

Support Vector Machines and Naive Bayes

Two classic classifiers with very different ideas. SVMs find the widest margin between classes, use soft margins (C) and kernels (RBF, polynomial) for non-linear boundaries, and need scaling; Naive Bayes applies Bayes' theorem with an independence assumption and is a fast, strong baseline for text. Includes SVR, probability calibration and when to choose each.

● ML track (added)

✦ scikit-learn 1.7+

Intermediate

⌚ 100 min

By the end of this lesson you can

- Explain maximum-margin classification, support vectors and the role of C
- Use linear and RBF kernels and tune C and gamma
- Apply SVR for regression and know SVM scaling and size limits
- Explain Naive Bayes with Bayes' theorem and its independence assumption
- Build Gaussian and multinomial Naive Bayes models, including a text classifier

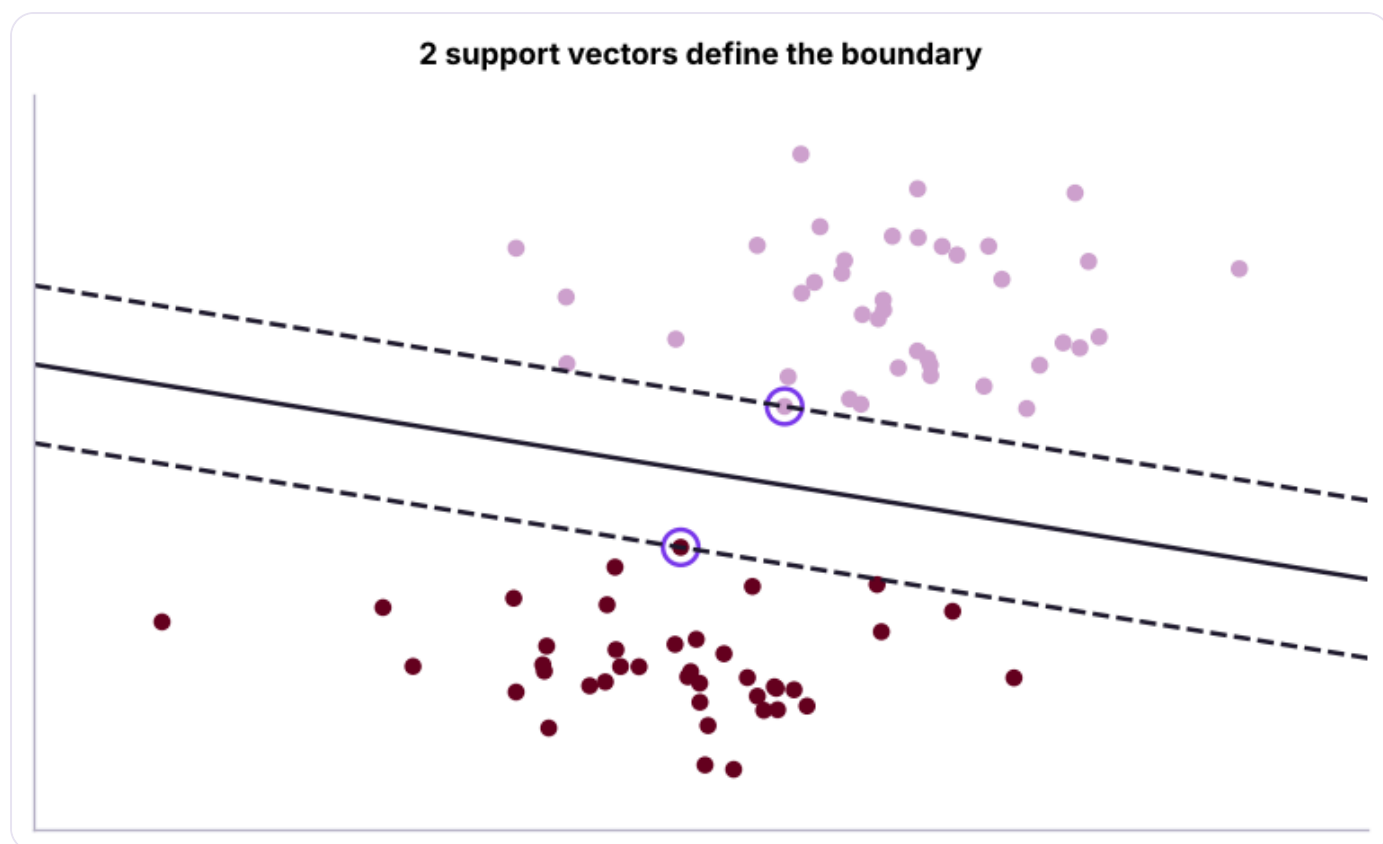
Datasets used

bank_loans.csv

customer_reviews.csv

Support vector machines: the widest street

Among all lines (hyperplanes) that separate two classes, an SVM picks the one with the **largest margin**: the widest "street" between the classes. Only the points on the edge of the street, the **support vectors**, determine it.



