LSE ST510

Author: Jin Zhu

Week 4: Support vector machines

Topics cover

1. Multi-class support vector machines

- one-versus-one approach
- one-versus-rest approach

2. Parameter selection
3. Kernel principal component analysis

We will use the benchmarked Python libraries for support vector machines (SVM) to illustrate parts 1-2.

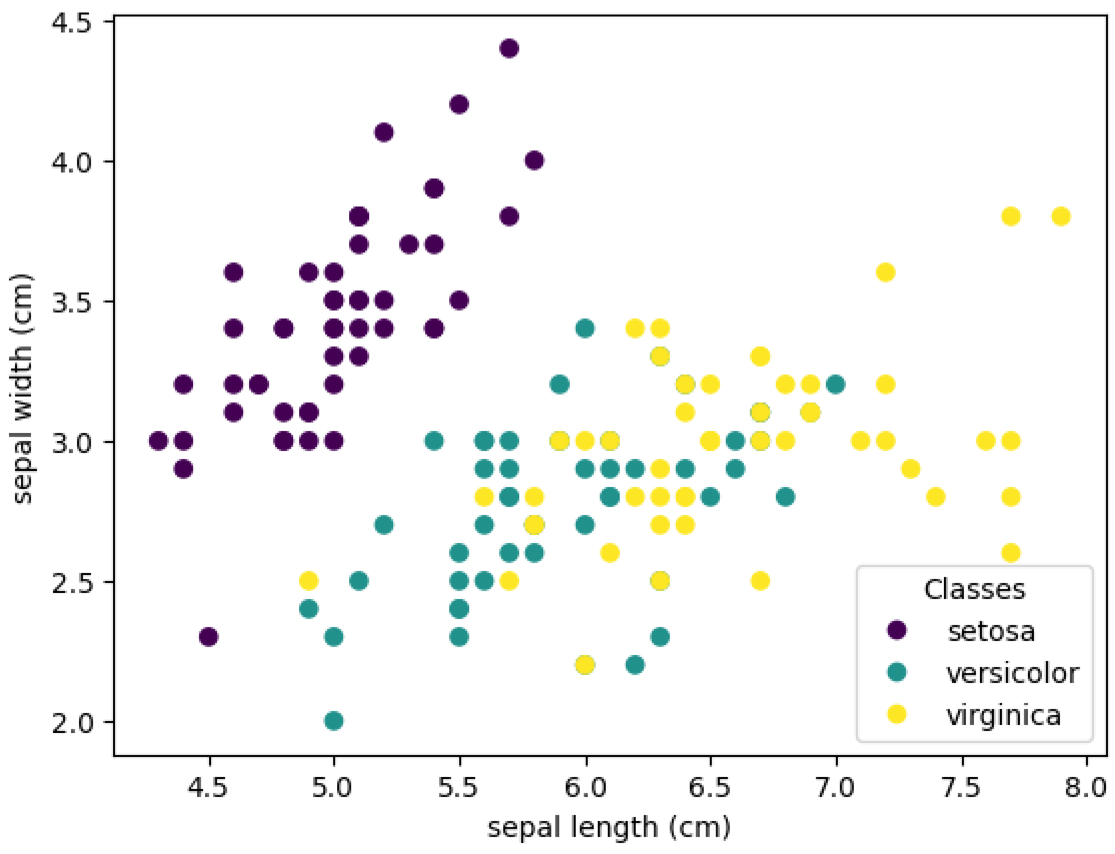
In [ ]:
```python
from sklearn import svm
```

Multi-class support vector machines

Support vector machines (SVMs) are a class of machine learning algorithms that are frequently used nowadays. Named after their method of learning a decision boundary, SVMs are binary classifiers, meaning that they are designed to work with a scenario involving two classes, typically labeled as 0 and 1. However, in practice, many scenarios require multi-class classification.

For example, consider the iris dataset, which includes three species: setosa, versicolor, and virginica, along with measurements related to these species, such as the length and width of the sepal.

In [ ]:
```python
from sklearn import datasets
iris = datasets.load_iris()

import matplotlib.pyplot as plt
_, ax = plt.subplots()
scatter = ax.scatter(iris.data[:, 0], iris.data[:, 1], c=iris.target)
ax.set(xlabel=iris.feature_names[0], ylabel=iris.feature_names[1])
_ = ax.legend(
    scatter.legend_elements()[0], iris.target_names, loc="lower right", title="Classes"
)
```



We would like to use the features of the sepal (e.g., its width and length) to classify its species. How should we accordingly modify the SVM to address these issues?

One-versus-one approach

Basic idea: break down the problem to multiple binary classification problems.

Support there are M classes in total, denoted as $\{1, 2, \ldots, M\}$. The training procedure is:

> For any two classes $i, j \in \{1, 2, \ldots, M\}$ with $i < j$, train a binary SVM.

Therefore, there are $\frac{1}{2}(M-1)M$ binary SVMs.

Prediction procedure:

> For a new input x, we recruit $\frac{1}{2}(M-1)M$ binary SVMs to predict its class. Then, the class with the most votes is predicted.

