L09Hyperparameter_Tuning

October 19, 2025

1 Hyperparameter Tuning — Student Template (INTERMEDIATE+)

This version contains **real coding tasks**. Many cells include `TODO` markers and `raise NotImplementedError()` until you implement the logic.

Do this: - Randomized + Grid Search - Custom ROC/CM plots - Learning & Validation curves - Save artifacts

Rules - matplotlib only; one chart per figure. - Keep RANDOM_STATE = 42.

[1]:
```python
# ================================
# Imports & Configuration  (TODO)
# ================================

import os, warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV, learning_curve, validation_curve, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score, roc_curve, confusion_matrix, ConfusionMatrixDisplay
from sklearn.utils import check_random_state

import joblib

RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)
rng = check_random_state(RANDOM_STATE)
```

```
plt.rcParams["figure.figsize"] = (7, 5)
```

[2]:
```
# ========================
# 1) Load & Inspect Data
# ========================

data = load_breast_cancer(as_frame=True)
X = data.data
y = data.target

print("X shape:", X.shape)
print("Class distribution:", dict(zip(*np.unique(y, return_counts=True))))
display(X.head())

feature_names = X.columns.tolist()
target_names = data.target_names.tolist()
```

```
X shape: (569, 30)
Class distribution: {0: 212, 1: 357}
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	\
0	17.99	10.38	122.80	1001.0	0.11840	
1	20.57	17.77	132.90	1326.0	0.08474	
2	19.69	21.25	130.00	1203.0	0.10960	
3	11.42	20.38	77.58	386.1	0.14250	
4	20.29	14.34	135.10	1297.0	0.10030	

	mean compactness	mean concavity	mean concave points	mean symmetry	\
0	0.27760	0.3001	0.14710	0.2419	
1	0.07864	0.0869	0.07017	0.1812	
2	0.15990	0.1974	0.12790	0.2069	
3	0.28390	0.2414	0.10520	0.2597	
4	0.13280	0.1980	0.10430	0.1809	

	mean fractal dimension	…	worst radius	worst texture	worst perimeter	\
0	0.07871	…	25.38	17.33	184.60	
1	0.05667	…	24.99	23.41	158.80	
2	0.05999	…	23.57	25.53	152.50	
3	0.09744	…	14.91	26.50	98.87	
4	0.05883	…	22.54	16.67	152.20	

	worst area	worst smoothness	worst compactness	worst concavity	\
0	2019.0	0.1622	0.6656	0.7119	
1	1956.0	0.1238	0.1866	0.2416	
2	1709.0	0.1444	0.4245	0.4504	
3	567.7	0.2098	0.8663	0.6869	
4	1575.0	0.1374	0.2050	0.4000	

```
   worst concave points  worst symmetry  worst fractal dimension
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300
4                0.1625          0.2364                  0.07678

[5 rows x 30 columns]
```

[3]:

```
# ============================================
# 2) Train/Test Split + Sanity Checks (TODO)
# ============================================

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=RANDOM_STATE
)

print("Train:", X_train.shape, "| Test:", X_test.shape)

assert set(X_train.columns) == set(X_test.columns), "Feature mismatch"
assert len(np.unique(y_train)) == 2 and len(np.unique(y_test)) == 2, "Expect␣
↪binary labels"
print("Basic checks passed.")
```

