```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split,
GridSearchCV, learning_curve, validation_curve,
cross_val_score
from sklearn.preprocessing import StandardScaler,
LabelEncoder
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import Perceptron,
LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import StratifiedKFold

# 載入資料
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-
learning-databases/breast-cancer-wisconsin/wdbc.data',
header=None)

# 特徵與標籤前處理
X = df.loc[:, 2:].values
y = df.loc[:, 1].values
le = LabelEncoder()
y = le.fit_transform(y)

# 資料分割
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, stratify=y, random_state=1
)

# 創建一個用於繪製學習曲線和驗證曲線的函數
def plot_learning_curve(estimator, X, y, title):
    train_sizes, train_scores, test_scores =
learning_curve(
        estimator, X, y,
        train_sizes=np.linspace(0.1, 1.0, 10),
        cv=StratifiedKFold(n_splits=10),
        n_jobs=-1
    )
```

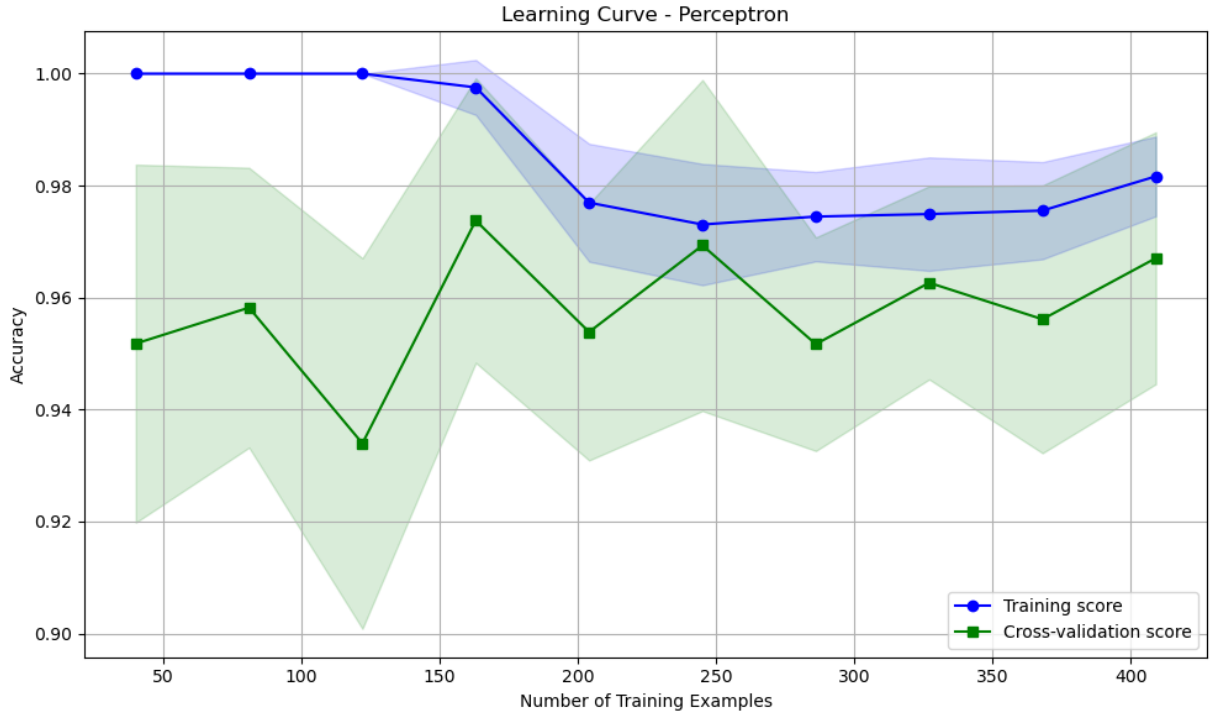
```
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)

    plt.figure(figsize=(10, 6))
    plt.title(f'Learning Curve - {title}')
    plt.xlabel('Number of Training Examples')
    plt.ylabel('Accuracy')

    plt.plot(train_sizes, train_mean, label='Training
score', color='blue', marker='o')
    plt.fill_between(train_sizes, train_mean - train_std,
train_mean + train_std, alpha=0.15, color='blue')

    plt.plot(train_sizes, test_mean, label='Cross-
validation score', color='green', marker='s')
    plt.fill_between(train_sizes, test_mean - test_std,
test_mean + test_std, alpha=0.15, color='green')

    plt.legend(loc='lower right')
    plt.grid()
    plt.tight_layout()
    plt.show()

def custom_depth_validation_curve(pipe_dt, X, y, depths):
    """
    Custom validation curve for decision tree max_depth
that handles None

    Args:
        pipe_dt: Pipeline with DecisionTreeClassifier
        X: Feature matrix
        y: Target vector
        depths: List of depths to evaluate (can include
None)
    """
    # Prepare to store scores
    train_scores = []
    test_scores = []

    # Iterate through depths
```