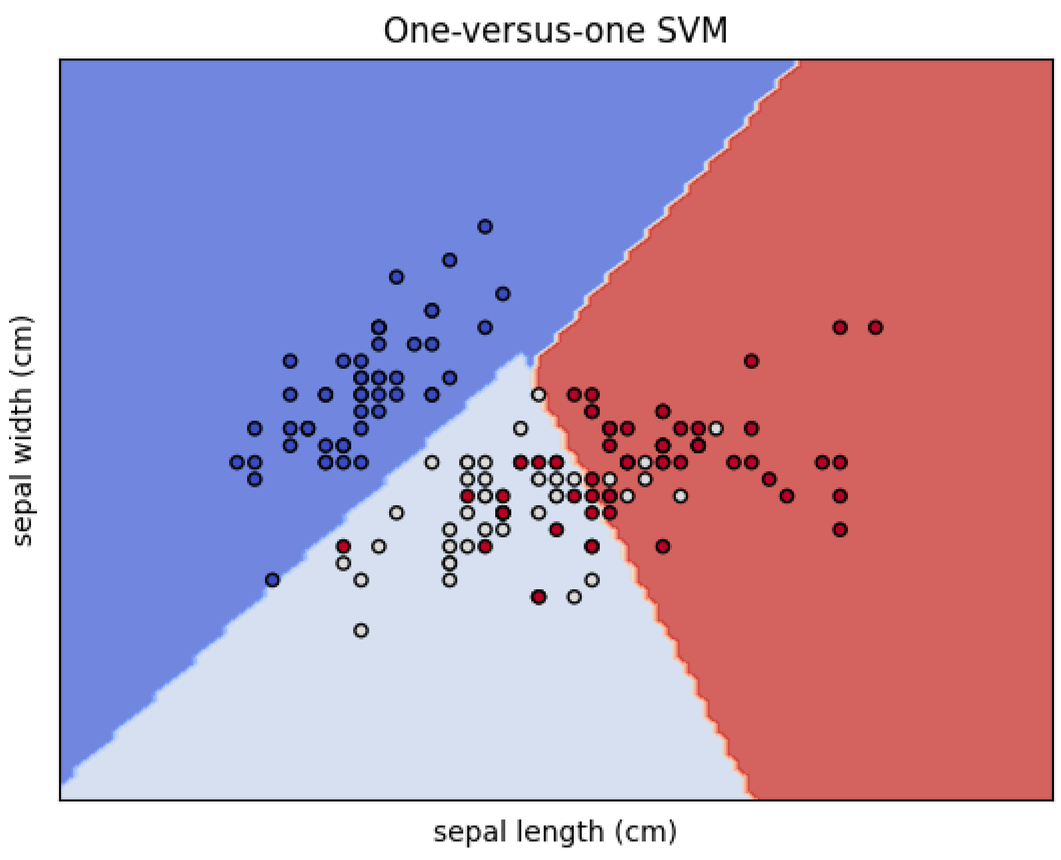
Here is an example of employing the one-versus-one approach with a multi-class support vector machine in Python. For simplicity, the regularization parameter C in the binary SVMs is set to 1.0.

```
In [ ]: # Take the first two features. We could avoid this by using a two-dim dataset
        X = iris.data[:, :2]
        y = iris.target

        # SVM regularization parameter
        C = 1.0

        # Initial the one-versus-one SVM by setting kernel and regularization parameter
        clf = svm.SVC(kernel="linear", C=C)

        # Fit the training data
        clf.fit(X, y)
```

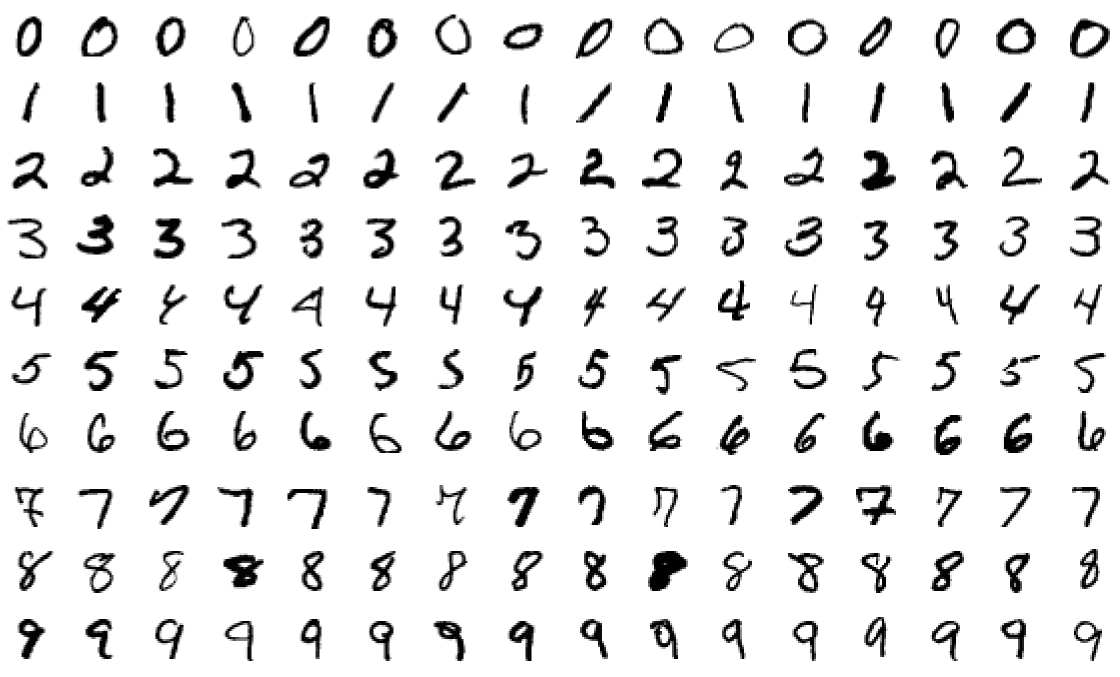
```
Out[ ]: ▾        SVC
        SVC(kernel='linear')
```

Then, we can visualize the fitted SVM model:

```
In [ ]: from sklearn.inspection import DecisionBoundaryDisplay

        fig, ax = plt.subplots(1, 1)
        disp = DecisionBoundaryDisplay.from_estimator(
            clf,
            X,
            response_method="predict",
            cmap=plt.cm.coolwarm,
            alpha=0.8,
            ax=ax,
            xlabel=iris.feature_names[0],
            ylabel=iris.feature_names[1],
        )
        ax.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
        ax.set_xticks(())
        ax.set_yticks(())
        ax.set_title("One-versus-one SVM")
        plt.show()
```



- Disadvantages: One-vs-one classification becomes computationally intensive when M is large because it requires training $\frac{1}{2}(M-1)M$ binary classification models. A typical example where the one-vs-one approach will be computationally intensive is with the MNIST dataset, which has 10 classes to predict. For this dataset, the one-vs-one approach would need to train 55 binary SVMs.



