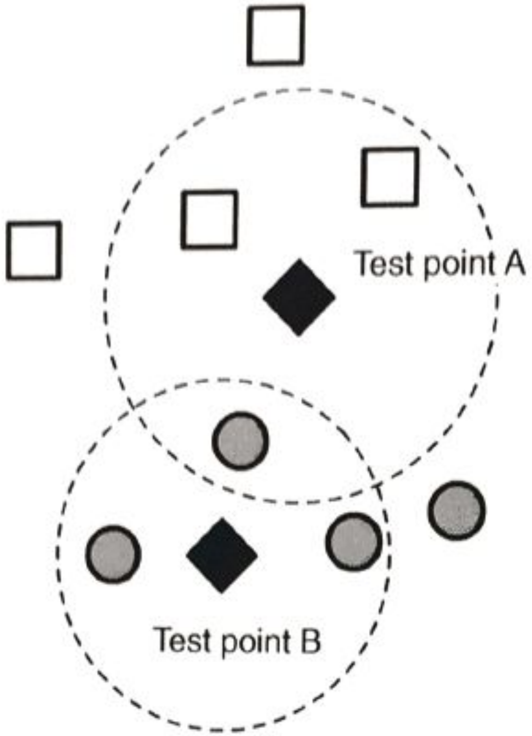
Non-Probabilistic Classification Algorithms (I)

K-Nearest Neighbors

- Use the majority class of K sample points closed to the test point A

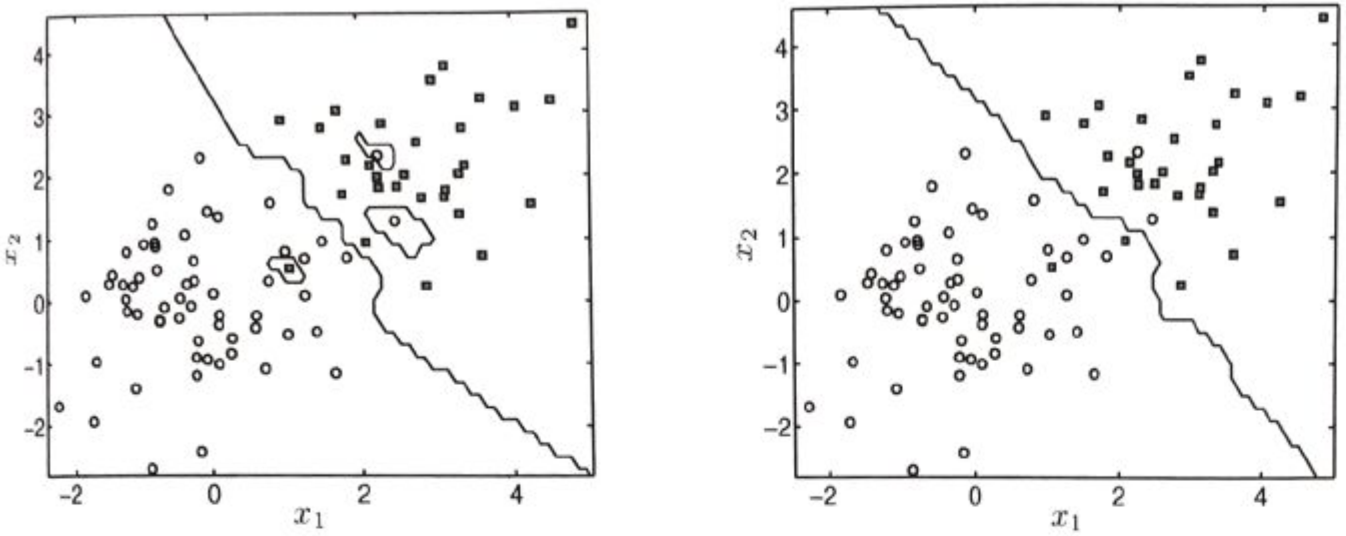


K-Nearest Neighbors (cont.)

- Simple
- Multi-class classification
- Issues
  - How to make sure majority votes exist?
  - How to determine the ‘distance’ between two dataset points?
    - Euclidean distance
    - Minkowski distance
  - How to choose K (hyperparameter)?

Choice of K

Overfitting the noise when K is too small



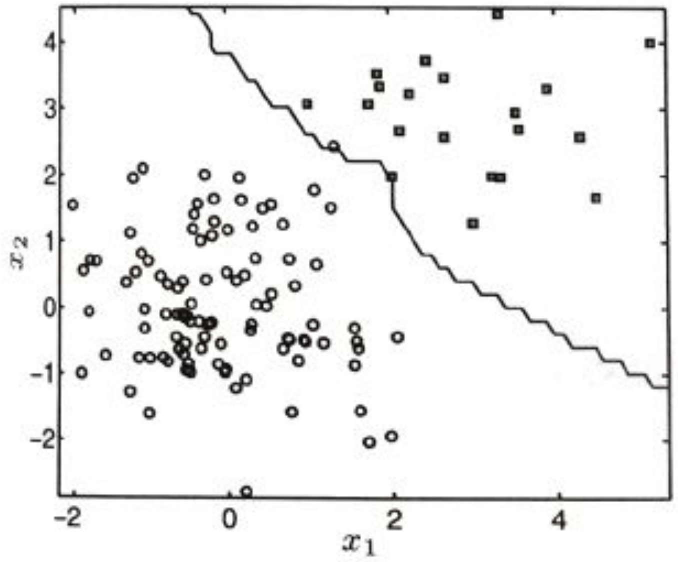
(a) Decision boundary when $K = 1$.

(b) Decision boundary when $K = 5$.

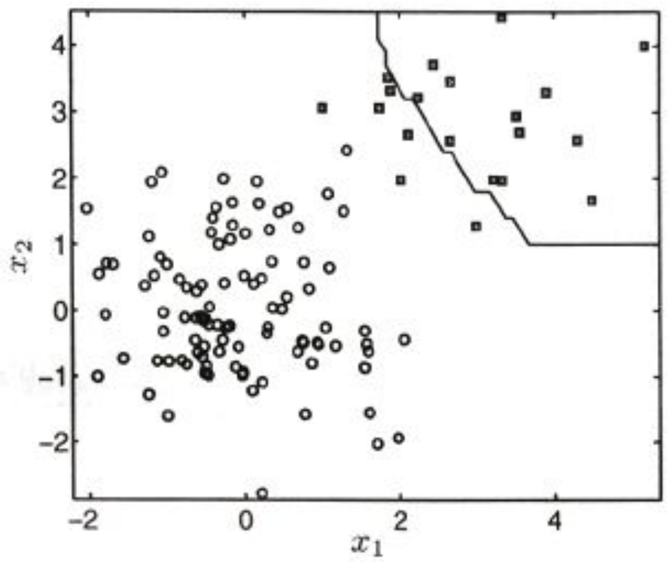
FIGURE 5.9 Binary classification dataset and decision boundaries for $K = 1$ and $K = 5$.

Choice of K (Cont.)