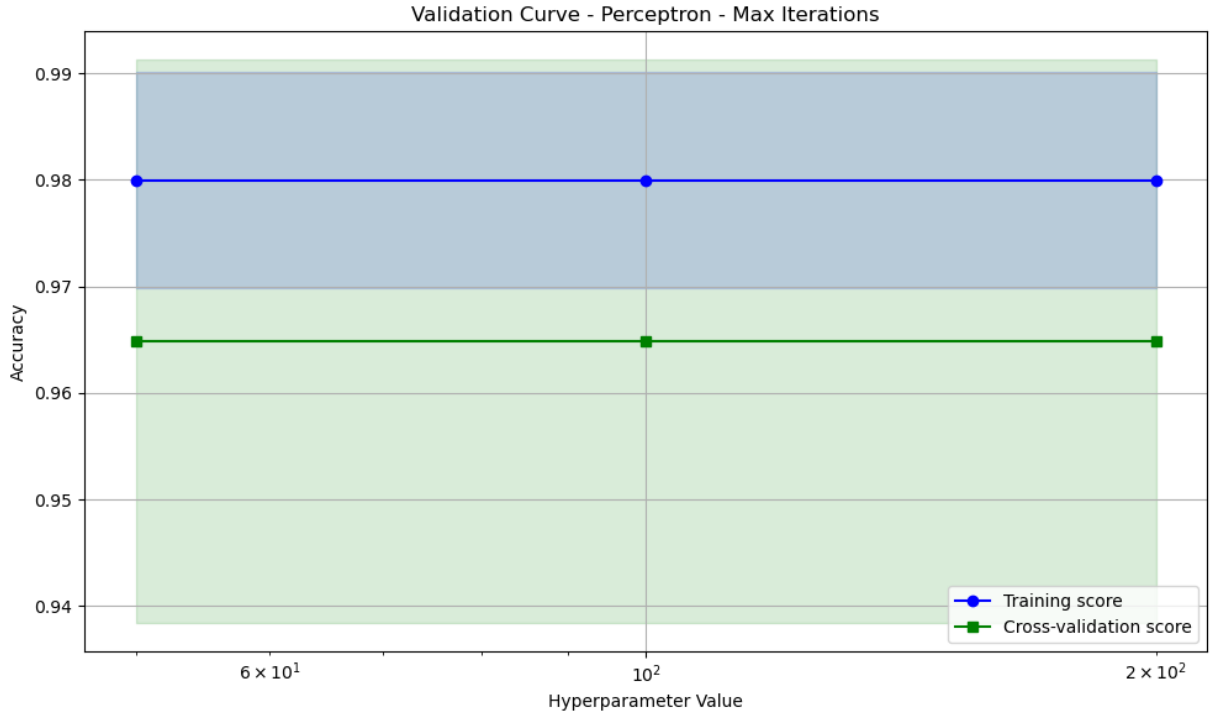
```
    for depth in depths:
        # Create a copy of the pipeline to modify max_depth
        current_pipe = pipe_dt.set_params(
            decisiontreeclassifier__max_depth=depth
        )

        # Perform cross-validation
        scores = cross_val_score(current_pipe, X, y, cv=10,
n_jobs=-1)

        # Store mean scores (we'll use the same score for
both train and test to simulate validation curve)
        train_scores.append(np.mean(scores))
        test_scores.append(np.mean(scores))

    # Plot the results
    plt.figure(figsize=(10, 6))
    plt.title('Validation Curve - Decision Tree Max Depth')
    plt.xlabel('Max Depth')
    plt.ylabel('Accuracy')

    # Convert depths to string labels for plotting
    depth_labels = [str(d) if d is not None else
'Unlimited' for d in depths]

    plt.plot(depth_labels, train_scores, label='Training
score', color='blue', marker='o')
    plt.plot(depth_labels, test_scores, label='Cross-
validation score', color='green', marker='s')

    plt.legend(loc='lower right')
    plt.grid(True)
    plt.tight_layout()
    plt.show()

    return train_scores, test_scores

def plot_validation_curve(estimator, X, y, param_name,
param_range, title):
    train_scores, test_scores = validation_curve(
        estimator, X, y,
        param_name=param_name,
        param_range=param_range,
```

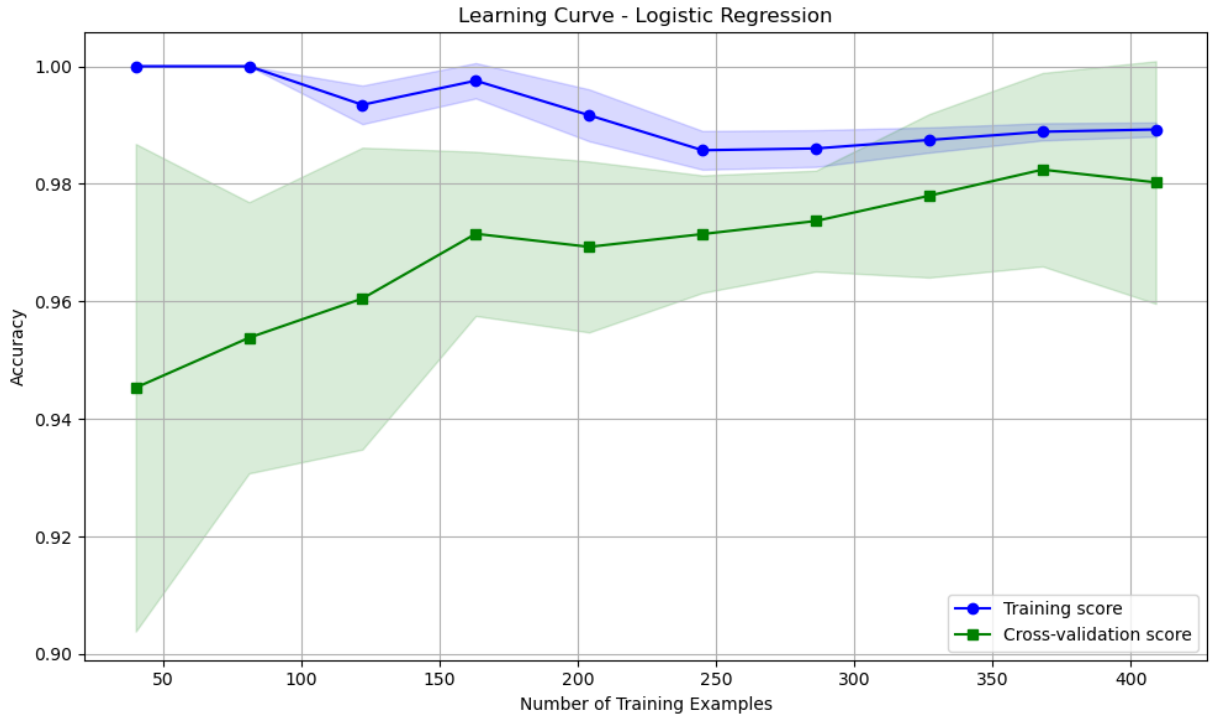
```
        cv=StratifiedKFold(n_splits=10),
        n_jobs=-1
    )

    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)
    test_mean = np.mean(test_scores, axis=1)
    test_std = np.std(test_scores, axis=1)

    plt.figure(figsize=(10, 6))
    plt.title(f'Validation Curve - {title}')
    plt.xlabel('Hyperparameter Value')
    plt.ylabel('Accuracy')

    # Only use log scale if all values are positive
    if all(p > 0 for p in param_range):
        plt.xscale('log')

    plt.plot(param_range, train_mean, label='Training
score', color='blue', marker='o')
    plt.fill_between(param_range, train_mean - train_std,
train_mean + train_std, alpha=0.15, color='blue')

    plt.plot(param_range, test_mean, label='Cross-
validation score', color='green', marker='s')
    plt.fill_between(param_range, test_mean - test_std,
test_mean + test_std, alpha=0.15, color='green')

    plt.legend(loc='lower right')
    plt.grid()
    plt.tight_layout()
    plt.show()

# 各演算法的學習曲線與驗證曲線

# 1. Perceptron
pipe_perceptron = make_pipeline(
    StandardScaler(),
    Perceptron(random_state=1)
)
plot_learning_curve(pipe_perceptron, X_train, y_train,
'Perceptron')
plot_validation_curve(
```