```
Train: (455, 30) | Test: (114, 30)
Basic checks passed.
```

[4]:

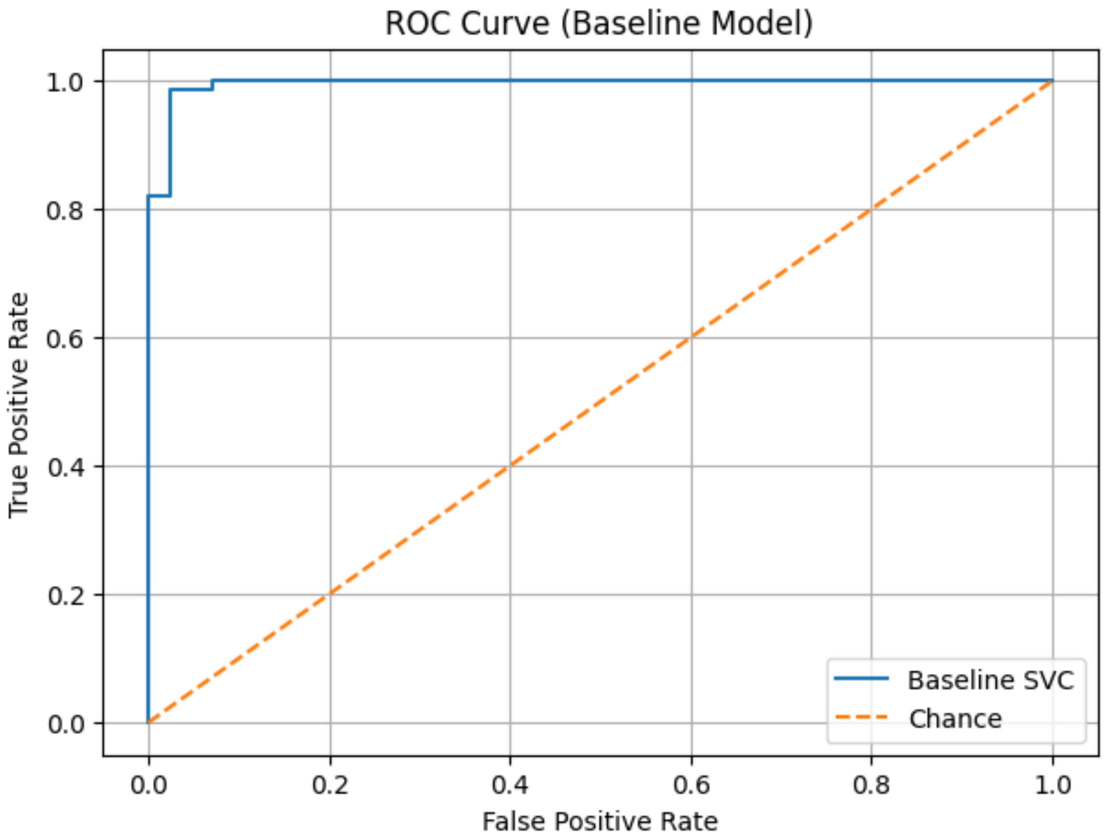
```
# ==========================================
# 3) Baseline Pipeline & CV Accuracy (TODO)
# ==========================================

baseline_pipe = Pipeline([
    ("scaler", StandardScaler(with_mean=True, with_std=True)),
    ("svc", SVC(probability=True, random_state=RANDOM_STATE))
])

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=RANDOM_STATE)

baseline_cv_scores = cross_val_score(
    estimator=baseline_pipe,
    X=X_train, y=y_train,
    cv=cv, scoring="accuracy"
)
print(f"Baseline CV Accuracy: {baseline_cv_scores.mean():.4f} ±␣
↪{baseline_cv_scores.std():.4f}")

baseline_pipe.fit(X_train, y_train)
```

```
y_proba_base = baseline_pipe.predict_proba(X_test)[:, 1]
y_pred_base = baseline_pipe.predict(X_test)

print("\n[Baseline] Test Accuracy:", accuracy_score(y_test, y_pred_base))
print("[Baseline] ROC AUC:", roc_auc_score(y_test, y_proba_base))
print("\n[Baseline] Classification Report:\n", classification_report(y_test,␣
↪y_pred_base))
```

```
Baseline CV Accuracy: 0.9692 ± 0.0146

[Baseline] Test Accuracy: 0.9824561403508771
[Baseline] ROC AUC: 0.9950396825396826

[Baseline] Classification Report:
               precision    recall  f1-score   support

           0       0.98      0.98      0.98        42
           1       0.99      0.99      0.99        72

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

[5]:

```
# ================================
# 3b) Baseline ROC Curve (TODO)
# ================================

# Implement the ROC plot for baseline
fpr, tpr, _ = roc_curve(y_test, y_proba_base)
plt.figure()
plt.plot(fpr, tpr, label="Baseline SVC")
plt.plot([0, 1], [0, 1], linestyle="--", label="Chance")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve (Baseline Model)")
plt.legend()
plt.grid(True)
plt.show()
```