A linear SVM: the solid line is the decision boundary, dashed lines are the margins, circled points are the support vectors.

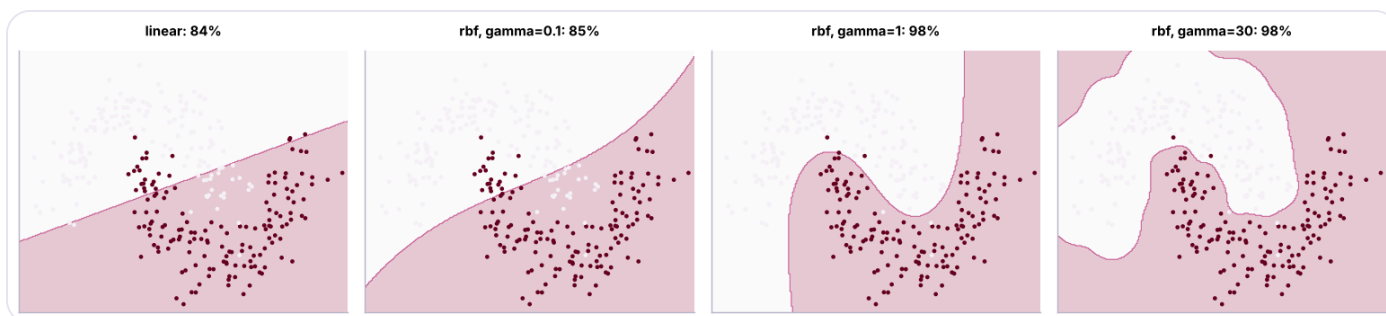
Soft margin: the C parameter

Real data overlap, so SVMs allow some points inside the margin or on the wrong side, with a penalty controlled by **C**:

C	Margin	Behaviour
Small (0.01)	Wide	Tolerates errors: simpler, more bias
Large (100)	Narrow	Tries to classify every point: complex, more variance

Kernels: non-linear boundaries

The **kernel trick** computes similarities as if the data were mapped into a higher-dimensional space where a straight separator exists, without computing that space explicitly.



Linear vs RBF kernels on curved data, and the effect of gamma (how far one point's influence reaches).

Kernel	Use	Key parameters
linear	Many features (text), linearly separable data	C
rbf (Gaussian)	Default for non-linear problems	C, gamma
poly	Polynomial interactions	degree, C

Large **gamma** makes each point's influence very local (wiggly boundary, overfitting); small gamma gives smooth boundaries.

SVM on the Bank Loans data

Python

```

from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold,
cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
b = pd.read_csv("bank_loans.csv").dropna(subset=["default"])
feats = ["age", "ed", "employ", "address", "income", "debtinc", "creddebt", "othdebt"]
Xb, yb = b[feats], b.default.astype(int)
Xtr, Xte, ytr, yte = train_test_split(Xb, yb, test_size=0.3, stratify=yb, random_state=42)
cv = StratifiedKFold(5, shuffle=True, random_state=0)
gs = GridSearchCV(make_pipeline(StandardScaler(), SVC(kernel="rbf")), {"svc__C": [0.1, 0.3,
1, 3, 10], "svc__gamma": [0.003, 0.01, 0.03, 0.1, 0.3]}, cv=cv, scoring="roc_auc",
n_jobs=-1).fit(Xtr, ytr)
print("best:", gs.best_params_, f"| CV AUC {gs.best_score_:.3f} | test AUC
{roc_auc_score(yte, gs.decision_function(Xte)):.3f}")
lin = make_pipeline(StandardScaler(), SVC(kernel="linear", C=0.1)).fit(Xtr, ytr)
print(f"linear SVM test AUC {roc_auc_score(yte, lin.decision_function(Xte)):.3f} | logistic
regression {roc_auc_score(yte, make_pipeline(StandardScaler(),
LogisticRegression()).fit(Xtr, ytr).predict_proba(Xte)[: , 1]):.3f}")

```

OUTPUT

```
best: {'svc__C': 3, 'svc__gamma': 0.003} | CV AUC 0.788 | test AUC 0.809
linear SVM test AUC 0.813 | logistic regression 0.822
```

SVMs output **decision scores**, not probabilities. `SVC(probability=True)` adds Platt scaling (slower); `CalibratedClassifierCV` is the general tool.