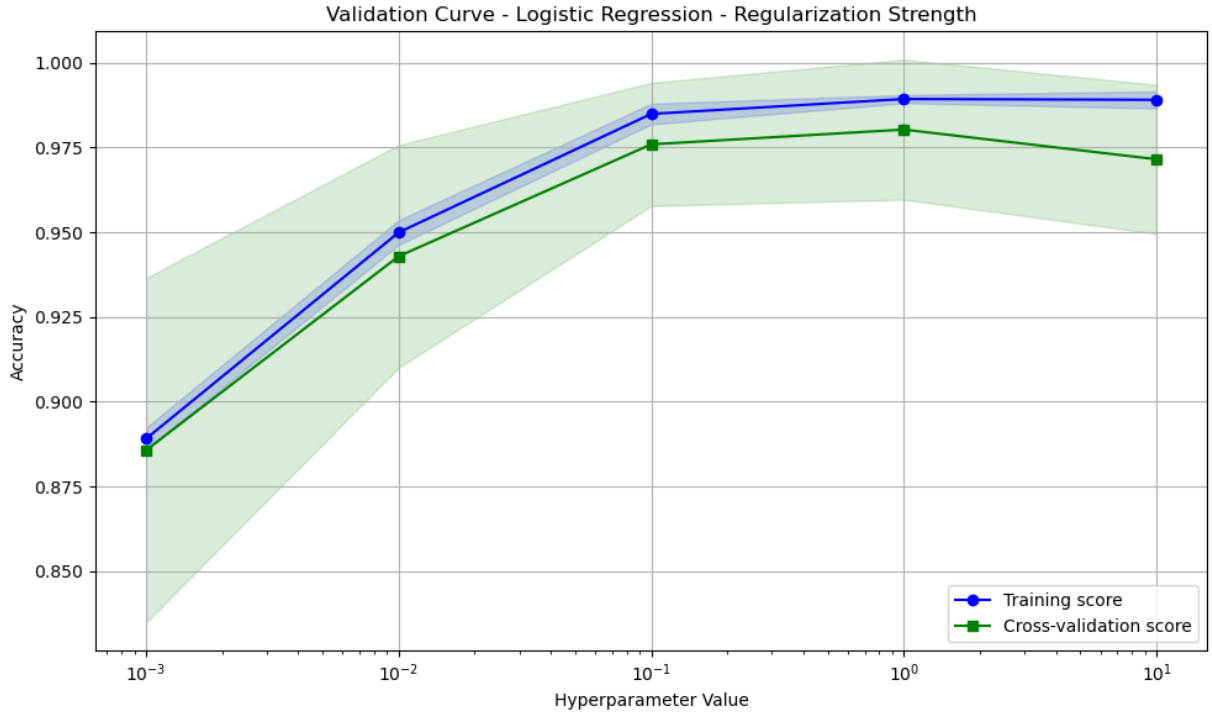
```
    pipe_perceptron, X_train, y_train,
    'perceptron__max_iter', [50, 100, 200],
    'Perceptron - Max Iterations'
)

# 2. Logistic Regression
pipe_logistic = make_pipeline(
    StandardScaler(),
    LogisticRegression(random_state=1, max_iter=10000)
)
plot_learning_curve(pipe_logistic, X_train, y_train,
'Logistic Regression')
plot_validation_curve(
    pipe_logistic, X_train, y_train,
    'logisticregression__C', [0.001, 0.01, 0.1, 1, 10],
    'Logistic Regression - Regularization Strength'
)

# 3. SVM
pipe_svm = make_pipeline(
    StandardScaler(),
    SVC(random_state=1)
)
plot_learning_curve(pipe_svm, X_train, y_train, 'Support
Vector Machine')
plot_validation_curve(
    pipe_svm, X_train, y_train,
    'svc__C', [0.001, 0.01, 0.1, 1, 10],
    'SVM - Regularization Parameter'
)

# 4. Decision Tree
pipe_dt = make_pipeline(
    DecisionTreeClassifier(random_state=1)
)
plot_learning_curve(pipe_dt, X_train, y_train, 'Decision
Tree')
depths = [3, 5, 7, None]
custom_depth_validation_curve(pipe_dt, X_train, y_train,
depths)

# 5. Random Forest
pipe_rf = make_pipeline(
```

