class_weight="balanced"`. Need probabilities: `probability=True` (slower).

7. SVM vs logistic regression

	SVM	**Logistic regression**
Goal	Widest margin	Best probability estimates
Loss	Hinge loss	Log loss
Output	Class and distance score	Probability 0 to 1
Curved boundaries	Yes, with kernels	Only with manual features
Large datasets	Kernel SVM slows down	Scales to millions
Interpretability	Harder with kernels	Easy

8. Code examples (all tested)

Your first SVM: will the student pass?

PYTHON

```
import numpy as np
from sklearn.svm import SVC

# Hours studied and attendance (%) for 12 students, and whether they passed (1) or failed (0)
X = np.array([
    [1, 40], [2, 55], [2, 35], [3, 50], [4, 45], [3, 30],   # failed
    [6, 75], [7, 80], [5, 85], [8, 70], [7, 90], [6, 65],   # passed
])
y = np.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

model = SVC(kernel="linear", C=1.0)
model.fit(X, y)

print("Weights (w):", model.coef_[0].round(3))
print("Bias (b):", model.intercept_.round(3))
print("Support vectors per class:", model.n_support_)
print("Support vectors:\n", model.support_vectors_)

new_students = np.array([[5, 62], [6, 50]])  # 5 hrs / 62% and 6 hrs / 50%
print("Predictions:", model.predict(new_students))
print("Distance score:", model.decision_function(new_students).round(2))
```

OUTPUT

```
Weights (w): [0.069 0.172]
Bias (b): [-10.621]
Support vectors per class: [1 1]
Support vectors:
 [[ 2. 55.]
 [ 6. 65.]]
Predictions: [1 0]
Distance score: [ 0.41 -1.59]
```

What C does to the margin

PYTHON

```
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

# Two groups that overlap a little
X, y = make_blobs(n_samples=300, centers=2, cluster_std=2.2, random_state=7)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=7)

for C in [0.01, 1, 100]:
    model = SVC(kernel="linear", C=C).fit(X_train, y_train)
    margin = 2 / np.linalg.norm(model.coef_[0])
    print(f"C={C:<5} support vectors={len(model.support_):<4}"
          f"margin width={margin:.2f}  train acc={model.score(X_train, y_train):.3f}"
          f"  test acc={model.score(X_test, y_test):.3f}")
```

OUTPUT

```
C=0.01  support vectors=47  margin width=4.39  train acc=0.962  test acc=0.956
C=1     support vectors=16  margin width=1.33  train acc=0.971  test acc=0.933
C=100   support vectors=12  margin width=0.86  train acc=0.971  test acc=0.933
```

Linear vs polynomial vs RBF on circles

PYTHON

```
from sklearn.datasets import make_circles
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

# One class forms a ring around the other: no straight line can split them
X, y = make_circles(n_samples=400, factor=0.4, noise=0.08, random_state=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

for kernel in ["linear", "poly", "rbf"]:
    model = SVC(kernel=kernel, degree=2, C=1.0).fit(X_train, y_train)
    print(f"{kernel:<7} test accuracy = {model.score(X_test, y_test):.3f}")
```

OUTPUT

```
linear  test accuracy = 0.408
poly    test accuracy = 1.000
rbf     test accuracy = 1.000
```

What gamma does (overfitting)

PYTHON

```
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

X, y = make_moons(n_samples=400, noise=0.25, random_state=3)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=3)

for gamma in [0.1, 1, 10, 100]:
    model = SVC(kernel="rbf", C=1.0, gamma=gamma).fit(X_train, y_train)
    print(f"gamma={gamma:<5} train acc={model.score(X_train, y_train):.3f}"
          f"  test acc={model.score(X_test, y_test):.3f}  support vectors={len(model.support_)}")
```

OUTPUT

```
gamma=0.1   train acc=0.882  test acc=0.867  support vectors=115
gamma=1     train acc=0.932  test acc=0.925  support vectors=73
gamma=10    train acc=0.961  test acc=0.908  support vectors=102
gamma=100   train acc=0.993  test acc=0.892  support vectors=244
```