Support vector regression

Python

```
h = pd.read_csv("house_prices.csv").dropna(subset=["area_sqft"]); h =
h[h.price_lakh.between(5, 300)]
hf = ["area_sqft", "bedrooms", "bathrooms", "age_years", "distance_to_center_km",
"parking"]
from sklearn.metrics import mean_absolute_error
a, bq, c, d = train_test_split(h[hf], h.price_lakh, test_size=0.2, random_state=0)
svr = make_pipeline(StandardScaler(), SVR(kernel="rbf", C=100, epsilon=2,
gamma=0.05)).fit(a, c)
print(f"SVR test MAE {mean_absolute_error(d, svr.predict(bq)):.2f} lakh")
```

OUTPUT

SVR test MAE 10.41 lakh

SVR fits a tube of width ϵ around the data and ignores errors inside it.

SVM practicalities

- Always **scale** features.
- Kernel SVMs scale roughly $O(n^2)$ – $O(n^3)$: fine for thousands of rows, slow beyond ~100,000. Use `LinearSVC`, `SGDClassifier(loss="hinge")` or kernel approximations (`Nystroem`) for big data.
- Strong for high-dimensional, small-to-medium data (text, genomics, image features).

Naive Bayes

Naive Bayes applies **Bayes' theorem** (ML lesson 2) and assumes features are **conditionally independent** given the class:

$$P(c \mid x_1, \dots, x_n) \propto P(c) \prod_{j=1}^n P(x_j \mid c)$$

The assumption is rarely true ("naive"), yet the classifier often works well because it only needs the **ranking** of classes to be right.

Variant	Features	Typical use
GaussianNB	Continuous, roughly normal	Quick baseline on numeric data
MultinomialNB	Counts (word counts)	Text classification, spam
BernoulliNB	Binary (word present/absent)	Short texts
ComplementNB	Counts, imbalanced classes	Text with skewed classes

Python

```
from sklearn.naive_bayes import GaussianNB, MultinomialNB, ComplementNB
gnb = GaussianNB().fit(Xtr, ytr)
print(f"Gaussian NB on bank loans: test AUC {roc_auc_score(yte, gnb.predict_proba(Xte)[:,\n1]):.3f} | class priors {gnb.class_prior_.round(3)}")
```

OUTPUT

```
Gaussian NB on bank loans: test AUC 0.797 | class priors [0.678 0.322]
```

Text classification with multinomial Naive Bayes

Python

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import classification_report, accuracy_score
rv = pd.read_csv("customer_reviews.csv")
print(rv.shape, rv.sentiment.value_counts().to_dict()); print(rv.review_text.sample(3,\nrandom_state=1).tolist())
Rtr, Rte, rtr, rte = train_test_split(rv.review_text, rv.sentiment, test_size=0.25,\nstratify=rv.sentiment, random_state=0)
nb = make_pipeline(CountVectorizer(ngram_range=(1, 2), min_df=2),\nMultinomialNB(alpha=0.5)).fit(Rtr, rtr)
print(f"multinomial NB accuracy {accuracy_score(rte, nb.predict(Rte)):.3f}")
print(classification_report(rte, nb.predict(Rte), digits=3))
vec, clf = nb[0], nb[-1]; vocab = np.array(vec.get_feature_names_out())
for i, cls in enumerate(clf.classes_):\n    lift = clf.feature_log_prob_[i] - clf.feature_log_prob_.mean(axis=0)\n    print(f"most indicative of {cls:8s}:", vocab[np.argsort(-lift)[:6]].tolist())
print(nb.predict(["battery died after two days, very disappointed", "excellent sound and\ngreat value", "it is okay for the price"]))
```

OUTPUT

```
(3000, 8) {'positive': 1589, 'negative': 964, 'neutral': 447}
['Delivered on the expected date', 'Nothing special, does the job.', 'Very
disappointed. The hinge broke within weeks, too expensive compared to rivals, the box
arrived super damaged though the bass is rich']
multinomial NB accuracy 0.940
```

	precision	recall	f1-score	support
negative	0.978	0.938	0.958	241
neutral	0.845	0.830	0.838	112
positive	0.944	0.972	0.958	397
accuracy			0.940	750
macro avg	0.923	0.913	0.918	750
weighted avg	0.940	0.940	0.940	750

```

most indicative of negative: ['worst purchase', 'worst', 'ever', 'purchase ever',
'regret', 'regret buying']
most indicative of neutral : ['is fine', 'it is', 'fine', 'mixed', 'feelings so',
'feelings']
most indicative of positive: ['superb product', 'not bad', 'bad', 'at all', 'bad at',
'loved it']
['negative' 'positive' 'neutral']

```

Notice that "bad" is one of the strongest **positive** indicators: in these reviews it mostly appears in "not bad at all". Word pairs (`ngram_range=(1, 2)`) let the model learn such negations; single words alone would get them wrong. The neutral class is hardest, because neutral reviews mix praise and complaints.

Python

```
svm_text = make_pipeline(TfidfVectorizer(ngram_range=(1, 2), min_df=2, sublinear_tf=True),
LinearSVC(C=0.5)).fit(Rtr, rtr)
print(f"TF-IDF + linear SVM accuracy {accuracy_score(rte, svm_text.predict(Rte)):.3f}")
```

OUTPUT