The pattern is lost due to imbalanced dataset when K is too big



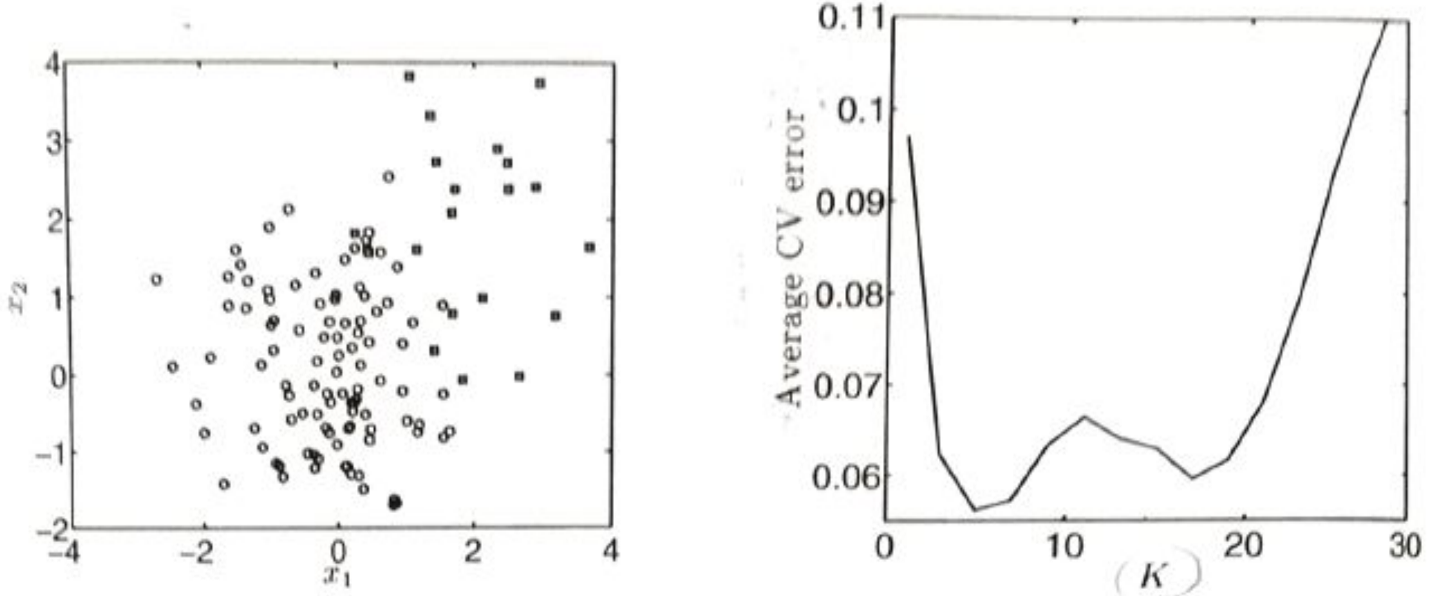
(a) Decision boundary when $K = 5$.



(b) Decision boundary when $K = 39$.

FIGURE 5.10 Second binary classification dataset and decision boundaries for $K = 5$ and $K = 39$.

Hyper-parameter Tuning



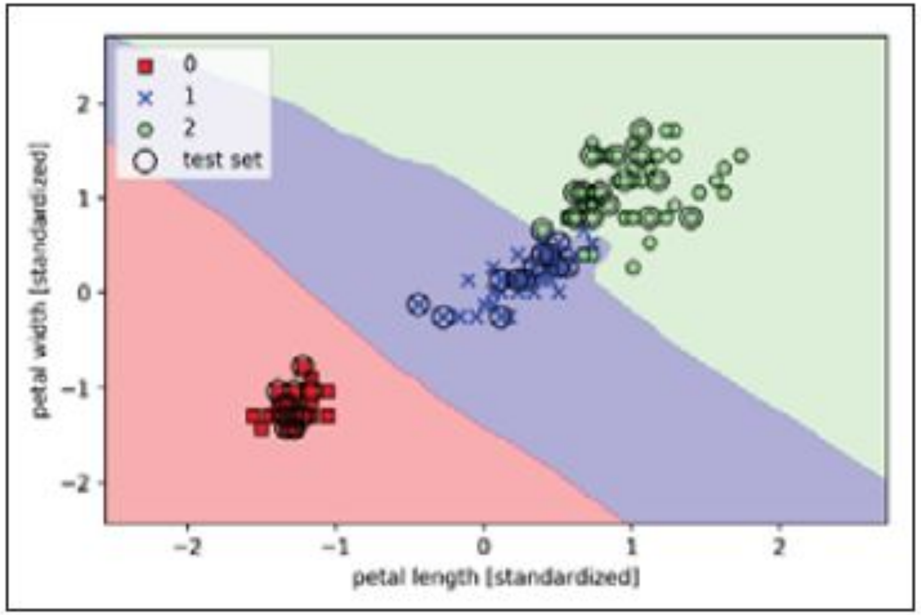
(a) Binary classification dataset. Note the class inbalance: the grey squares class has fewer members than the white circles.

(b) Average cross-validation error as K is increased.

FIGURE 5.11 Using cross-validation to find the best value of K. Ten-fold cross-validation was used and the reported error is averaged both over the folds and over 100 different partitions of the data into folds.

Example of KNN

```
>>> from sklearn.neighbors import KNeighborsClassifier
>>> knn = KNeighborsClassifier(n_neighbors=5, p=2,
...                            metric='minkowski')
>>> knn.fit(X_train_std, y_train)
>>> plot_decision_regions(X_combined_std, y_combined,
...                       classifier=knn, test_idx=range(105,150))
>>> plt.xlabel('petal length [standardized]')
>>> plt.ylabel('petal width [standardized]')
>>> plt.legend(loc='upper left')
>>> plt.show()
```



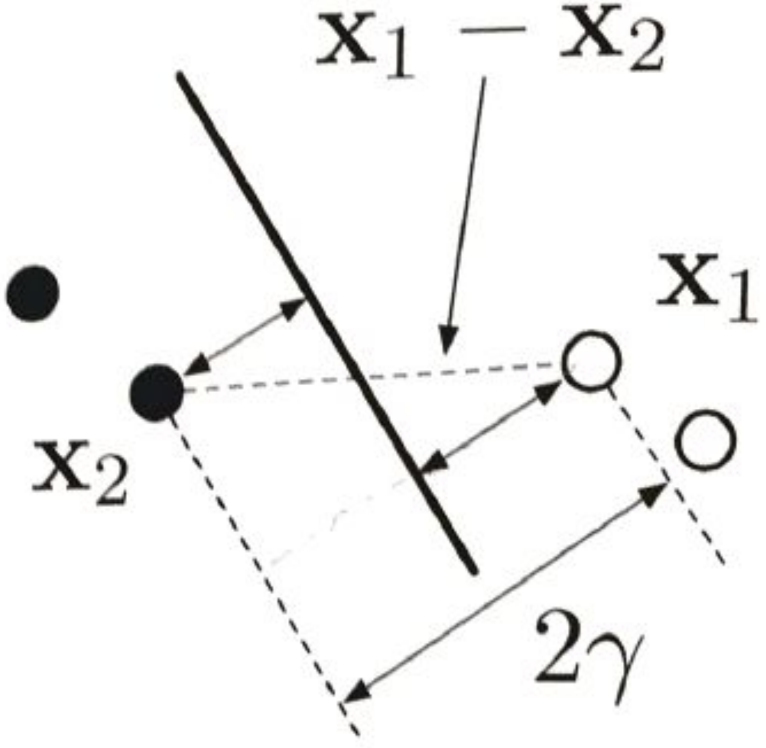
Support Vector Machines (SVM)

- Binary classification
- Dataset points can be separated with a Hyperplane defined by

  $\mathbf{w}^T \mathbf{x} + b$
- Given the hyperplane, y_n can be estimated with the equation

  $y_n = \text{sign}(\mathbf{w}^T \mathbf{x}_n + b)$
- There are likely many Hyperplanes (i.e. **w** and b) which can separate the dataset points
- The best hyperplane maximize the *margin,* the perpendicular distance to the closest points on either side

SVM Margin



SVM Margin (cont.)