One-vs-Rest approach

Basic idea: distinguish between one label and all the others, where the class prediction with highest score wins.

Training procedure:

> For $i \in \{1, \dots, M\}$, train a binary SVM to classify label i and the other labels.

Consequently, there are M binary SVMs.

Prediction procedure for a new input $\mathbf{x}$:

> For each binary SVM with estimated parameter $\mathbf{w}^{(i)}, b^{(i)}$, compute the classification score $\langle \mathbf{w}^{(i)}, \mathbf{x} \rangle + b^{(i)}$

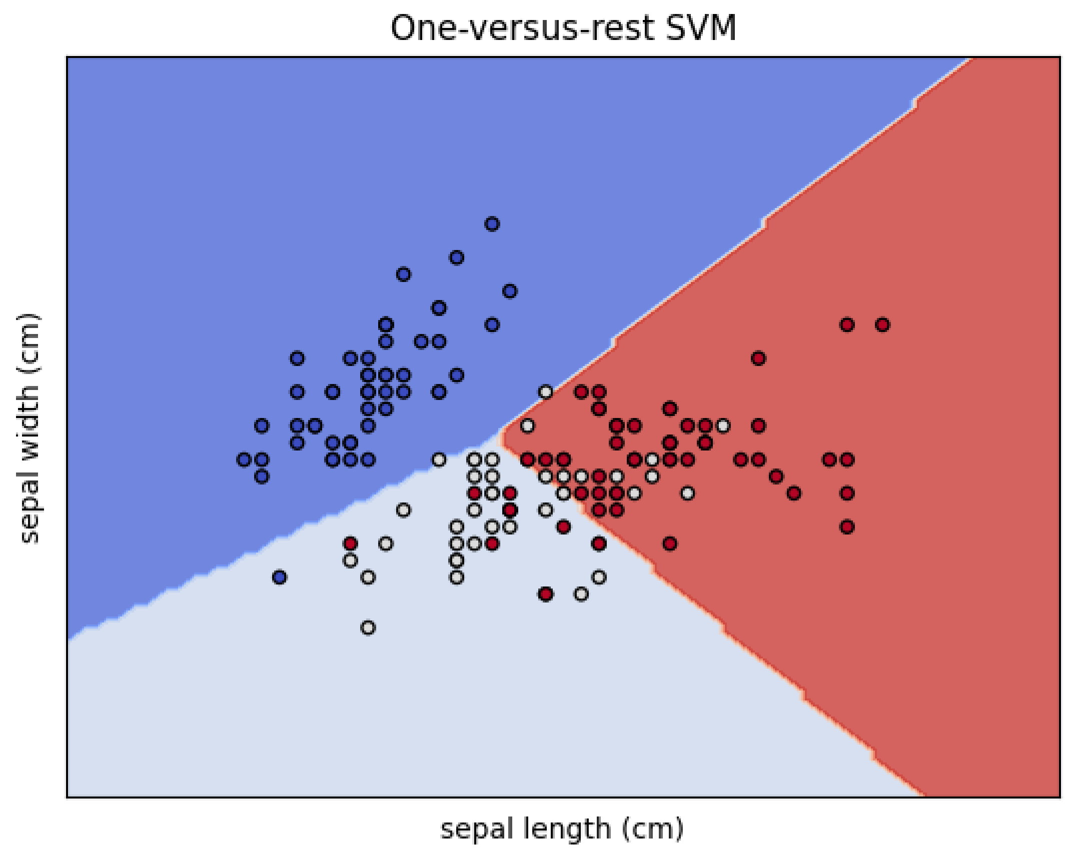
> $\arg\max_i \langle \mathbf{w}^{(i)}, \mathbf{x} \rangle + b^{(i)}$ is the predicted label.

- Advantages: In practice, one-vs-rest classification is usually preferred, since the results are mostly similar, but the runtime is significantly less.

How to use the one-vs-rest approach based multi-class SVM in Python?

The `scikit-learn` library already implements the one-vs-rest approach into `LinearSVC`, which has a similar usage as `SVC`. We illustrate its usage via the iris dataset.

In [ ]:
```python
# Initial one-vs-rest SVM and set the regularization parameter
clf = svm.LinearSVC(C=C, max_iter=10000, dual="auto")

# Fit the training data
clf.fit(X, y)

fig, ax = plt.subplots(1, 1)
disp = DecisionBoundaryDisplay.from_estimator(
    clf,
    X,
    response_method="predict",
    cmap=plt.cm.coolwarm,
    alpha=0.8,
    ax=ax,
    xlabel=iris.feature_names[0],
    ylabel=iris.feature_names[1],
)
ax.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, s=20, edgecolors="k")
ax.set_xticks(())
ax.set_yticks(())
ax.set_title("One-versus-rest SVM")
plt.show()
```



Crammer-singer approach

- Unlike previous two approaches which typically decompose a multi-class problem into multiple independent binary classification

tasks, Crammer-singer (CS) approach proposed a direct method for training multi-class predictors by optimizing a joint objective over all classes.

- While CS approach is interesting from a theoretical perspective as it is consistent, it is seldom used in practice as it rarely leads to better accuracy and is more expensive to compute.

> Suggested reference: Crammer, Koby, and Yoram Singer. "On the algorithmic implementation of multiclass kernel-based vector machines." Journal of machine learning research 2.Dec (2001): 265-292.