[6]:
```
# ======================================================
# 4) Broad Search: RandomizedSearchCV (TODO)
# ======================================================

def log_space(rng, low_exp, high_exp, size):
    return np.power(10, rng.uniform(low=low_exp, high=high_exp, size=size))

# TODO: Build param_distributions
param_distributions = {
    "svc__C": log_space(rng, -2, 3, 50),
    "svc__gamma": log_space(rng, -4, 1, 50),
    "svc__kernel": ["rbf"]
}

# TODO: Configure & fit RandomizedSearchCV
rand_search = RandomizedSearchCV(
    estimator=baseline_pipe,
    param_distributions=param_distributions,
    n_iter=30,
    cv=cv,
```

```
    scoring="accuracy",
    n_jobs=-1,
    random_state=RANDOM_STATE
)
rand_search.fit(X_train, y_train)

print("Best params (Randomized):", rand_search.best_params_)
print("Best CV score (Randomized):", f"{rand_search.best_score_:.4f}")
```

```
Best params (Randomized): {'svc__kernel': 'rbf', 'svc__gamma':
0.013731092468240284, 'svc__C': 9.16374180877878}
Best CV score (Randomized): 0.9758
```

[7]:

```
# ============================================================
# 5) Fine Search: GridSearchCV around random best (TODO)
# ============================================================

best_C = rand_search.best_params_["svc__C"]
best_gamma = rand_search.best_params_["svc__gamma"]

C_grid = [best_C / 3, best_C, best_C * 3]
gamma_grid = [best_gamma / 3, best_gamma, best_gamma * 3]

grid_params = {"svc__kernel": ["rbf"], "svc__C": C_grid, "svc__gamma":
 ↪gamma_grid}

grid_search = GridSearchCV(
    estimator=baseline_pipe,
    param_grid=grid_params,
    cv=cv,
    scoring="accuracy",
    n_jobs=-1
)
grid_search.fit(X_train, y_train)

print("Best params (Grid):", grid_search.best_params_)
print("Best CV score (Grid):", f"{grid_search.best_score_:.4f}")
```

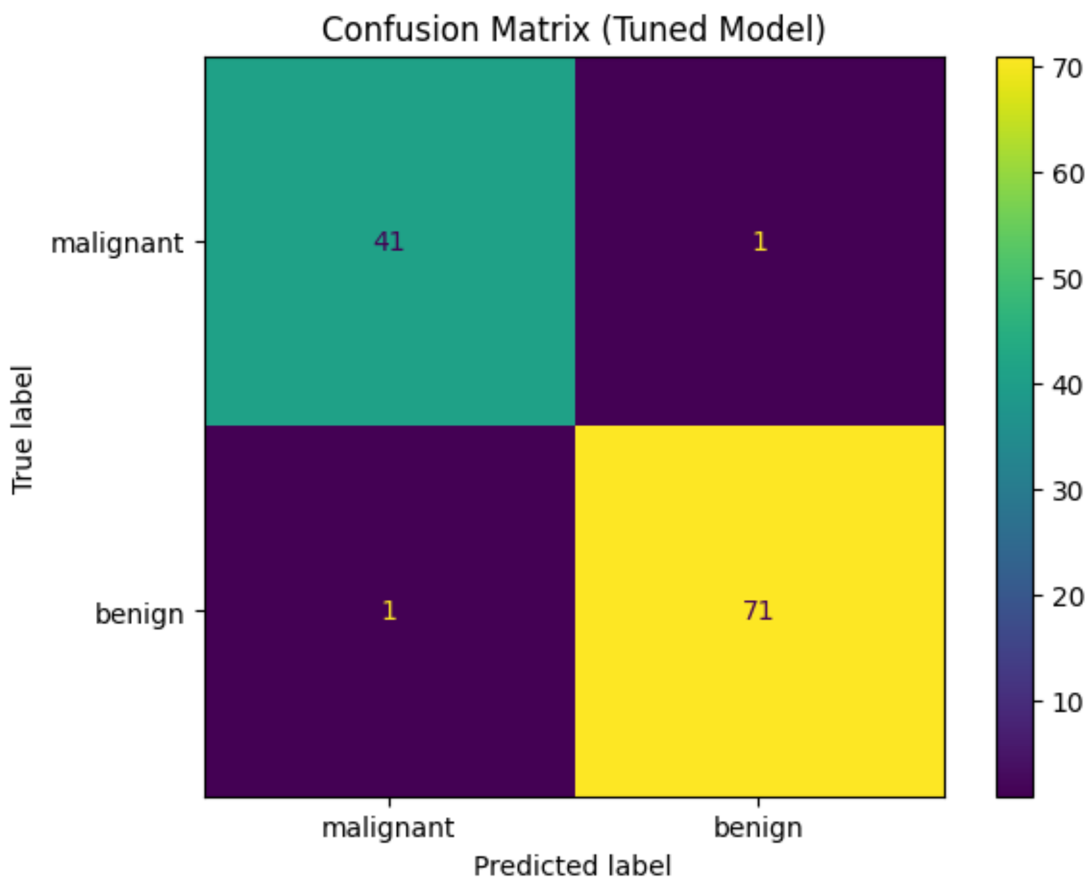
```
Best params (Grid): {'svc__C': 9.16374180877878, 'svc__gamma':
0.0045770308227467615, 'svc__kernel': 'rbf'}
Best CV score (Grid): 0.9758
```