- The margin can be calculated as

  $$2\Upsilon = \frac{1}{\|\boldsymbol{w}\|}\ \boldsymbol{w}^T\ (\boldsymbol{x}_1\ - \boldsymbol{x}_2)$$

  where $\|\boldsymbol{w}\| = \sqrt{\boldsymbol{w}^T\boldsymbol{w}}$

- If we apply the constraint $\mathbf{w}^{\mathrm{T}}\ \mathbf{x} + b = \pm 1$, we will arrive at the following formulas for margin

  $$\Upsilon = \frac{1}{\|\boldsymbol{w}\|}$$

Optimization Problem to Find the Hyperplane

$$\underset{\mathbf{w}}{\operatorname{argmin}} \quad \frac{1}{2}||\mathbf{w}||^2$$

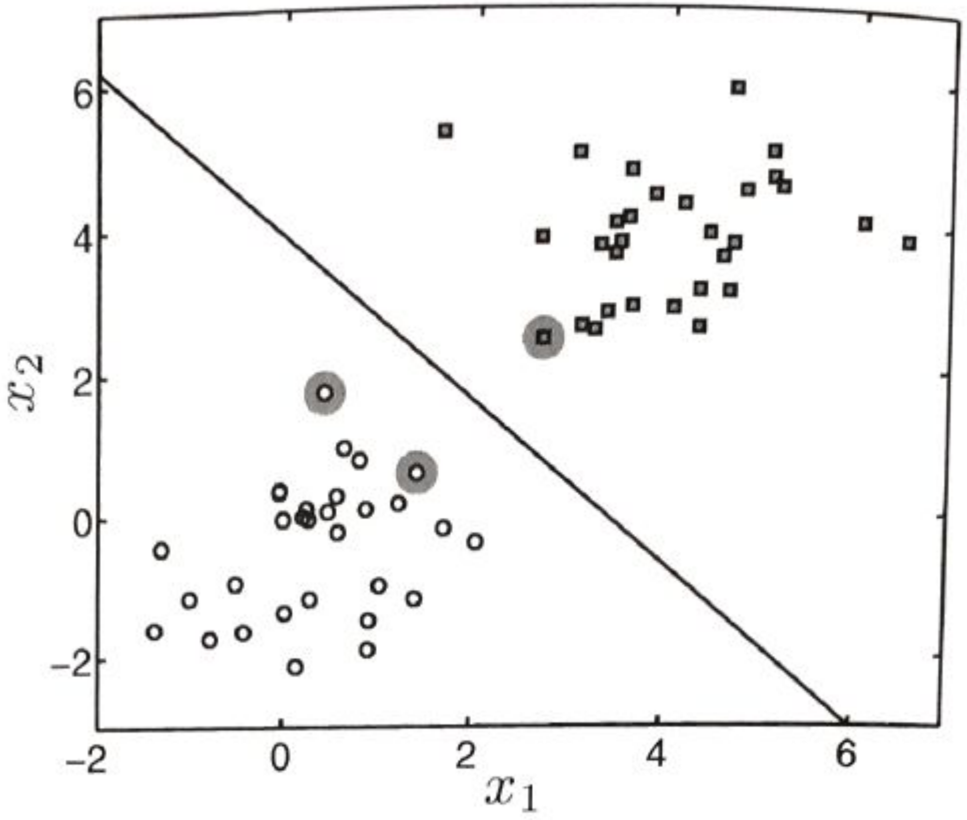
$$\text{subject to} \quad t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) \geq 1, \text{ for all } n.$$

Converted with Lagrange multipliers,
α_n which satisfy the constraint:
$\mathbf{w} = \sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n$

$$\text{argmax} \quad \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n,m=1}^{N} \alpha_m \alpha_n t_m t_n \mathbf{x}_m^{\mathrm{T}} \mathbf{x}_n,$$

$$\text{subject to} \quad \alpha_n \geq 0, \ \sum_{n=1}^{N} \alpha_n t_n = 0,$$

Support Vectors



Support vectors lies on hyperplanes:

$\mathbf{w}^T \mathbf{x} + b = \pm 1$
where $\mathbf{w} = \sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n$

SVM Prediction

- Find the parameters **w** and b satisfying the following constraints

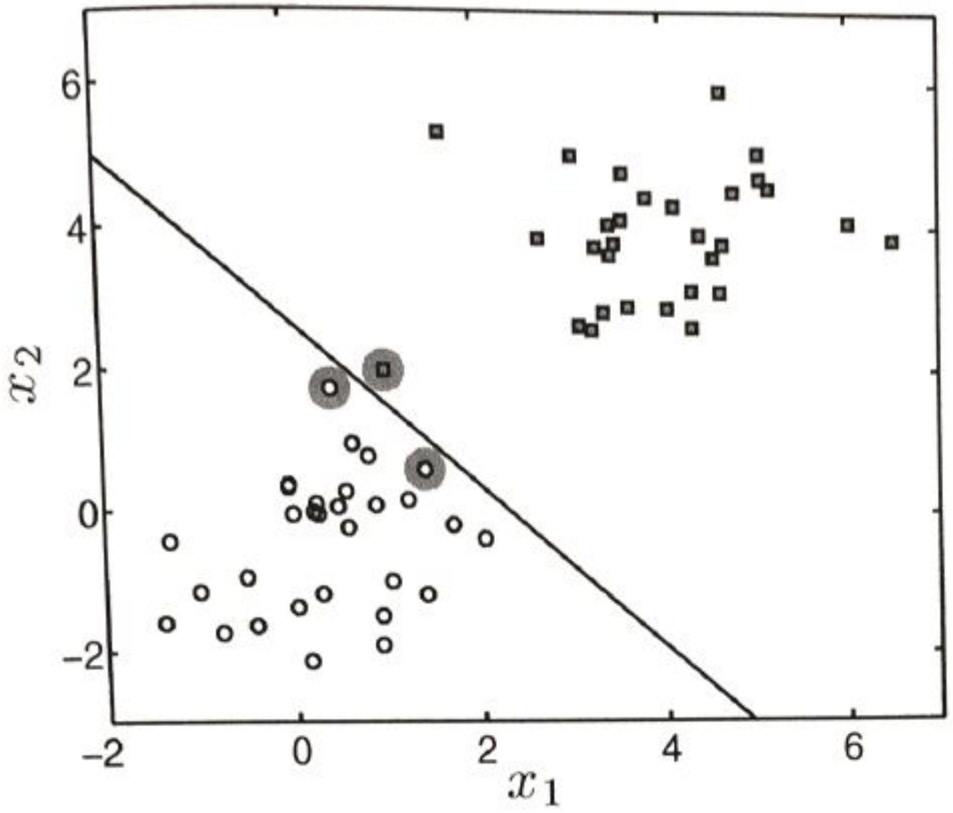
$$\mathbf{w} = \sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n$$
$$\mathbf{w}^T \mathbf{x} + b = \pm 1$$

- Given $\mathbf{x}_{new}$, calculate t_{new} with the equation

$$T_{new} = \mathbf{w}^T \mathbf{x}_{new} + b$$

Soft Margin

- Soft Margin is introduced to avoid the overfitting



Optimization Problem for Soft Margin

$$\underset{\mathbf{w}}{\operatorname{argmin}} \quad \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} + C\sum_{n=1}^{N}\xi_n$$

subject to $\xi_n \geq 0$ and $t_n(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + b) \geq 1 - \xi_n$ for all n.