Parameter selection

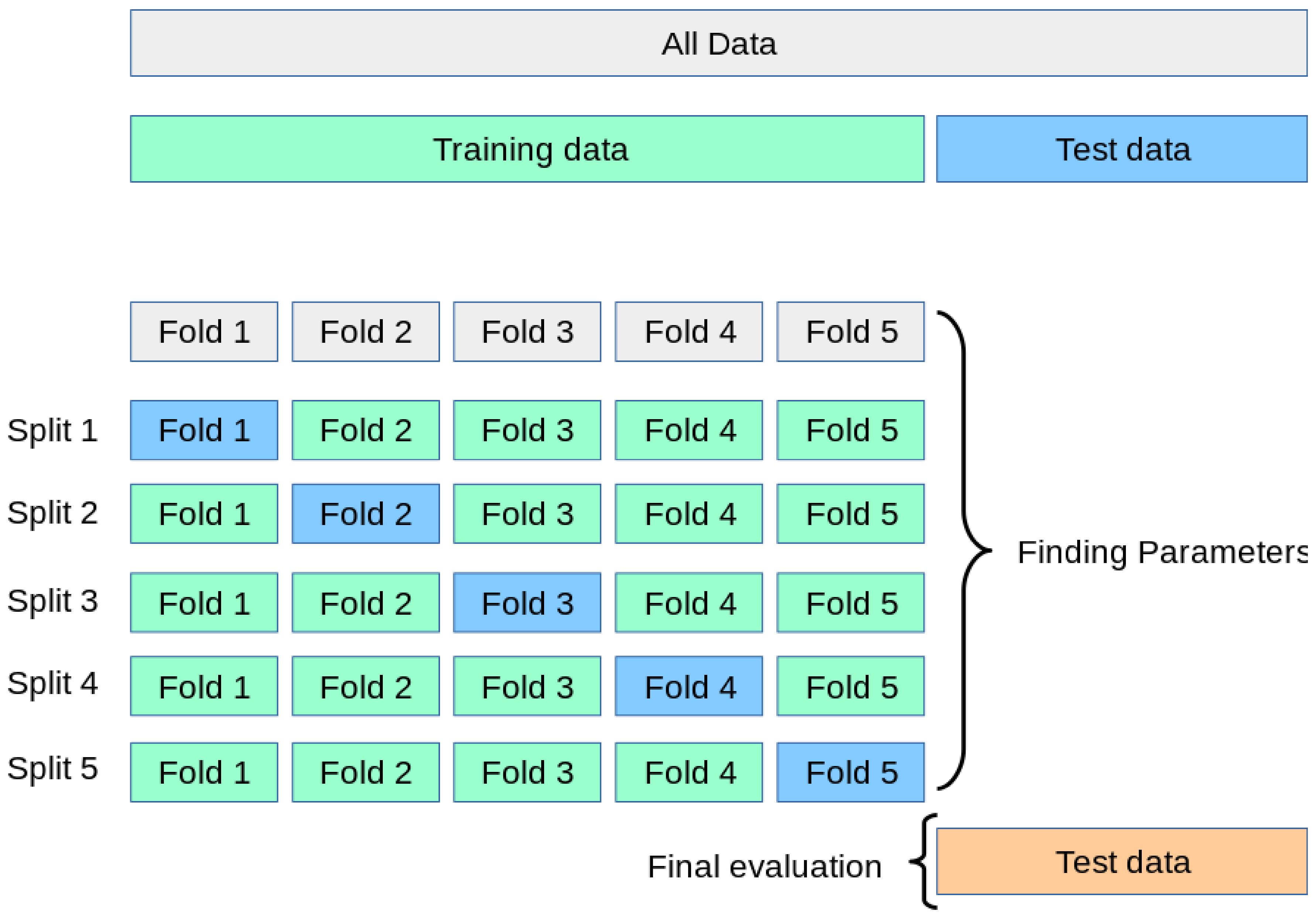
According to the soft classification formulation of SVM:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i \text{ s.t. } y_i(\langle \mathbf{w},\mathbf{x}_i\rangle + b) \geq 1-\xi_i, \xi_i \geq 0 \forall i,$$

The C parameter trades off correct classification of training examples against the maximization of the decision function's margin. For larger values of C, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. A lower C will encourage a larger margin, therefore a simpler decision function, at the cost of training accuracy. In other words, C acts as a regularization parameter in the SVM. The basic guidelines are summarized below:

- If the underlying model is associated with large noise, then you should decrease it: decreasing C corresponds to more regularization.
- Otherwise, you should increase C since the hard classification assumption is likely to hold, and we don't need as much regularization.

For `SVC` and `LinearSVC`, $C = 1$ by default, and it's a reasonable default choice. Given a dataset, a more reasonable method for selecting C is through cross-validation.



The simplest way to use cross-validation is to call the `cross_val_score` helper function in `scikit-learn` on the estimator and the dataset.