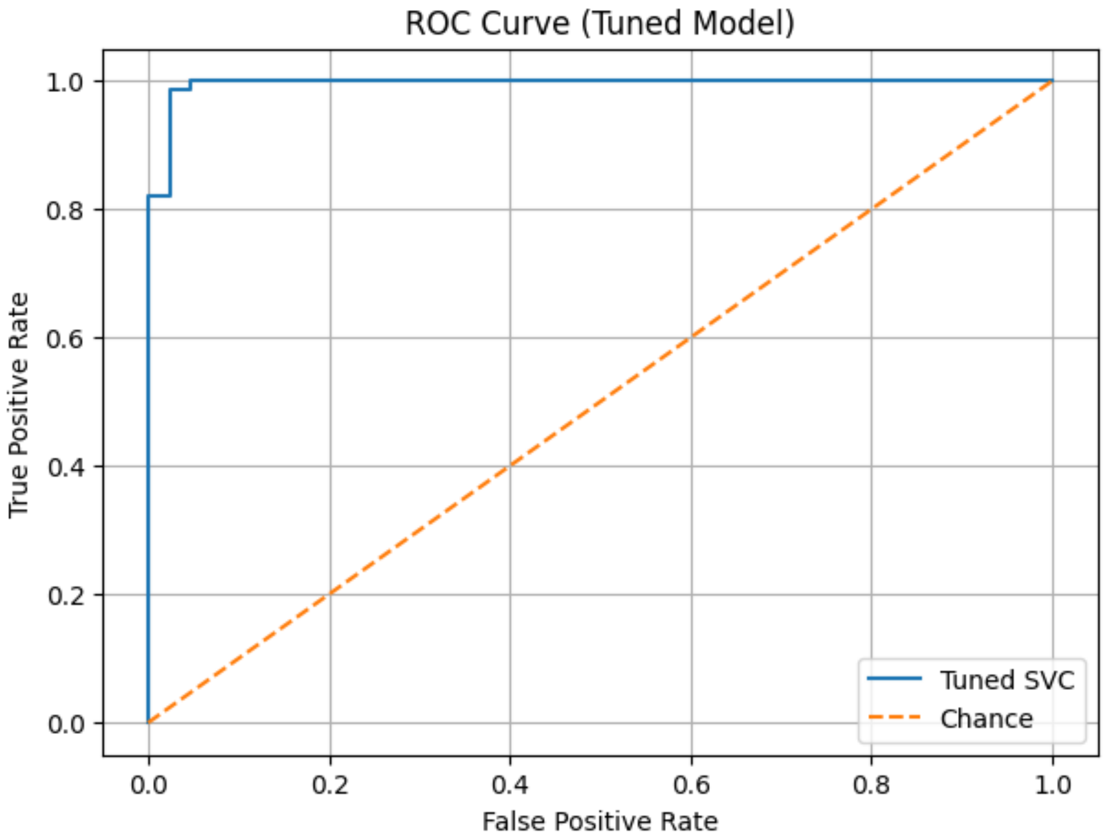
[8]:

```
# ======================================
# 6) Evaluate Tuned Best Model (TODO)
# ======================================

best_model = grid_search.best_estimator_
```

```python
y_pred = best_model.predict(X_test)
y_proba = best_model.predict_proba(X_test)[:, 1]

print("Test Accuracy (tuned):", accuracy_score(y_test, y_pred))
print("ROC AUC (tuned):", roc_auc_score(y_test, y_proba))
print("\nClassification Report (tuned):\n", classification_report(y_test,␣
↪y_pred))

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=target_names)
fig, ax = plt.subplots()
disp.plot(ax=ax, values_format="d")
ax.set_title("Confusion Matrix (Tuned Model)")
plt.show()

# ROC curve for tuned model
fpr, tpr, _ = roc_curve(y_test, y_proba)
plt.figure()
plt.plot(fpr, tpr, label="Tuned SVC")
plt.plot([0, 1], [0, 1], linestyle="--", label="Chance")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve (Tuned Model)")
plt.legend()
plt.grid(True)
plt.show()
```