Converted with Lagrange multipliers

$$\underset{\alpha}{\operatorname{argmax}} \quad \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{n,m=1}^{N}\alpha_n\alpha_m t_n t_m \mathbf{x}_n^{\mathsf{T}}\mathbf{x}_m$$

subject to $\sum_{n=1}^{N}\alpha_n t_n = 0$ and $0 \leq \alpha_n \leq C$, for all n.

Hyper-parameter: C

- C controls the penalty for the misclassification errors

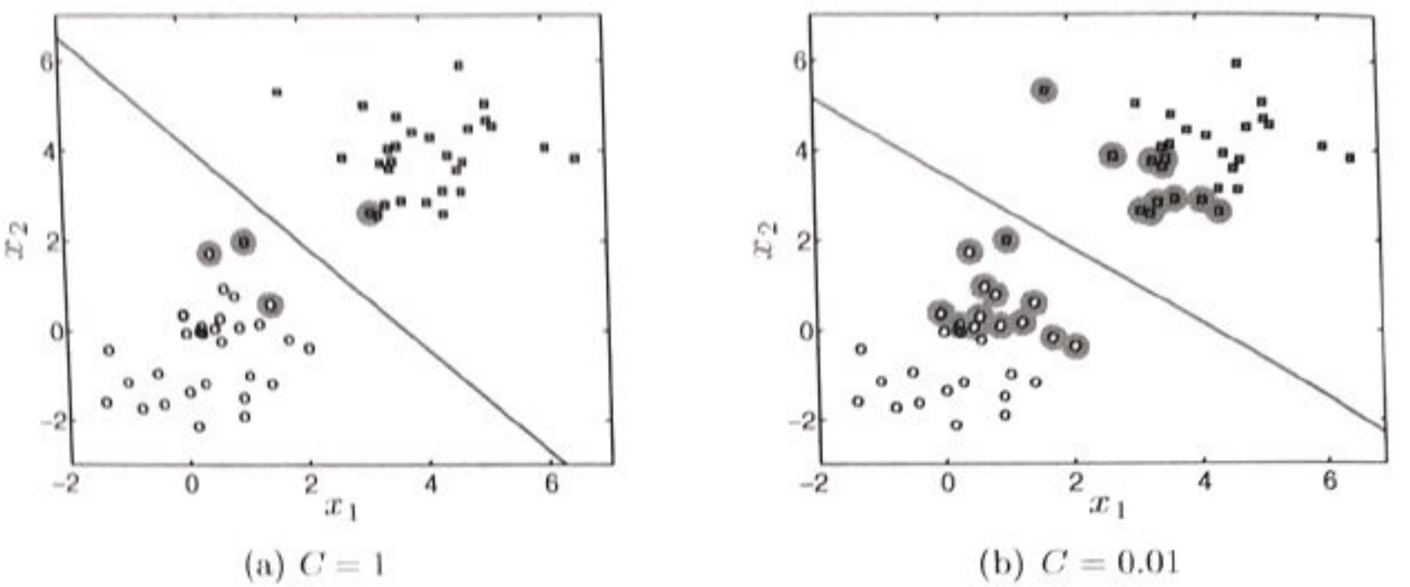
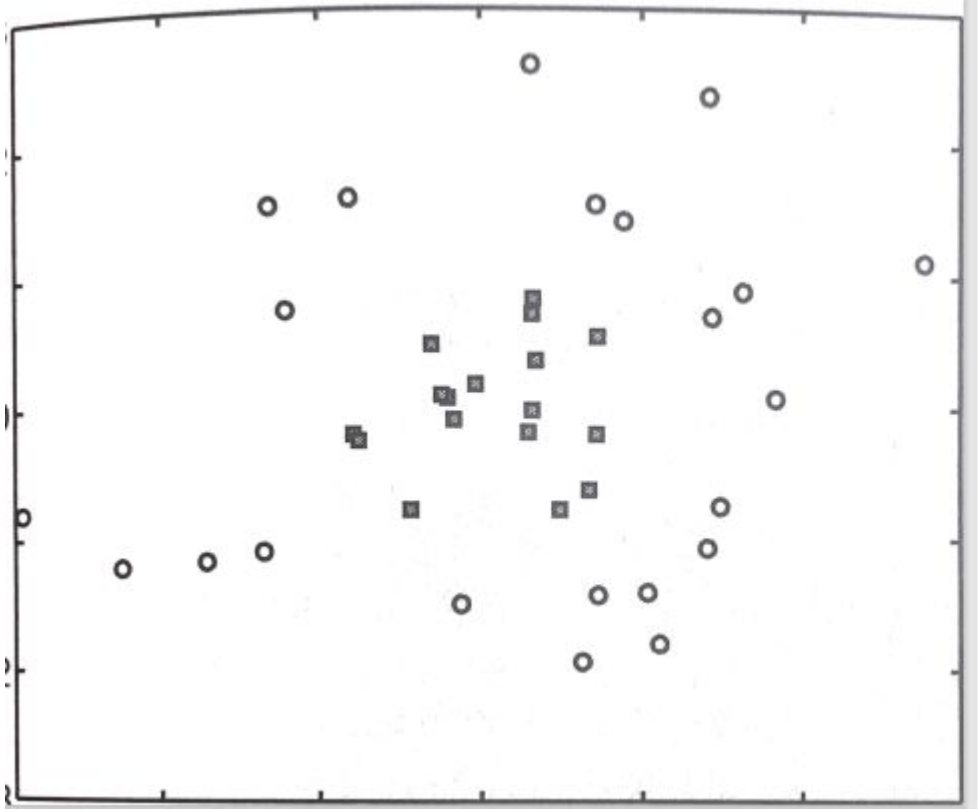


FIGURE 5.16 Decision boundary and support vectors for a linear SVM with a soft margin for two values of the margin parameter C. The influence of the stray support vector has been reduced.

Nonlinearly Separable Dataset



Hyper-parameter: Kernels

$$\underset{\boldsymbol{\alpha}}{\text{argmax}} \quad \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n,m=1}^{N} \alpha_n \alpha_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m)$$

$$\text{subject to} \quad \sum_{n=1}^{N} \alpha_n t_n = 0 \ \text{ and } \ 0 \le \alpha_n \le C, \ \text{ for all } n.$$