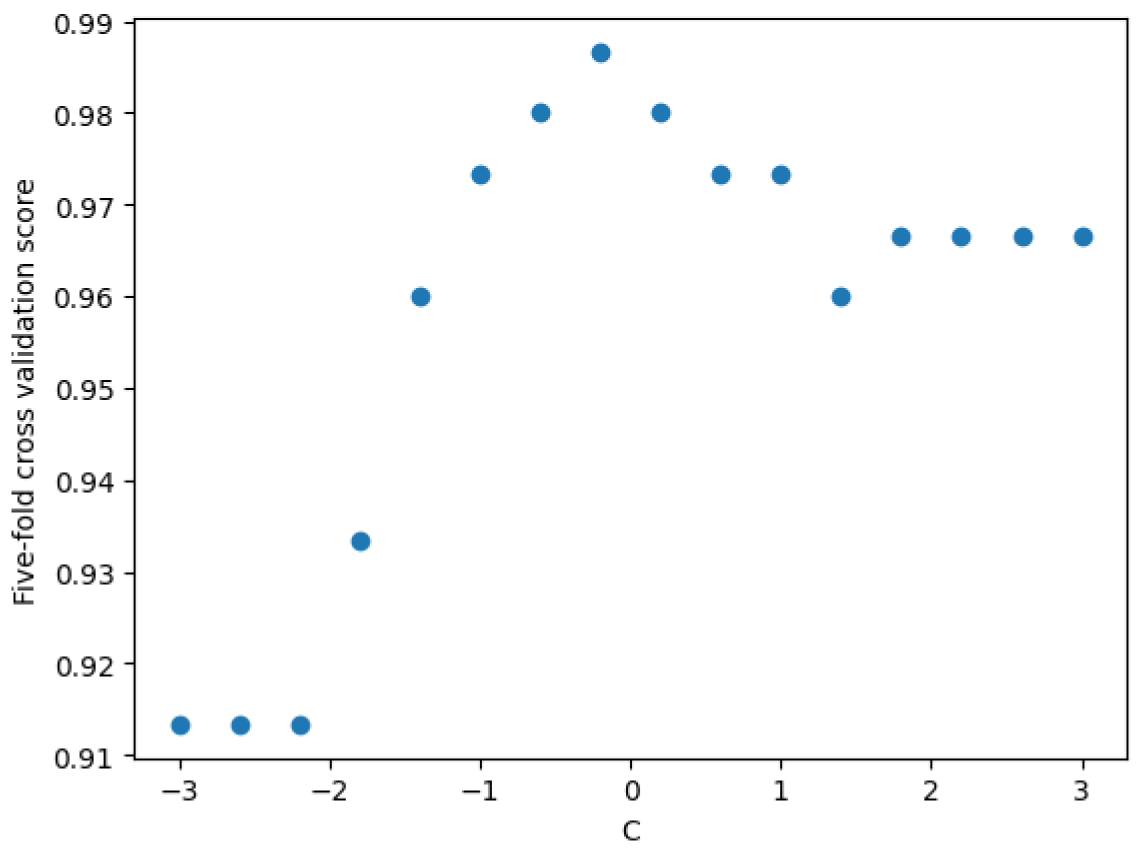
The following example demonstrates how to estimate the accuracy of a linear kernel support vector machine on the iris dataset. This is achieved by splitting the data, fitting a model, and computing the score 5 consecutive times, with different splits each time:

In [ ]:
```
from sklearn.model_selection import cross_val_score
import numpy as np
from sklearn import datasets
X, y = datasets.load_iris(return_X_y=True)

C_list = np.logspace(start=-3, stop=3, num=16)
print(np.round(C_list, 4))
```

```
[1.000000e-03 2.500000e-03 6.300000e-03 1.580000e-02 3.980000e-02
 1.000000e-01 2.512000e-01 6.310000e-01 1.584900e+00 3.981100e+00
 1.000000e+01 2.511890e+01 6.309570e+01 1.584893e+02 3.981072e+02
 1.000000e+03]
```

Above are 32 possible values of C that range within the interval $[10^{-3}, 10^{3}]$. We will choose the value that leads to the highest cross-validation score. Below is the implementation for this.

In [ ]:
```
clf = svm.SVC(kernel='linear', random_state=42)
score_list = []
for C in C_list:
    clf.set_params(C = C)
    scores = cross_val_score(clf, X, y, cv=5)
    score_list.append(scores.mean())

plt.scatter(np.log10(np.array(C_list)), np.array(score_list))
plt.xlabel("C")
plt.ylabel("Five-fold cross validation score")
plt.show()
```



We can see that a moderate value of C can lead to the highest cross-validation score for the iris dataset.

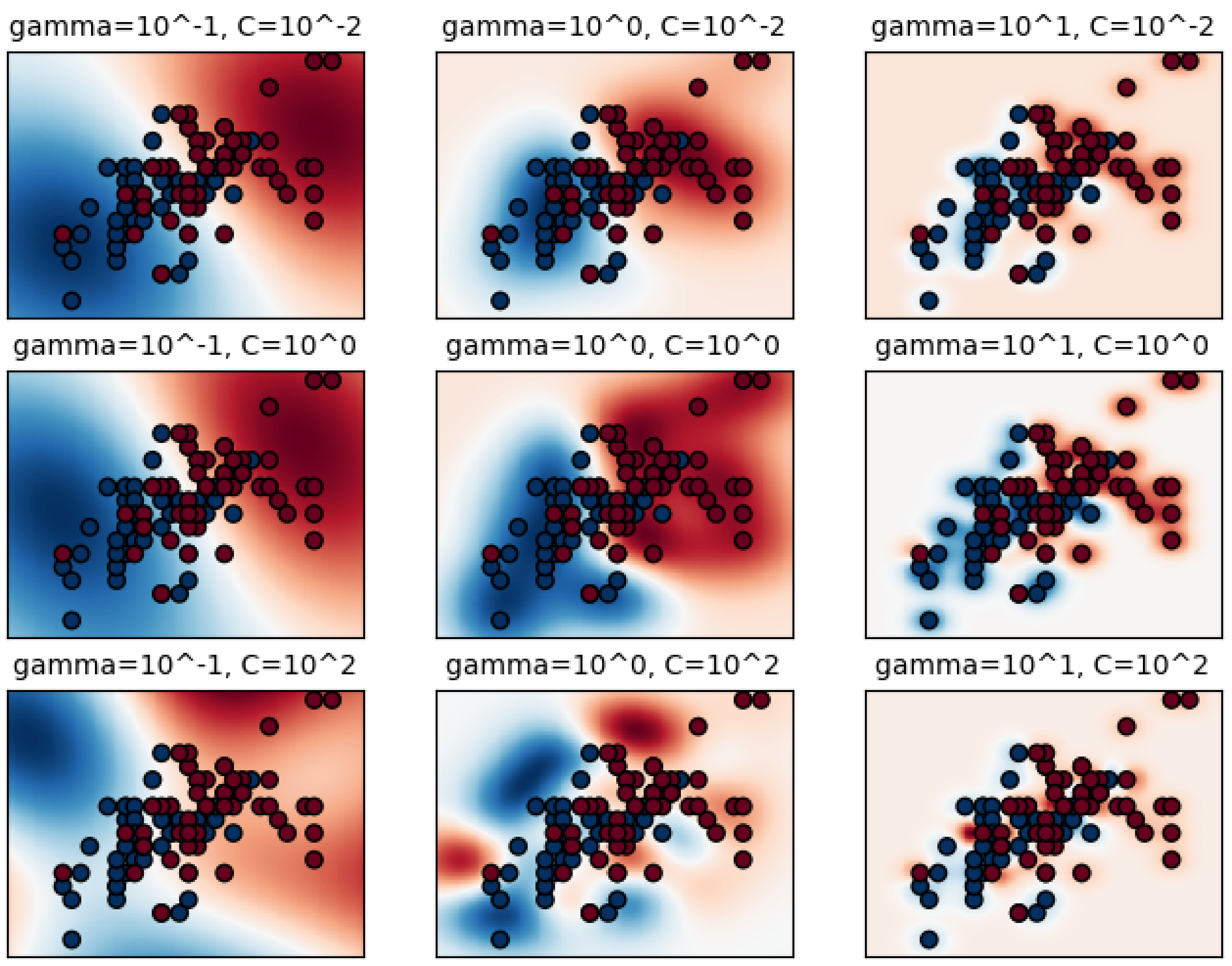
Other important tuning parameters for SVMs are the parameters in the kernel function. The most common kernel functions are:

- Polynomial kernel: $(\gamma\langle\mathbf{x},\mathbf{x}'\rangle + r)^d$
- Radial basis function (RBF) kernel: $\exp(-\gamma|\mathbf{x}-\mathbf{x}'|^2)$, where $\gamma > 0$

Next, we will illustrate the effect of the parameter γ of the RBF kernel.

Intuitively, the γ parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. The γ parameter can be seen as the inverse of the radius of influence of samples selected by the model as support vectors. When γ is very small, the model is too constrained and cannot capture the complexity or 'shape' of the data. The region of influence of any selected support vector would include the whole training set, resulting in a model that behaves similarly to a linear model with a set of hyperplanes that separate the centers of high density of any pair of two classes. To illustrate it, we present a heatmap of the classifier's boundary as a function of C and γ.

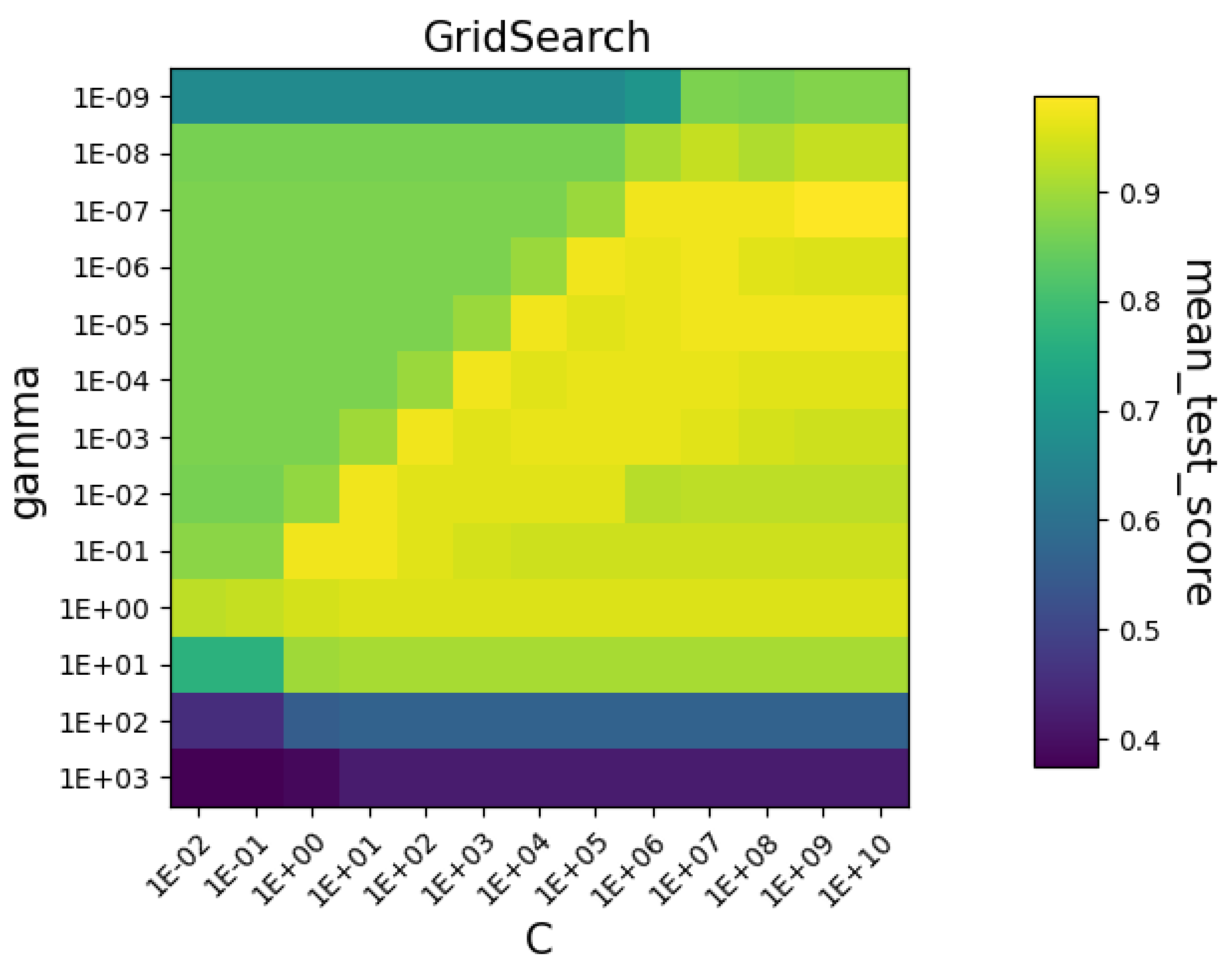
In [ ]:
```python
X_2d = X[:, :2]
X_2d = X_2d[y > 0]
y_2d = y[y > 0]
y_2d -= 1

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X = scaler.fit_transform(X)
X_2d = scaler.fit_transform(X_2d)

from sklearn.model_selection import GridSearchCV

C_2d_range = [1e-2, 1, 1e2]
gamma_2d_range = [1e-1, 1, 1e1]
classifiers = []
for C in C_2d_range:
    for gamma in gamma_2d_range:
        clf = svm.SVC(C=C, gamma=gamma)
        clf.fit(X_2d, y_2d)
        classifiers.append((C, gamma, clf))

plt.figure(figsize=(8, 6))
xx, yy = np.meshgrid(np.linspace(-3, 3, 200), np.linspace(-3, 3, 200))
for k, (C, gamma, clf) in enumerate(classifiers):
    # evaluate decision function in a grid
    Z = clf.decision_function(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    # visualize decision function for these parameters
    plt.subplot(len(C_2d_range), len(gamma_2d_range), k + 1)
    plt.title("gamma=10^%d, C=10^%d" % (np.log10(gamma), np.log10(C)), size="medium")

    # visualize parameter's effect on decision function
    plt.pcolormesh(xx, yy, -Z, cmap=plt.cm.RdBu)
    plt.scatter(X_2d[:, 0], X_2d[:, 1], c=y_2d, cmap=plt.cm.RdBu_r, edgecolors="k")
    plt.xticks(())
    plt.yticks(())
    plt.axis("tight")
```