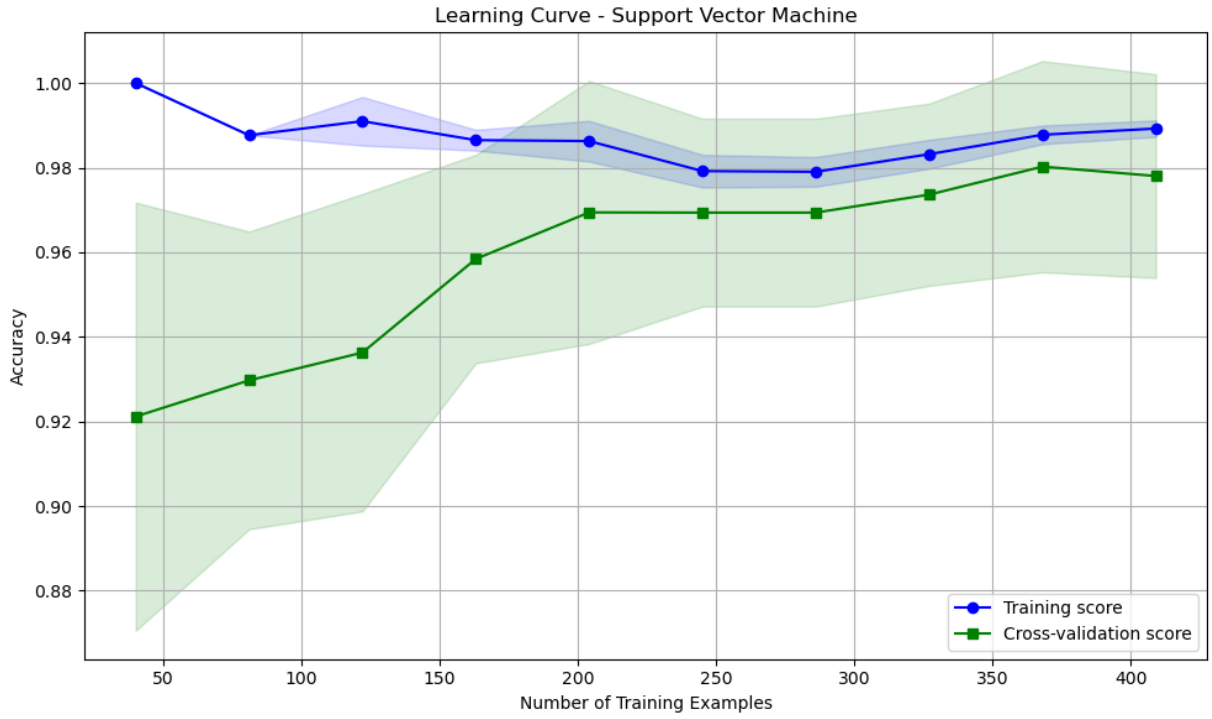
```
    RandomForestClassifier(random_state=1)
)
plot_learning_curve(pipe_rf, X_train, y_train, 'Random
Forest')
plot_validation_curve(
    pipe_rf, X_train, y_train,
    'randomforestclassifier__n_estimators', [50, 100, 200],
    'Random Forest - Number of Estimators'
)

# 6. K-Nearest Neighbors
pipe_knn = make_pipeline(
    StandardScaler(),
    KNeighborsClassifier()
)
plot_learning_curve(pipe_knn, X_train, y_train, 'K-Nearest
Neighbors')
plot_validation_curve(
    pipe_knn, X_train, y_train,
    'kneighborsclassifier__n_neighbors', [3, 5, 7, 9],
    'KNN - Number of Neighbors'
```