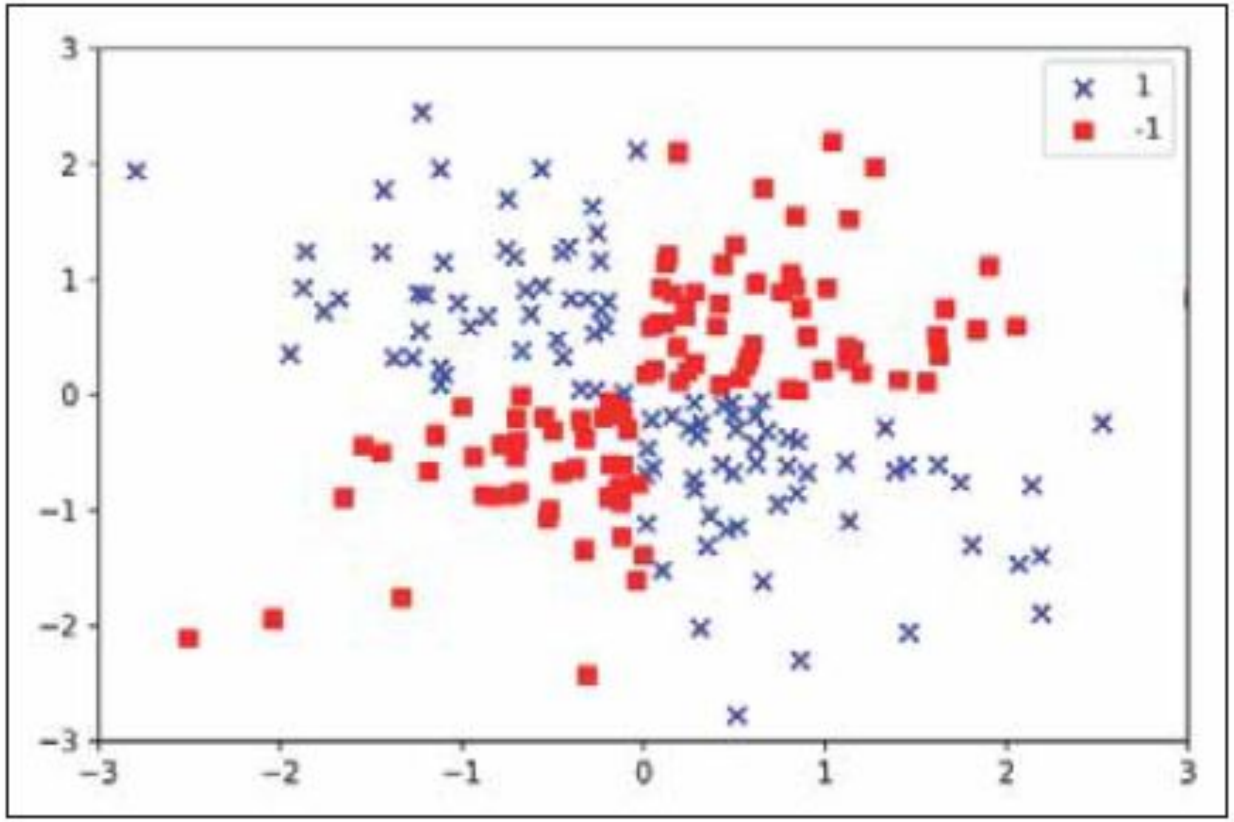
$$t_{\text{new}} = \text{sign}\left(\sum_{n=1}^{N} \alpha_n t_n k(\mathbf{x}_n, \mathbf{x}_{\text{new}}) + b \right).$$

Off-the-shelf Kernel Functions

$$\begin{aligned}
\text{linear} \quad k(\mathbf{x}_n, \mathbf{x}_m) &= \mathbf{x}_n^{\mathsf{T}} \mathbf{x}_m. \\
\text{Gaussian} \quad k(\mathbf{x}_n, \mathbf{x}_m) &= \exp\left\{ -\gamma (\mathbf{x}_n - \mathbf{x}_m)^{\mathsf{T}} (\mathbf{x}_n - \mathbf{x}_m) \right\}. \\
\text{polynomial} \quad k(\mathbf{x}_n, \mathbf{x}_m) &= (1 + \mathbf{x}_n^{\mathsf{T}} \mathbf{x}_m)^{\gamma}.
\end{aligned}$$

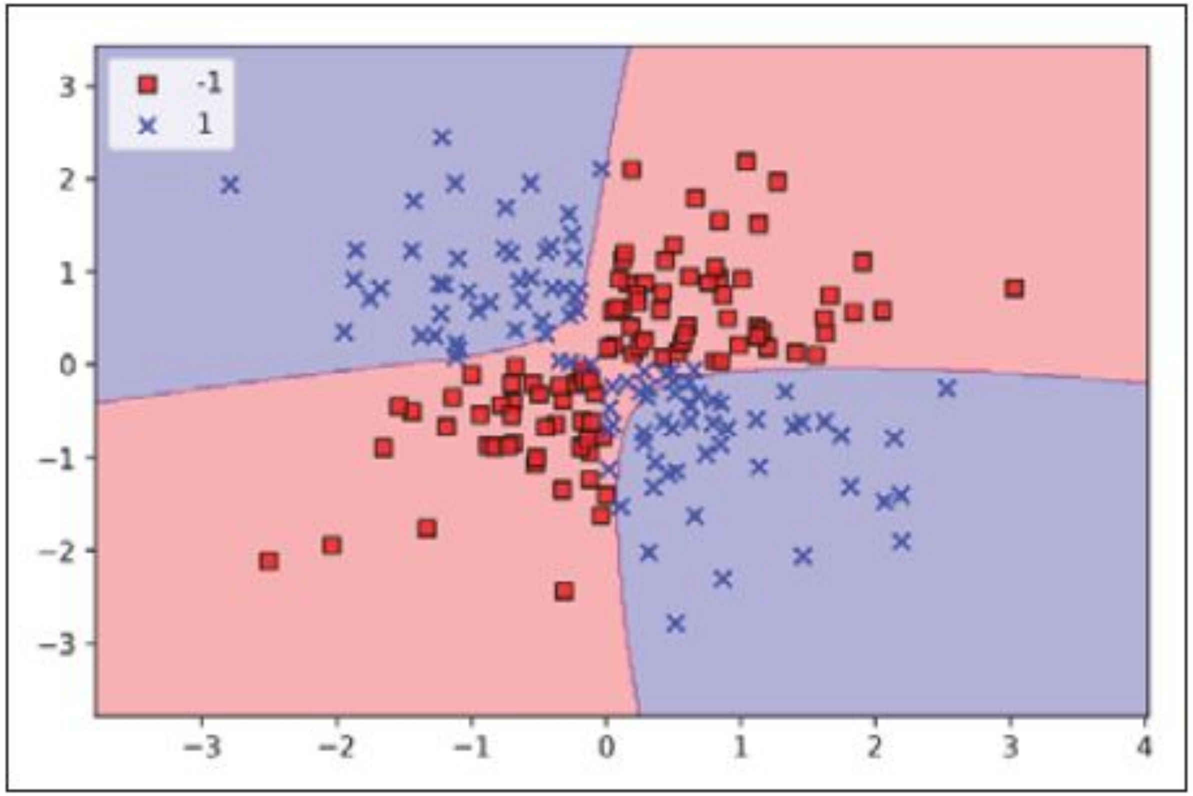
XOR Dataset

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> np.random.seed(1)
>>> X_xor = np.random.randn(200, 2)
>>> y_xor = np.logical_xor(X_xor[:, 0] > 0,
...                        X_xor[:, 1] > 0)
>>> y_xor = np.where(y_xor, 1, -1)
>>> plt.scatter(X_xor[y_xor == 1, 0],
...             X_xor[y_xor == 1, 1],
...             c='b', marker='x',
...             label='1')
>>> plt.scatter(X_xor[y_xor == -1, 0],
...             X_xor[y_xor == -1, 1],
...             c='r',
...             marker='s',
...             label='-1')
>>> plt.xlim([-3, 3])
>>> plt.ylim([-3, 3])
>>> plt.legend(loc='best')
>>> plt.show()
```



3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3
1
-1