Thus, we search for C and γ simultaneously and visualize the cross-validation score in a heatmap.

In [ ]:

```python
# select C and gamma
C_range = np.logspace(-2, 10, 13)
gamma_range = np.logspace(-9, 3, 13)
param_grid = dict(gamma=gamma_range, C=C_range)
grid = GridSearchCV(svm.SVC(), param_grid=param_grid, cv=5)
grid.fit(X, y)

# visualize cross validation score
import pandas as pd
def make_heatmap(ax, gs, is_sh=False, make_cbar=False):
    """Helper to make a heatmap."""
    results = pd.DataFrame(gs.cv_results_)
    results[["param_C", "param_gamma"]] = results[["param_C", "param_gamma"]].astype(
        np.float64
    )
    if is_sh:
        scores_matrix = results.sort_values("iter").pivot_table(
            index="param_gamma",
            columns="param_C",
            values="mean_test_score",
            aggfunc="last",
        )
    else:
        scores_matrix = results.pivot(
            index="param_gamma", columns="param_C", values="mean_test_score"
        )

    im = ax.imshow(scores_matrix)

    ax.set_xticks(np.arange(len(C_range)))
    ax.set_xticklabels(["{:.0E}".format(x) for x in C_range])
    ax.set_xlabel("C", fontsize=15)

    ax.set_yticks(np.arange(len(gamma_range)))
    ax.set_yticklabels(["{:.0E}".format(x) for x in gamma_range])
    ax.set_ylabel("gamma", fontsize=15)

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")

    if is_sh:
        iterations = results.pivot_table(
            index="param_gamma", columns="param_C", values="iter", aggfunc="max"
        ).values
        for i in range(len(gamma_range)):
            for j in range(len(C_range)):
                ax.text(
                    j,
                    i,
                    iterations[i, j],
                    ha="center",
                    va="center",
                    color="w",
                    fontsize=20,
                 )

    if make_cbar:
        fig.subplots_adjust(right=0.8)
        cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
        fig.colorbar(im, cax=cbar_ax)
        cbar_ax.set_ylabel("mean_test_score", rotation=-90, va="bottom", fontsize=15)

fig, ax = plt.subplots(ncols=1, sharey=True)
make_heatmap(ax, grid, make_cbar=True)
ax.set_title("GridSearch", fontsize=15)
plt.show()
```

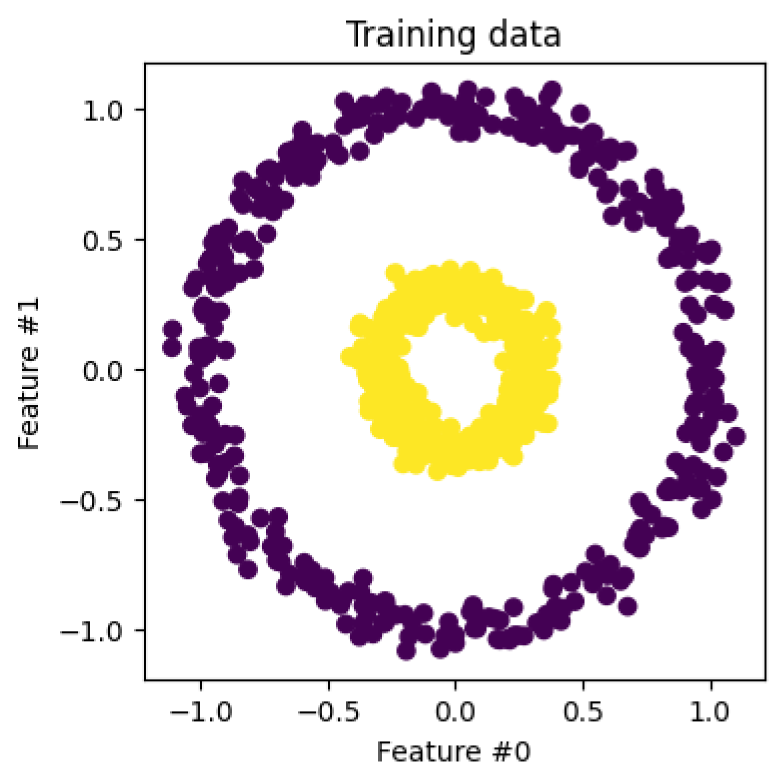


Kernel principal component analysis

In this part, we will introduce kernel principal component analysis (KPCA), which is an extension of principal component analysis (PCA) that utilizes the techniques of kernel methods. By using a kernel, the originally linear operations of PCA are performed in a reproducing kernel Hilbert space, showcasing distinctive advantages not shared by PCA. To illustrate this, we create a dataset comprised of two nested circles.