Linear SVM from scratch with NumPy

PYTHON

```
import numpy as np
from sklearn.datasets import make_blobs

class LinearSVM:
    """Soft-margin linear SVM trained with gradient descent on the hinge loss."""

    def __init__(self, C=1.0, lr=0.0001, epochs=2000):
        self.C, self.lr, self.epochs = C, lr, epochs

    def fit(self, X, y):
        y = np.where(y <= 0, -1, 1)          # SVM needs labels -1 and +1
        n, d = X.shape
        self.w, self.b = np.zeros(d), 0.0
        for epoch in range(self.epochs):
            margins = y * (X @ self.w + self.b)
            wrong = margins < 1               # points inside the margin or misclassified
            # gradient of 0.5*||w||^2 + C * sum(max(0, 1 - y(w.x + b)))
            grad_w = self.w - self.C * (y[wrong] @ X[wrong])
            grad_b = -self.C * y[wrong].sum()
            self.w -= self.lr * grad_w
            self.b -= self.lr * grad_b
            if epoch % 500 == 0:
                hinge = np.maximum(0, 1 - margins).sum()
                loss = 0.5 * self.w @ self.w + self.C * hinge
                print(f"epoch {epoch:<4} loss={loss:.2f}")
        return self

    def predict(self, X):
        return np.where(X @ self.w + self.b >= 0, 1, 0)

X, y = make_blobs(n_samples=200, centers=2, cluster_std=1.2, random_state=4)
svm = LinearSVM(C=1.0).fit(X, y)
print("w =", svm.w.round(3), " b =", round(svm.b, 3))
print("training accuracy =", (svm.predict(X) == y).mean())
print("margin width =", round(2 / np.linalg.norm(svm.w), 3))

# Compare with scikit-learn on the same data
from sklearn.svm import SVC
print("scikit-learn accuracy =", SVC(kernel="linear", C=1.0).fit(X, y).score(X, y))
```

OUTPUT

```
epoch 0    loss=200.00
epoch 500  loss=36.68
epoch 1000 loss=35.82
epoch 1500 loss=35.40
w = [-0.318  1.408]  b = -0.634
training accuracy = 0.94
margin width = 1.386
scikit-learn accuracy = 0.935
```

Breast cancer: scaling and GridSearchCV

PYTHON

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

X, y = load_breast_cancer(return_X_y=True)          # 569 tumours, 30 features each
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y)

# 1) Without scaling
raw = SVC(kernel="rbf").fit(X_train, y_train)
print("Without scaling:", round(raw.score(X_test, y_test), 3))

# 2) With scaling + a small grid search over C and gamma
pipe = make_pipeline(StandardScaler(), SVC(kernel="rbf"))
grid = GridSearchCV(pipe, {"svc__C": [0.1, 1, 10, 100],
                           "svc__gamma": ["scale", 0.01, 0.1, 1]}, cv=5)
grid.fit(X_train, y_train)
print("Best settings:", grid.best_params_)
print("With scaling + tuning:", round(grid.score(X_test, y_test), 3))
```

OUTPUT

```
Without scaling: 0.923
Best settings: {'svc__C': 1, 'svc__gamma': 0.01}
With scaling + tuning: 0.972
```

Multiclass SVM on iris

PYTHON

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

X, y = load_iris(return_X_y=True)                    # 3 flower species
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

model = SVC(kernel="rbf", C=1.0).fit(X_train, y_train)
print("Classes:", model.classes_)
print("Test accuracy:", round(model.score(X_test, y_test), 3))
print("Predicted species for first 5 test flowers:", model.predict(X_test[:5]))
print("Actual species:                            ", y_test[:5])
```

OUTPUT

```
Classes: [0 1 2]
Test accuracy: 0.978
Predicted species for first 5 test flowers: [2 1 0 2 0]
Actual species:                             [2 1 0 2 0]
```

9. SVM interview questions

1. What is a support vector machine?

A supervised learning algorithm, used mainly for classification, that finds the hyperplane separating two classes with the largest possible margin. It can also do regression (SVR) and handle curved boundaries using kernels.