```
>>> svm = SVC(kernel='rbf', random_state=1, gamma=0.10, C=10.0)
>>> svm.fit(X_xor, y_xor)
>>> plot_decision_regions(X_xor, y_xor, classifier=svm)
>>> plt.legend(loc='upper left')
>>> plt.show()
```



Impact of SVM Hyper-parameters

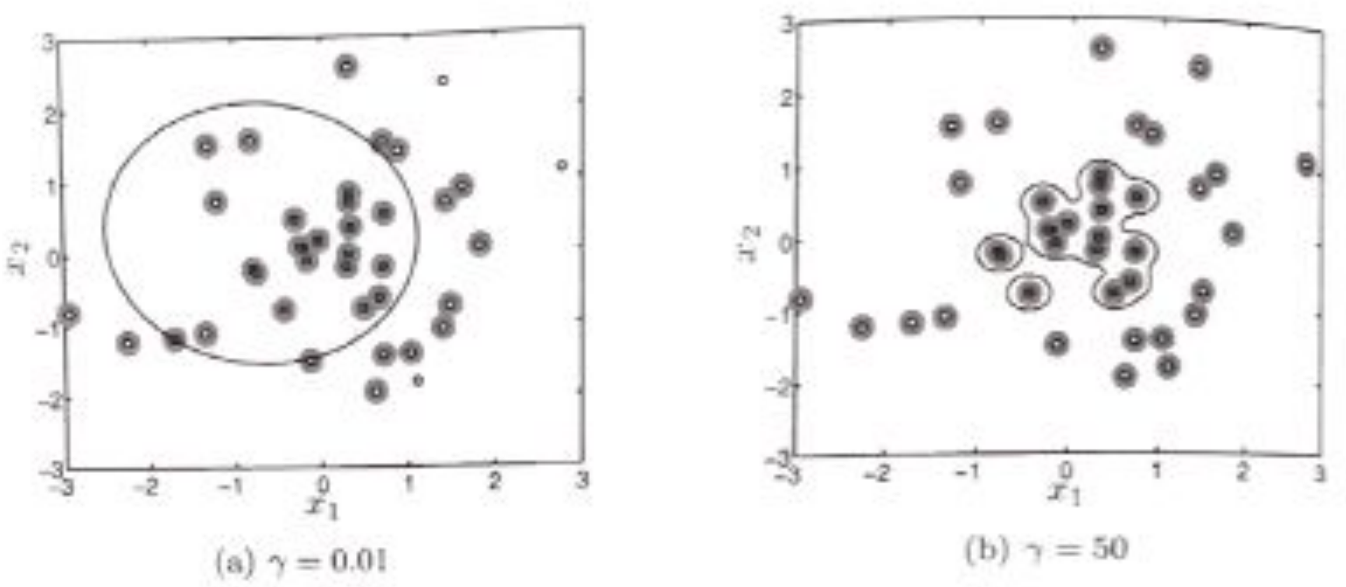
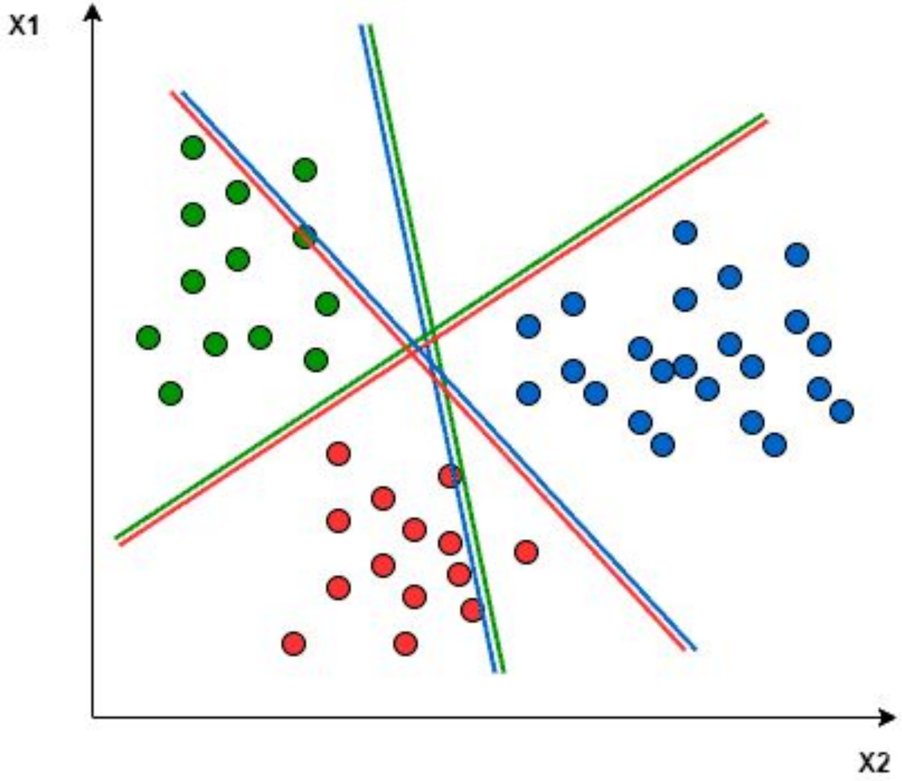


FIGURE 5.19 Decision boundary and support vectors for the dataset in Figure 5.17 using a Gaussian kernel with different values of the kernel parameter γ and $C = 10$.

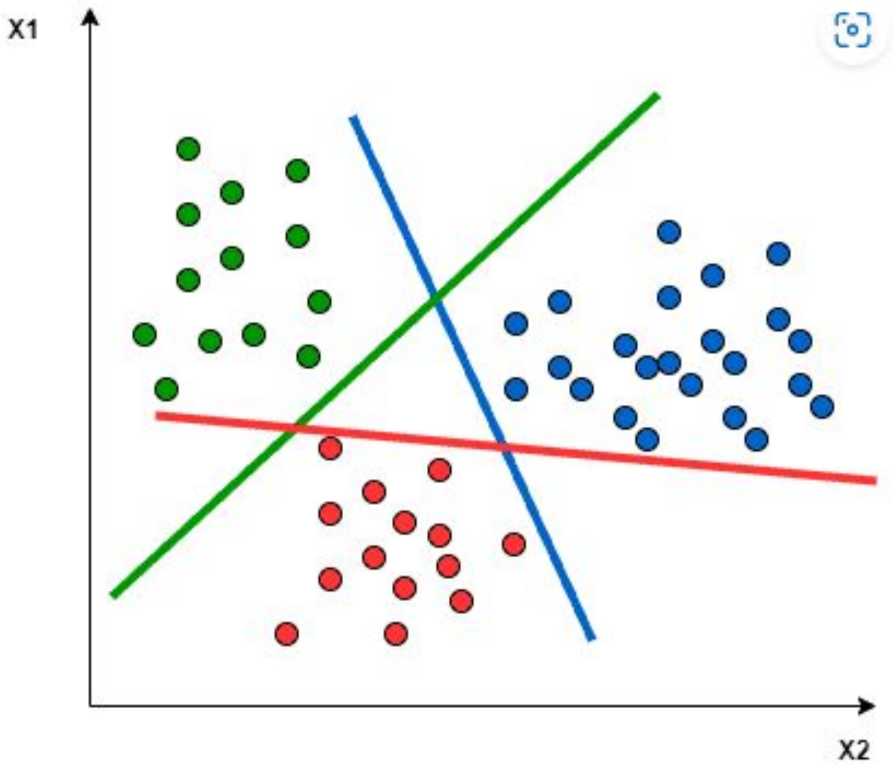
Multiclass SVM Classification

- One-to-One approach



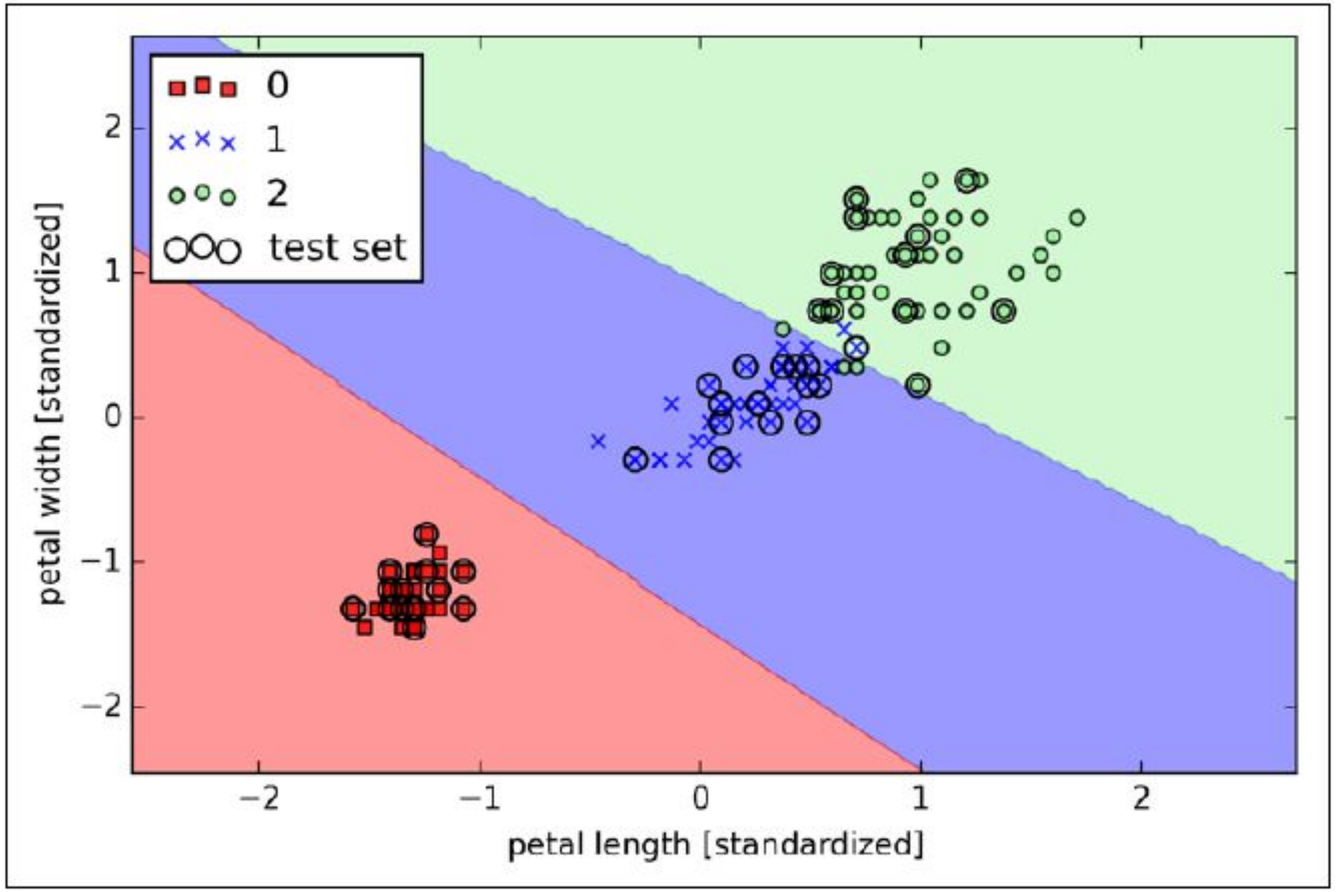
Multiclass SVM Classification (cont.)

- One-to-Rest approach

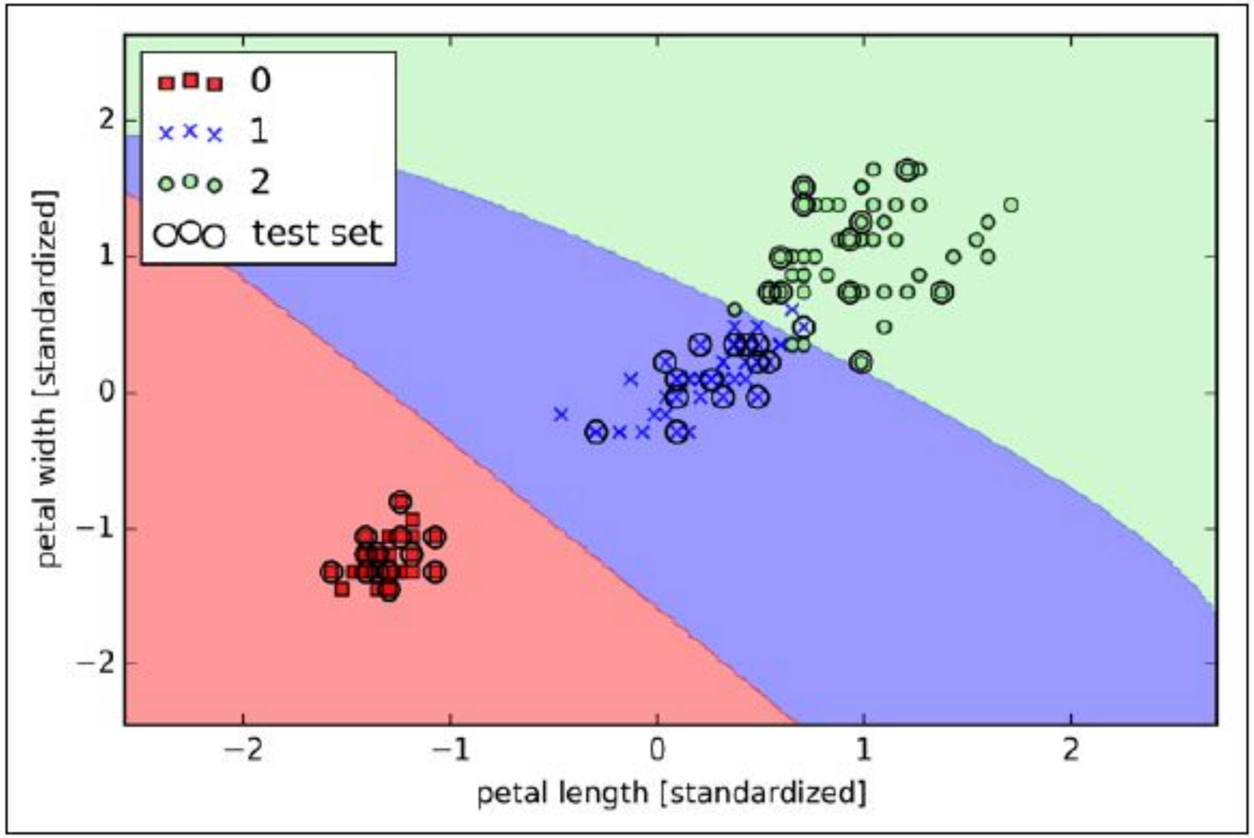


Python SVM (with C)