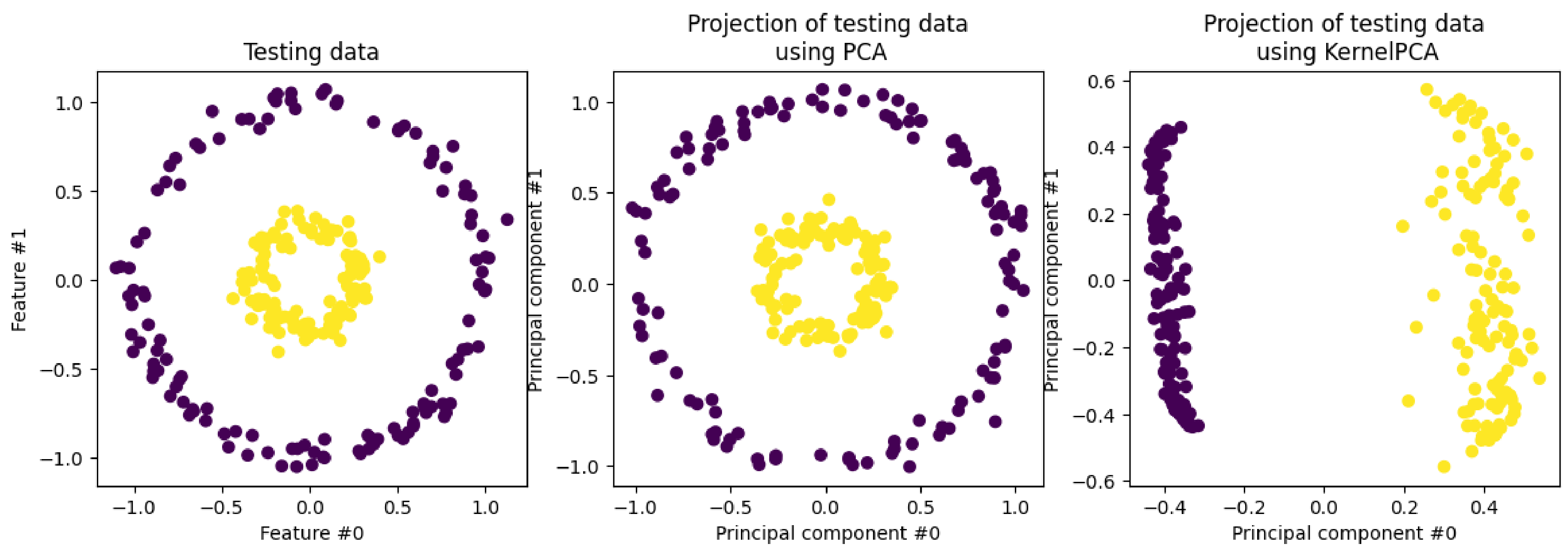
In [ ]:
```
from sklearn.datasets import make_circles
from sklearn.model_selection import train_test_split

X, y = make_circles(n_samples=1000, factor=0.3, noise=0.05, random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=0)

import matplotlib.pyplot as plt

_, ax = plt.subplots(ncols=1, sharex=True, sharey=True, figsize=(4, 4))

ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
ax.set_ylabel("Feature #1")
ax.set_xlabel("Feature #0")
ax.set_title("Training data")
```

Out[ ]:
```
Text(0.5, 1.0, 'Training data')
```



Now, we will use PCA both with and without a kernel to observe the effect of using such a kernel. The kernel used here is an RBF kernel. The `PCA` and `KernelPCA` implementations in `scikit-learn` are utilized for our study.

In [ ]:
```python
from sklearn.decomposition import PCA, KernelPCA

pca = PCA(n_components=2)
kernel_pca = KernelPCA(
    n_components=2, kernel="rbf", gamma=3, fit_inverse_transform=True, alpha=0.1
)

X_test_pca = pca.fit(X_train).transform(X_test)
X_test_kernel_pca = kernel_pca.fit(X_train).transform(X_test)

fig, (orig_data_ax, pca_proj_ax, kernel_pca_proj_ax) = plt.subplots(
    ncols=3, figsize=(14, 4)
)

orig_data_ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test)
orig_data_ax.set_ylabel("Feature #1")
orig_data_ax.set_xlabel("Feature #0")
orig_data_ax.set_title("Testing data")

pca_proj_ax.scatter(X_test_pca[:, 0], X_test_pca[:, 1], c=y_test)
pca_proj_ax.set_ylabel("Principal component #1")
pca_proj_ax.set_xlabel("Principal component #0")
pca_proj_ax.set_title("Projection of testing data\n using PCA")

kernel_pca_proj_ax.scatter(X_test_kernel_pca[:, 0], X_test_kernel_pca[:, 1], c=y_test)
kernel_pca_proj_ax.set_ylabel("Principal component #1")
kernel_pca_proj_ax.set_xlabel("Principal component #0")
_ = kernel_pca_proj_ax.set_title("Projection of testing data\n using KernelPCA")
```



Examining the PCA-derived projection depicted in the central image, we notice the absence of scaling alterations; the dataset, composed of two concentric circles centered at zero, retains its original isotropy. However, a rotation of the data is evident. Consequently, this type of projection proves ineffective for employing a linear classifier to differentiate between the two classes.

Employing a kernel facilitates a non-linear projection. Specifically, through the application of an RBF kernel, we anticipate the dataset to expand in such a manner that it largely maintains the relative proximities of data point pairs that are nearby in the initial space. This effect is visible in the right-hand figure, where data points belonging to the same class are more closely grouped than those of the opposing class, effectively disentangling the two sets of samples. This arrangement now permits the application of a linear classifier to segregate the samples into their respective classes.