2. What are support vectors?

The training points that lie on the margin, inside it, or on the wrong side of the boundary. They have non-zero Lagrange multipliers and are the only points that determine the decision boundary.

3. Why does SVM maximise the margin?

A wider margin leaves more room for new points that differ slightly from the training data, so the model tends to generalise better. In learning theory, a larger margin corresponds to lower model complexity.

4. What is the difference between a hard margin and a soft margin SVM?

A hard margin SVM allows no point inside the margin or on the wrong side, so it only works on perfectly separable data and is very sensitive to outliers. A soft margin SVM adds slack variables so some points can break the margin, at a cost controlled by C.

5. What does the C parameter do?

C is the penalty for points that violate the margin. A small C gives a wider, more forgiving margin and more support vectors. A large C gives a narrower margin that tries to classify every training point correctly, which can overfit noisy data.

6. What is the kernel trick?

A way to compute the dot product of two points as if they had been mapped to a higher-dimensional feature space, without computing that mapping. Since the SVM dual only uses dot products, this lets it learn non-linear boundaries cheaply.

7. Which kernels are commonly used?

Linear, polynomial, RBF (Gaussian) and sigmoid. RBF is the usual default when you do not know the shape of the data, and linear is preferred for high-dimensional data such as text.

8. What does gamma do in the RBF kernel?

It controls how far the influence of a single training point reaches. Low gamma gives smooth, wide-reaching influence and simple boundaries. High gamma gives very local influence and wiggly boundaries that can overfit.

9. Why is feature scaling important for SVM?

The RBF kernel uses distances and the linear kernel uses dot products. A feature with a large range, like salary in rupees, would dominate a feature with a small range, like age, unless both are scaled.

10. What is hinge loss?

The loss $\max(0, 1 - y \cdot f(x))$. It is zero when a point is on the correct side and outside the margin, and grows linearly as the point moves into the margin or across the boundary.

11. What is the difference between the primal and the dual form?

The primal form optimises the weights w and bias b directly. The dual form optimises one multiplier α per training point and only uses the data through dot products, which is what makes kernels possible. Kernel SVMs are trained in the dual.

12. What makes a function a valid kernel?

It must correspond to a dot product in some feature space. Mercer's condition says the kernel matrix it produces for any set of points must be symmetric and positive semi-definite.

13. How does SVM handle multiclass problems?

By combining binary SVMs. One-vs-one trains a classifier for every pair of classes and takes a vote; one-vs-rest trains one classifier per class against all others. scikit-learn's SVC uses one-vs-one.

14. Can SVM output probabilities?

Not directly. scikit-learn can estimate them with Platt scaling when you set probability=True, which fits a logistic model on the SVM scores using internal cross-validation. It makes training slower.

15. How do you handle imbalanced data with SVM?

Give the minority class a higher penalty using class_weight="balanced" (or custom weights), so mistakes on it cost more. Resampling the data and choosing a better metric than accuracy also help.

16. What is the time complexity of training an SVM?

Kernel SVM training is roughly between $O(n^2)$ and $O(n^3)$ in the number of training rows, and storing the kernel matrix takes $O(n^2)$ memory. Linear SVMs trained with LinearSVC or SGD scale roughly linearly.

17. When would you not use an SVM?

On very large datasets, when you need well-calibrated probabilities, when the model must be easy to explain, or on noisy data with heavily overlapping classes where ensembles usually do better.

18. What is the difference between SVM and logistic regression?

SVM maximises the margin using hinge loss and only depends on the support vectors. Logistic regression maximises likelihood using log loss, every point contributes, and it outputs probabilities directly.

Try the interactive SVM playground, the kernel trick demo and the quiz on the blog: codeanddebug.in/blog/support-vector-machine-in-machine-learning