```
>>> from sklearn.svm import SVC
   >>> svm = SVC(kernel='linear', C=1.0, random_state=0)
>>> svm.fit(X_train_std, y_train)
>>> plot_decision_regions(X_combined_std,
...                       y_combined, classifier=svm,
...                       test_idx=range(105,150))
>>> plt.xlabel('petal length [standardized]')
>>> plt.ylabel('petal width [standardized]')
>>> plt.legend(loc='upper left')
>>> plt.show()
```

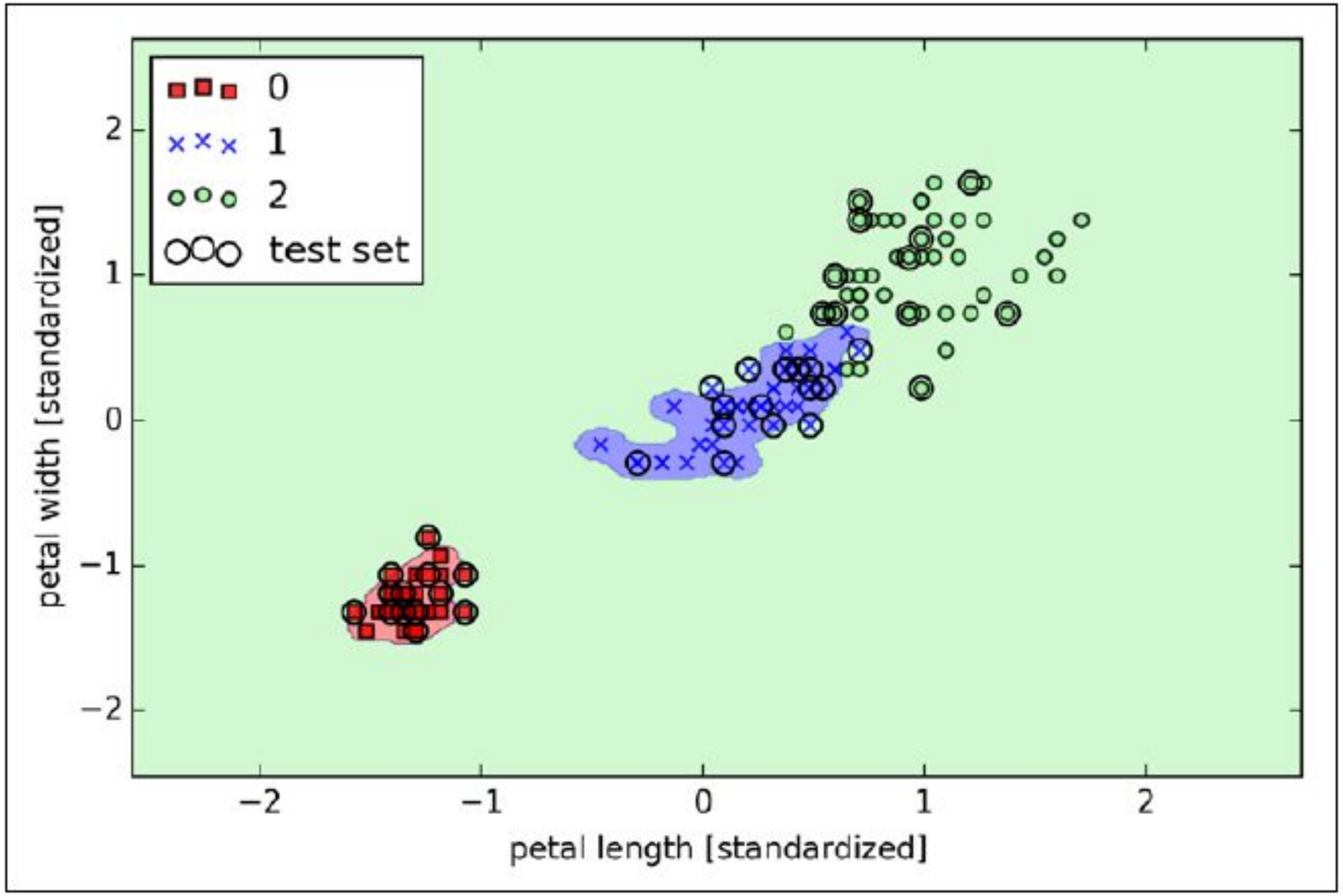

0
1
2
test set
petal width [standardized]
petal length [standardized]
−2
−1
0
1
2