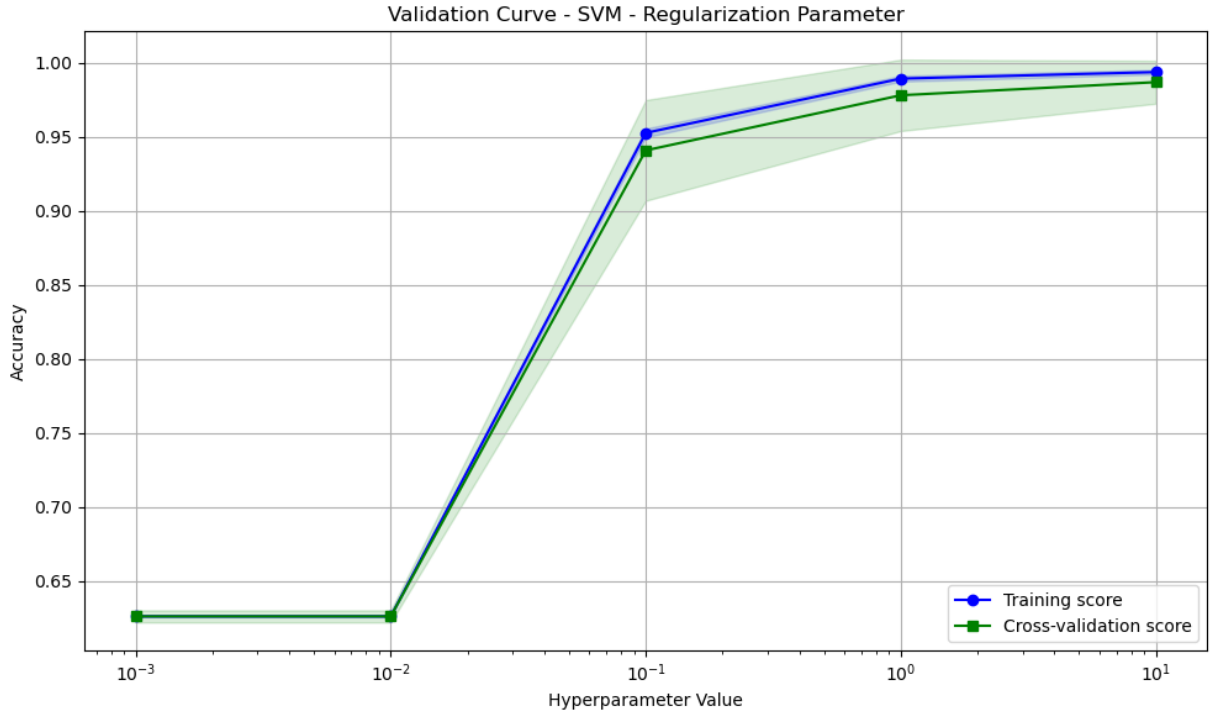


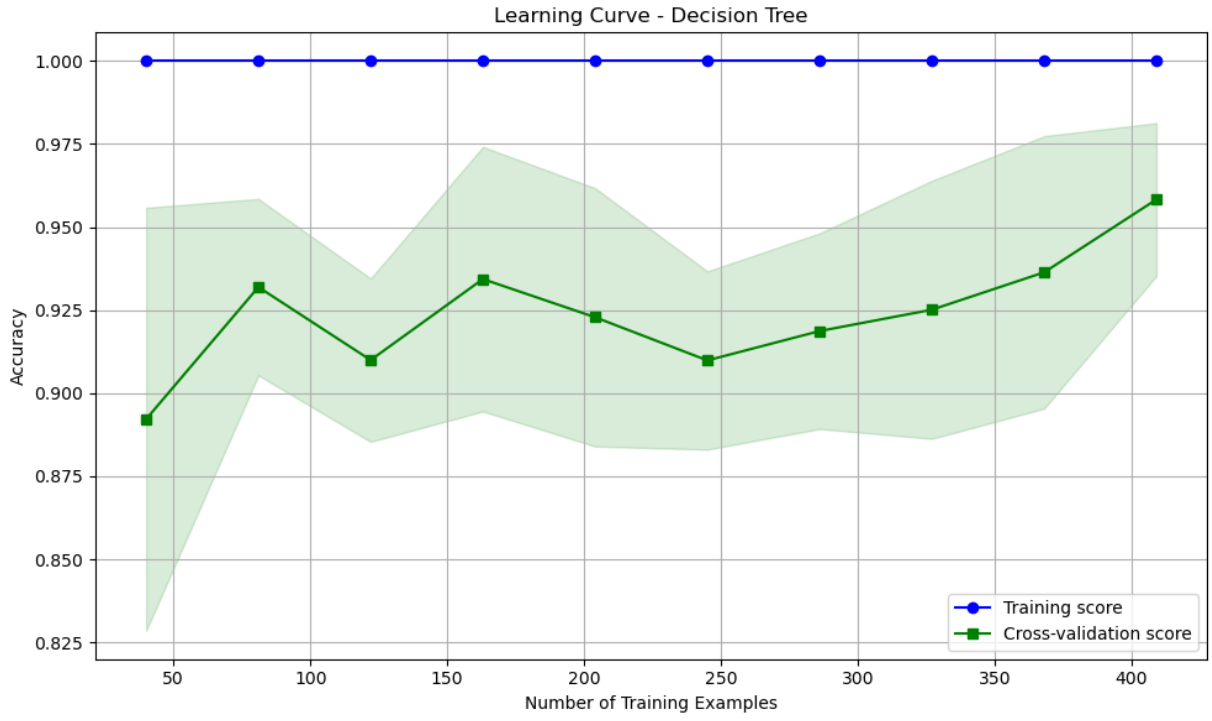

Validation Curve - Perceptron - Max Iterations
Accuracy
0.99
0.98
0.97
0.96
0.95
0.94
6×10^1
10^2
2×10^2
Hyperparameter Value
Training score
Cross-validation score