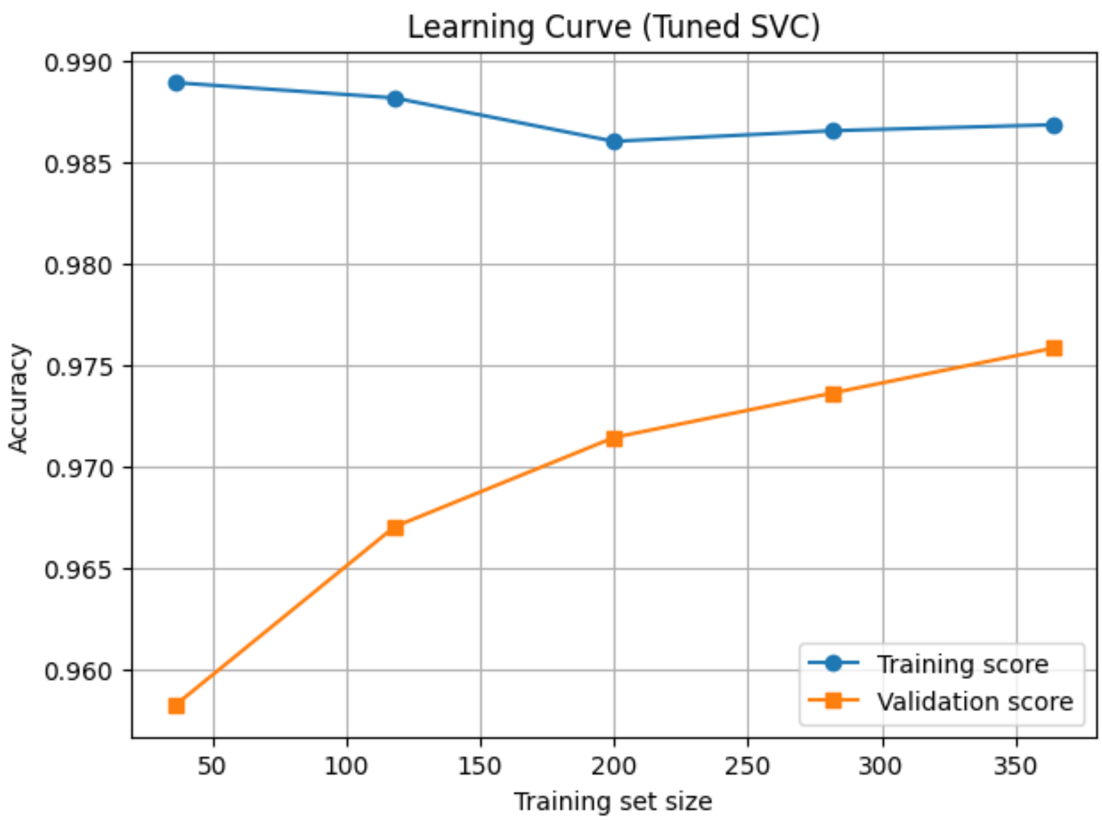
```
Test Accuracy (tuned): 0.9824561403508771
ROC AUC (tuned): 0.9953703703703703

Classification Report (tuned):
               precision    recall  f1-score   support

           0       0.98      0.98      0.98        42
           1       0.99      0.99      0.99        72