```
>>> svm = SVC(kernel='rbf', random_state=0, gamma=0.2, C=1.0)
>>> svm.fit(X_train_std, y_train)
>>> plot_decision_regions(X_combined_std,
...                       y_combined, classifier=svm,
...                       test_idx=range(105,150))
>>> plt.xlabel('petal length [standardized]')
>>> plt.ylabel('petal width [standardized]')
>>> plt.legend(loc='upper left')
>>> plt.show()
```


0
1
2
test set
2
1
0
−1
−2
−2
−1
0
1
2
petal width [standardized]
petal length [standardized]

Python Code with Both Hyper-parameters

```
>>> svm = SVC(kernel='rbf', random_state=0, gamma=100.0, C=1.0)
>>> svm.fit(X_train_std, y_train)
>>> plot_decision_regions(X_combined_std,
...                       y_combined, classifier=svm,
...                       test_idx=range(105,150))
>>> plt.xlabel('petal length [standardized]')
>>> plt.ylabel('petal width [standardized]')
>>> plt.legend(loc='upper left')
>>> plt.show()
```


0
1
2
test set
petal width [standardized]
petal length [standardized]
2
1
0
−1
−2
−2
−1
0
1
2

Comparison between SVM and KNN Algorithm

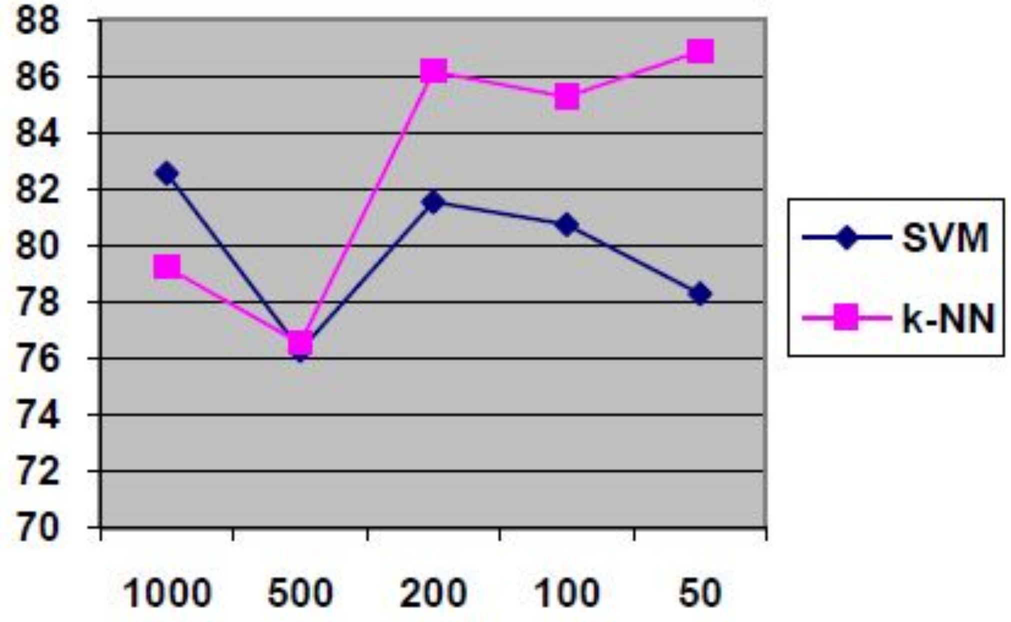


Figure 2.Graph showing the accuracy of both systems