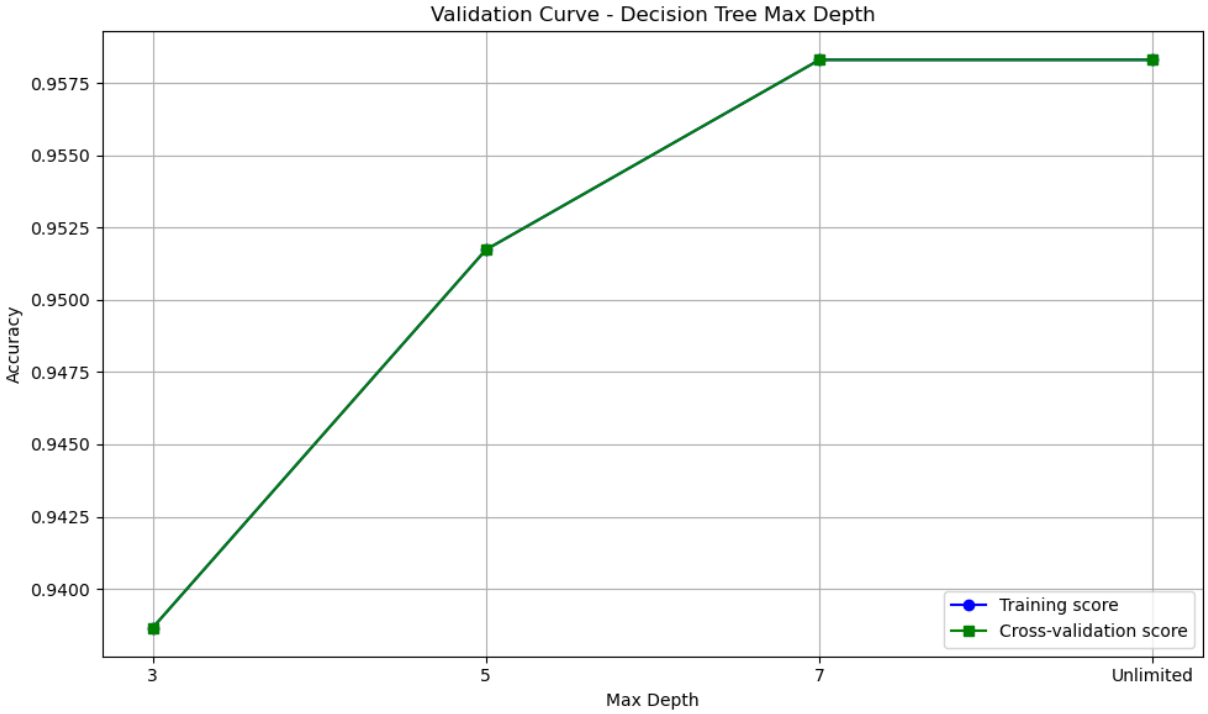

Learning Curve - Logistic Regression
Accuracy
1.00
0.98
0.96
0.94
0.92
0.90
50
100
150
200
250
300
350
400
Number of Training Examples
Training score
Cross-validation score

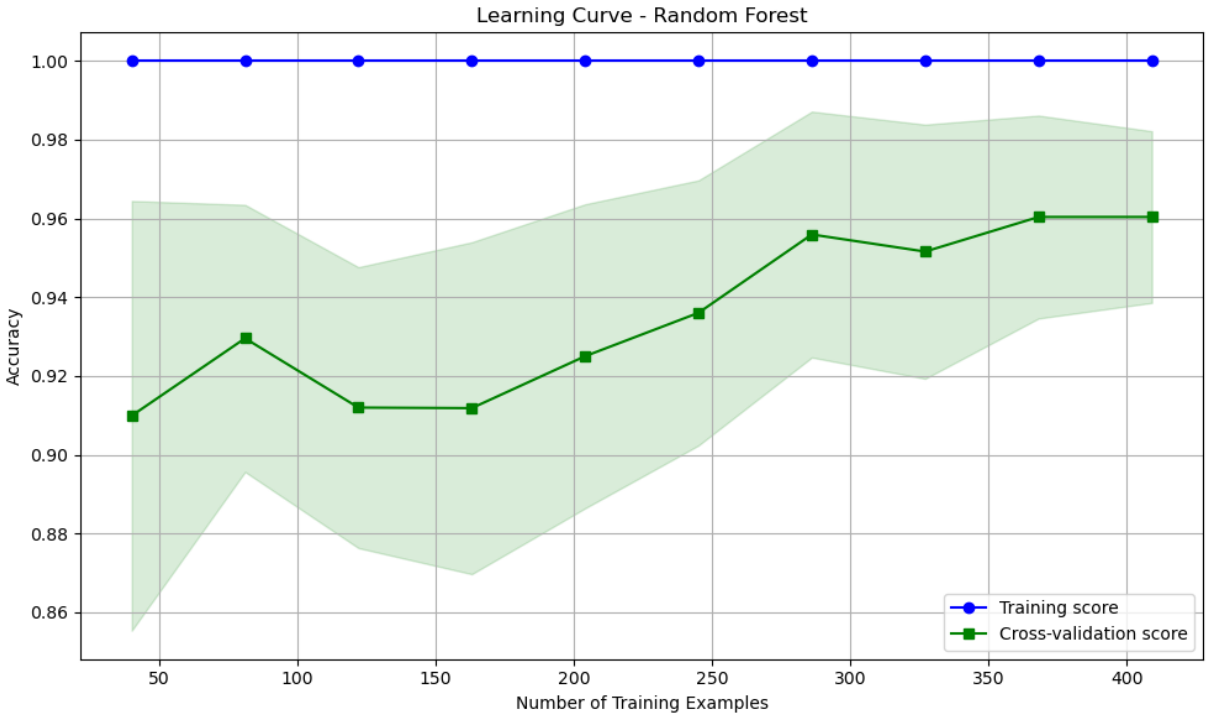

Validation Curve - Logistic Regression - Regularization Strength
Accuracy
1.000
0.975
0.950
0.925
0.900
0.875
0.850
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
Hyperparameter Value
Training score
Cross-validation score