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```


Confusion Matrix (Tuned Model)
malignant
benign
True label
41
1
1
71
malignant
benign
Predicted label
70
60
50
40
30
20
10



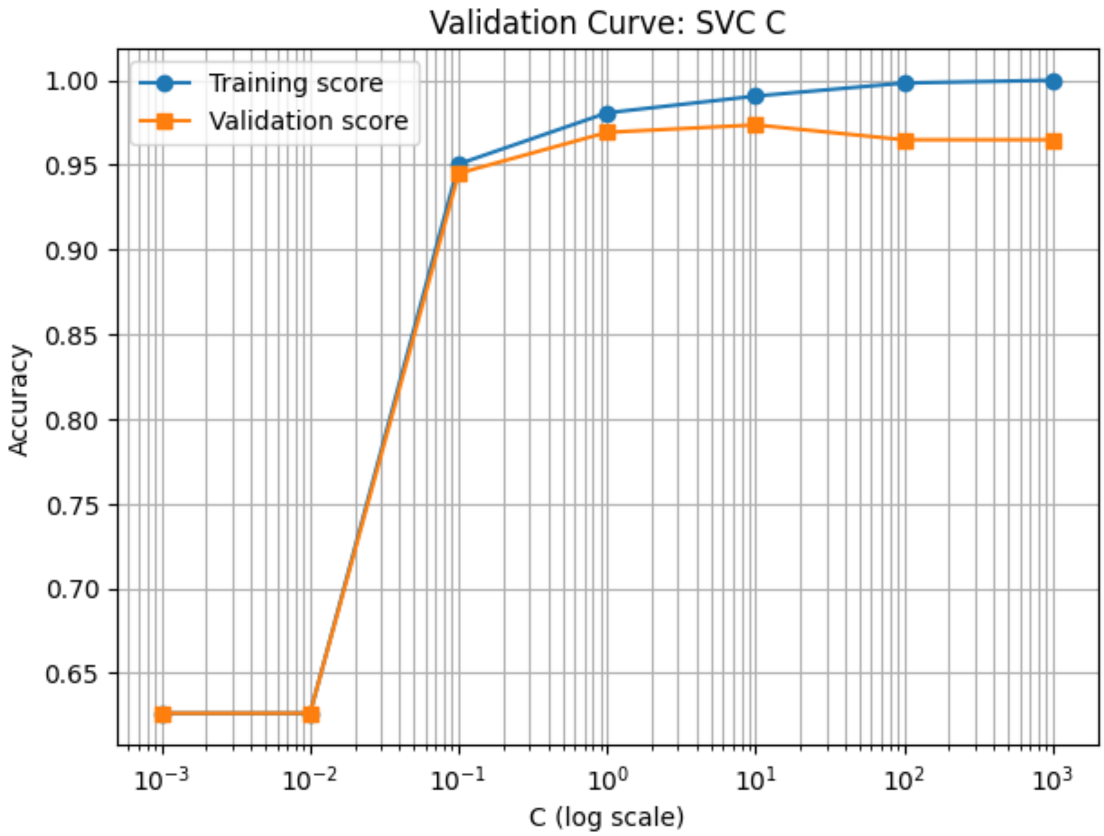
[9]:
```
# ==================================
# 7) Learning Curve Plot (TODO)
# ==================================

train_sizes, train_scores, valid_scores = learning_curve(
    estimator=best_model,
    X=X_train, y=y_train,
    cv=cv,
    scoring="accuracy",
    n_jobs=-1,
    train_sizes=np.linspace(0.1, 1.0, 5),
    shuffle=True,
    random_state=RANDOM_STATE
)

train_mean = train_scores.mean(axis=1)
valid_mean = valid_scores.mean(axis=1)

plt.figure()
plt.plot(train_sizes, train_mean, marker="o", label="Training score")
```

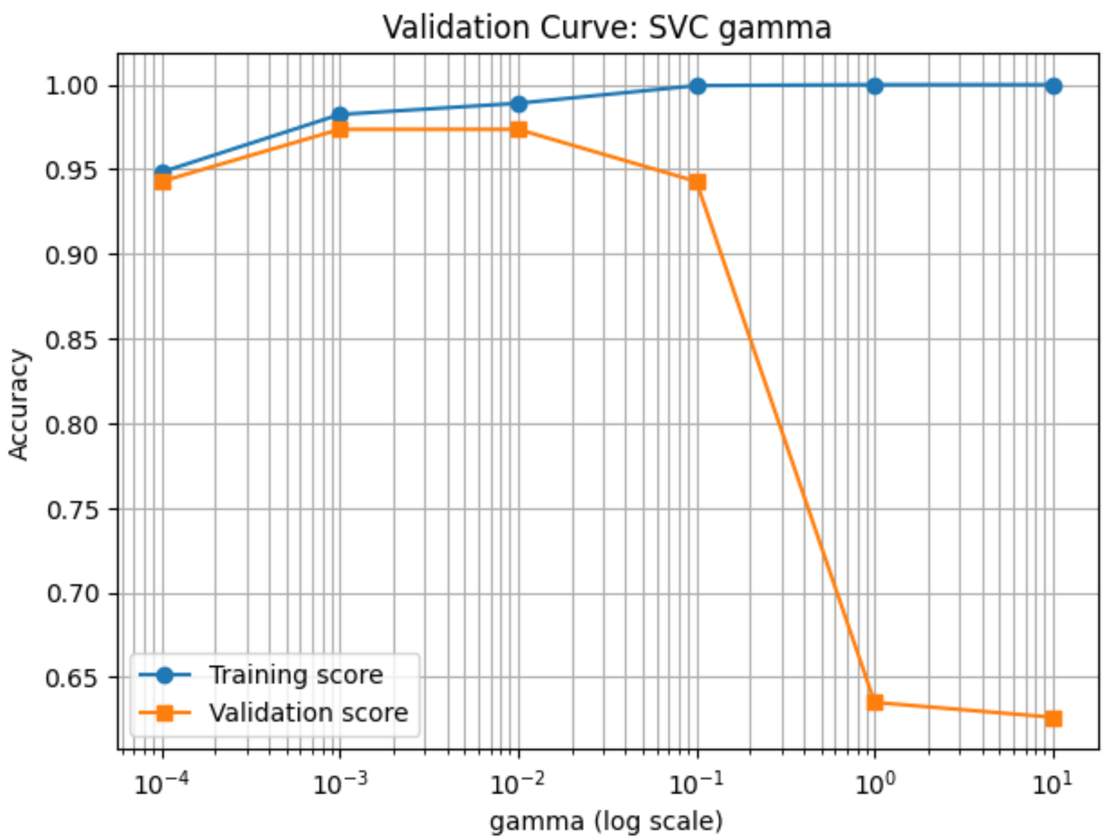
```
plt.plot(train_sizes, valid_mean, marker="s", label="Validation score")
plt.xlabel("Training set size")
plt.ylabel("Accuracy")
plt.title("Learning Curve (Tuned SVC)")
plt.legend()
plt.grid(True)
plt.show()
```



[10]:
```
# ===================================
# 8) Validation Curves (TODO)
# ===================================

# -- Vary C --
C_range = np.logspace(-3, 3, 7)
train_scores_C, valid_scores_C = validation_curve(
    estimator=Pipeline([("scaler", StandardScaler()),
                        ("svc", SVC(kernel="rbf", gamma=best_gamma,
 ↪probability=True, random_state=RANDOM_STATE))]),
    X=X_train, y=y_train,
    param_name="svc__C",
```

```
    param_range=C_range,
    cv=cv,
    scoring="accuracy",
    n_jobs=-1
)

plt.figure()
plt.semilogx(C_range, train_scores_C.mean(axis=1), marker="o", label="Training␣
 ↪score")
plt.semilogx(C_range, valid_scores_C.mean(axis=1), marker="s",␣
 ↪label="Validation score")
plt.xlabel("C (log scale)")
plt.ylabel("Accuracy")
plt.title("Validation Curve: SVC C")
plt.legend()
plt.grid(True, which="both")
plt.show()

# -- Vary gamma --
gamma_range = np.logspace(-4, 1, 6)
train_scores_g, valid_scores_g = validation_curve(
    estimator=Pipeline([("scaler", StandardScaler()),
                        ("svc", SVC(kernel="rbf", C=best_C, probability=True,␣
 ↪random_state=RANDOM_STATE))]),
    X=X_train, y=y_train,
    param_name="svc__gamma",
    param_range=gamma_range,
    cv=cv,
    scoring="accuracy",
    n_jobs=-1
)

plt.figure()
plt.semilogx(gamma_range, train_scores_g.mean(axis=1), marker="o",␣
 ↪label="Training score")
plt.semilogx(gamma_range, valid_scores_g.mean(axis=1), marker="s",␣
 ↪label="Validation score")
plt.xlabel("gamma (log scale)")
plt.ylabel("Accuracy")
plt.title("Validation Curve: SVC gamma")
plt.legend()
plt.grid(True, which="both")
plt.show()
```


Validation Curve: SVC C
Training score
Validation score
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
C (log scale)



[11]:
```
# ============================================
# 9) Save Artifacts: model + CSVs (TODO)
# ============================================

# Create directory
os.makedirs("artifacts", exist_ok=True)

# Save model
model_path = os.path.join("artifacts", "best_svc_pipeline.joblib")
joblib.dump(best_model, model_path)
print("Saved model to:", model_path)

# Save CV results
rand_results_path = os.path.join("artifacts", "randomized_search_results.csv")
grid_results_path = os.path.join("artifacts", "grid_search_results.csv")
pd.DataFrame(rand_search.cv_results_).to_csv(rand_results_path, index=False)
pd.DataFrame(grid_search.cv_results_).to_csv(grid_results_path, index=False)
print("Saved CV results to:", rand_results_path, "and", grid_results_path)
```

```
Saved model to: artifacts/best_svc_pipeline.joblib
Saved CV results to: artifacts/randomized_search_results.csv and
```

artifacts/grid_search_results.csv

1.1 Short Reflection

1. Is your tuned model **high-bias**, **high-variance**, or **balanced**? Explain using the learning curve.

The tuned model is well-balanced. The learning curve indicates that both training and validation scores reach high accuracy levels and converge with a small gap between them, demonstrating that the model generalizes effectively without significant overfitting or underfitting.

2. Which improved more from baseline to tuned — **accuracy** or **ROC AUC**? Why might that be?

ROC AUC slightly increased from 0.9950 to 0.9954, while accuracy remained at 98.25% because AUC measures threshold-free ranking. Tuning improved score ranking even though the predicted labels at the model's default decision boundary did not change. In other words, AUC assesses discrimination across all thresholds, so even small improvements in how well positives are ranked above negatives can raise AUC without affecting the final class predictions.

3. Where do you observe **over-regularization** in the validation curves?

Over-regularization occurs at low C values, where both training and validation accuracy decrease to about 0.63 because of excessive regularization. For gamma, low values sustain reasonable performance around 0.95 with smooth boundaries, while very high gamma leads to significant overfitting with training accuracy reaching 1.0 but validation dropping to 0.64.

4. If you collected **10× more data**, how would you adjust **C** or **gamma**, and why?

With 10 times more data, we would expand our search space to include higher values of C, as additional data reduces the risk of overfitting and permits less regularization. For gamma, the relationship is more complex. While higher gamma values can be explored safely without overfitting, more data might actually show that moderate gamma values generalize better, since we have enough samples to accurately estimate smoother decision boundaries. Instead of assuming more data means higher hyperparameters, we would systematically re-explore the entire hyperparameter space using cross-validation to empirically find the best settings for the larger dataset.