```
TF-IDF + linear SVM accuracy 0.947
```

Naive Bayes and linear SVMs on TF-IDF were the standard text classifiers before deep learning, and they are still excellent fast baselines (lesson 25 and the GenAI track go further).

When to choose which

Situation	Good choice
Small/medium data, complex boundary, scaled numeric features	RBF SVM
Very many sparse features (text)	Linear SVM, logistic regression, multinomial NB
Need a fast baseline, tiny training time, streaming updates	Naive Bayes (<code>partial_fit</code>)
Need calibrated probabilities	Logistic regression (or calibrate SVM/NB)
Large tabular data	Gradient boosting

Summary

KEY TAKEAWAYS

- SVMs maximise the margin; support vectors define the boundary; C trades margin width for training errors.
- Kernels (RBF, polynomial) give non-linear boundaries; gamma controls how local each point's influence is; always scale.
- SVMs give scores, not probabilities (calibrate); kernel SVMs are slow on very large data.
- Naive Bayes multiplies class-conditional probabilities under an independence assumption; it is fast and strong for text.
- TF-IDF + linear SVM or multinomial NB are classic, competitive text classifiers.

Quick quiz

6 questions · tap an answer to check it

1. What are support vectors?

- A** The training points on or inside the margin that define the boundary
- B All training points
- C The features
- D The predicted probabilities

Answer: A. Other points could move without changing the boundary.

2. A larger C in an SVM means...

- A** A narrower margin and fewer tolerated training errors (more complex)
- B A wider margin
- C More regularisation
- D A different kernel

Answer: A. Small C = more regularisation.

3. What does a very large gamma in an RBF SVM tend to cause?

- A** Overfitting with a very wiggly boundary
- B Underfitting
- C A linear boundary
- D Faster training

Answer: A. Each point influences only its close neighbourhood.

4. Why is Naive Bayes called "naive"?

- A** It assumes features are independent given the class
- B It is simple to code
- C It ignores priors
- D It needs no data

Answer: A. The assumption is rarely true, but it works well in practice.

5. Which Naive Bayes variant suits word counts?

- A GaussianNB
- B** MultinomialNB
- C BernoulliNB only for numbers
- D None

Answer: B. Counts per word per document.

6. Which step is essential before training an RBF SVM?

- A** Feature scaling
- B One-hot encoding the target
- C Removing all outliers
- D PCA always

Answer: A. Distances drive the kernel.

EXERCISE Practice

(1) Tune C and gamma for an RBF SVM on the telecom churn data (one-hot + scale) and compare with logistic regression. (2) Train BernoulliNB on the reviews with binary word features. (3) Use `CalibratedClassifierCV` on the linear SVM text model to get probabilities.

Show solution (worked answer)

Python

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.naive_bayes import BernoulliNB
from sklearn.calibration import CalibratedClassifierCV
tc = pd.read_csv("telecom_churn.csv"); Xc = tc.drop(columns=["customer_id", "churn",
"total_charges"]); yc = (tc.churn == "Yes").astype(int)
pre = ColumnTransformer([("n", StandardScaler(), ["senior_citizen", "tenure_months",
"monthly_charges", "support_calls"]), ("c", OneHotEncoder(), ["contract",
"internet_service", "payment_method"])]])
g = GridSearchCV(make_pipeline(pre, SVC()), {"svc__C": [0.1, 1, 10], "svc__gamma":
[0.01, 0.05, 0.2]}, cv=5, scoring="roc_auc").fit(Xc, yc); print(g.best_params_,
g.best_score_)
print(accuracy_score(rte, make_pipeline(CountVectorizer(binary=True),
BernoulliNB()).fit(Rtr, rtr).predict(Rte)))
cal = make_pipeline(TfidfVectorizer(ngram_range=(1, 2)),
CalibratedClassifierCV(LinearSVC(), cv=5)).fit(Rtr, rtr);
print(cal.predict_proba(Rte[:3]).round(2))
```

INTERVIEW CORNER Questions you may be asked

Q: Explain the kernel trick. It lets an algorithm that depends only on dot products work in a high-dimensional feature space by computing a similarity function (kernel) directly in the original space.

Q: SVM vs logistic regression? Both can learn linear boundaries; logistic regression gives probabilities and optimises log loss; SVMs maximise the margin with hinge loss and extend to non-linear boundaries with kernels, but scale worse to large datasets.

Q: Why does Naive Bayes work well for spam detection despite its assumption?

Classification only needs the correct class to have the highest score; with many weakly informative words, the independence errors largely cancel out, and the model is fast and robust with little data.

Practise more online

Free, trusted places to test yourself on this topic. Links open the original site in a new tab.