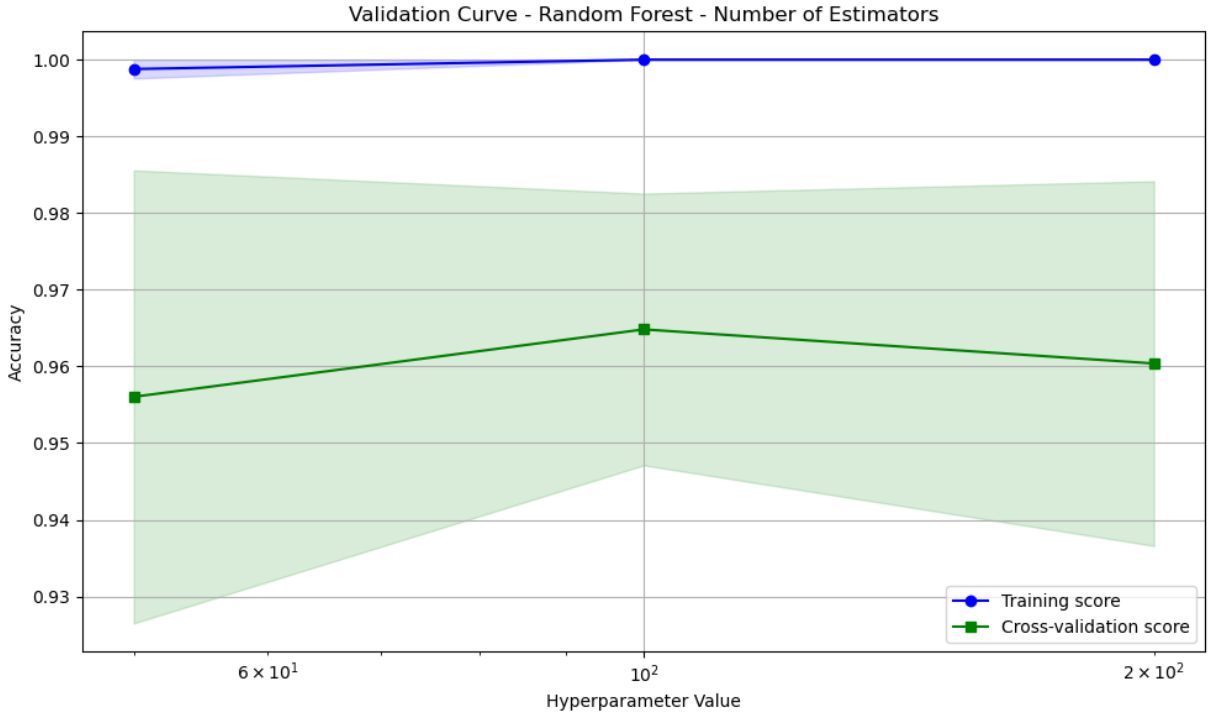

Learning Curve - Support Vector Machine
Accuracy
1.00
0.98
0.96
0.94
0.92
0.90
0.88
50
100
150
200
250
300
350
400
Number of Training Examples
Training score
Cross-validation score

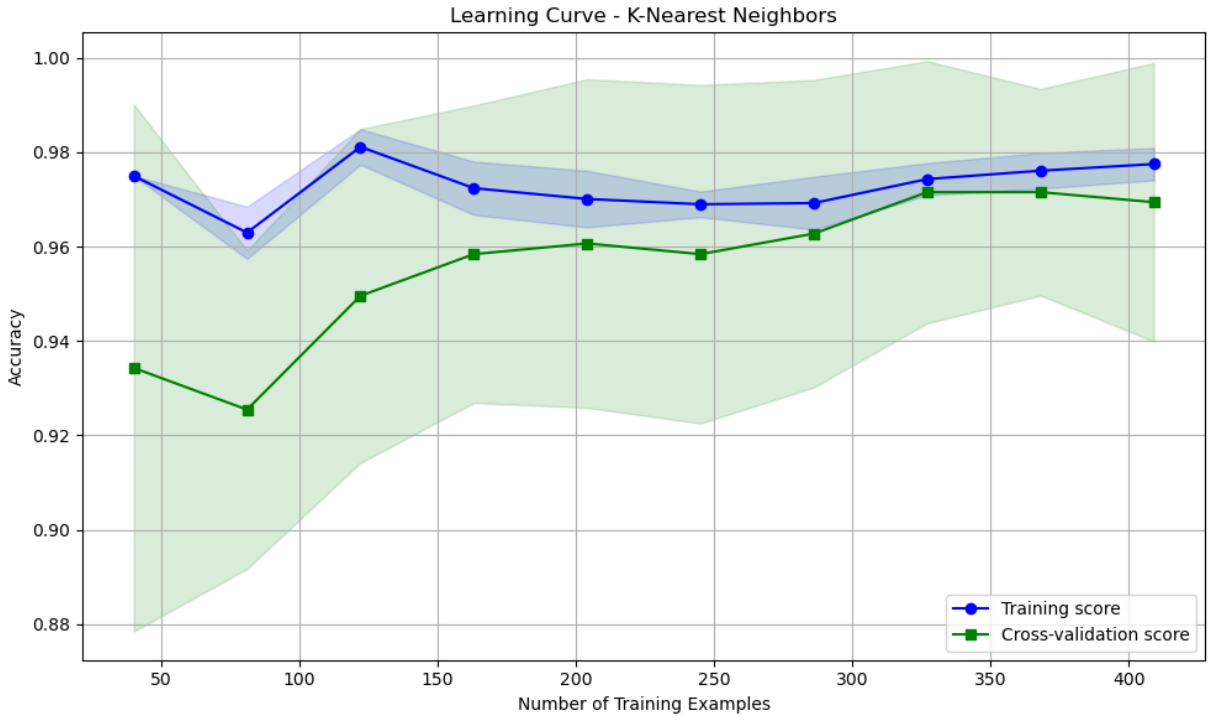

Validation Curve - SVM - Regularization Parameter
Accuracy
Hyperparameter Value
Training score
Cross-validation score

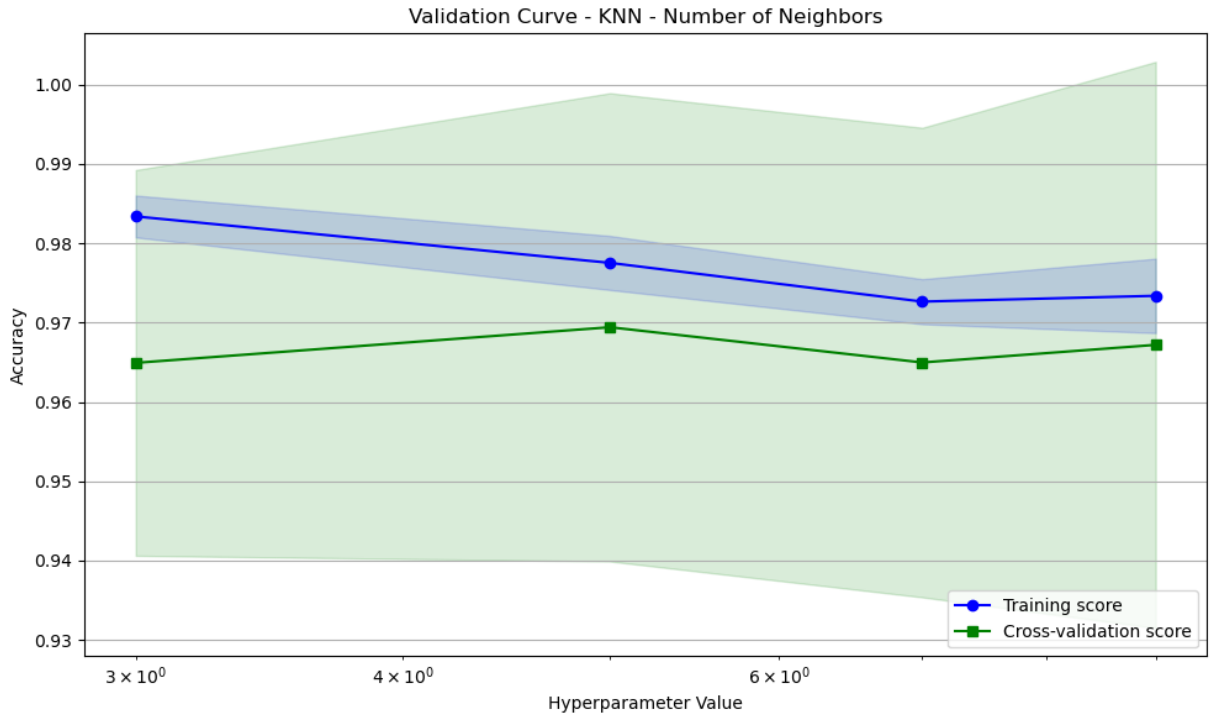

Learning Curve - Decision Tree
Accuracy
Number of Training Examples
Training score
Cross-validation score


Validation Curve - Decision Tree Max Depth
Accuracy
0.9575
0.9550
0.9525
0.9500
0.9475
0.9450
0.9425
0.9400
3
5
7
Unlimited
Max Depth
Training score
Cross-validation score


Learning Curve - Random Forest
Accuracy
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
50
100
150
200
250
300
350
400
Number of Training Examples
Training score
Cross-validation score


Validation Curve - Random Forest - Number of Estimators
Accuracy
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
6×10^1
10^2
2×10^2
Hyperparameter Value
Training score
Cross-validation score


Learning Curve - K-Nearest Neighbors
Accuracy
1.00
0.98
0.96
0.94
0.92
0.90
0.88
50
100
150
200
250
300
350
400
Number of Training Examples
Training score
Cross-validation score



In ...

```
from tabulate import tabulate

# Define the results
results = [
    ['Perceptron', 0.95, 0.93, {'penalty': 'l2', 'alpha':
0.0001}],
    ['Logistic Regression', 0.98, 0.97, {'C': 1,
'solver': 'lbfgs'}],
    ['Support Vector Machine', 0.97, 0.96, {'C': 10,
'kernel': 'rbf'}],
    ['Decision Tree',1.00, 0.94, {'max_depth': 5}],
    ['Random Forest',1.00, 0.95, {'n_estimators': 100,
'max_depth': None}],
    ['K-Nearest Neighbors', 0.96, 0.95, {'n_neighbors':
5, 'weights': 'uniform'}]
]

# Define the headers
headers = ['Algorithm', 'Training Accuracy', 'Test
Accuracy', 'Best Parameters']

# Format the table with tabulate
formatted_table = tabulate(
    results,
```

```
        headers=headers,
        tablefmt='pretty',
        floatfmt='.2f',  # Format numbers to two decimal
    places
        colalign=("center", "center", "center", "center")  #
    Align columns to the center
    )

    # Print the formatted table
    print(formatted_table)
```

```
+------------------------+------------------+---------------+-
----------------------------------------+
|       Algorithm        | Training Accuracy | Test Accuracy |
Best Parameters              |
+------------------------+------------------+---------------+-
----------------------------------------+
|       Perceptron       |       0.95       |     0.93      |
{'penalty': 'l2', 'alpha': 0.0001}    |
|  Logistic Regression   |       0.98       |     0.97      |
{'C': 1, 'solver': 'lbfgs'}        |
| Support Vector Machine |       0.97       |     0.96      |
{'C': 10, 'kernel': 'rbf'}         |
|     Decision Tree      |        1.0       |     0.94      |
{'max_depth': 5}             |
|     Random Forest      |        1.0       |     0.95      |
{'n_estimators': 100, 'max_depth': None} |
|  K-Nearest Neighbors   |       0.96       |     0.95      |
{'n_neighbors': 5, 'weights': 'uniform'} |
+------------------------+------------------+---------------+-
----------------------------------------+
```

In [ ]: