Лабораторная работа №10

Метод опорных векторов

Следующий тип обучения с учителем, который мы обсудим, – это метод опорных векторов. Мы рассматривали использование линейного метода опорных векторов для задач классификации в разделе «Линейные модели для классификации». Ядерный метод опорных векторов (часто его просто называют SVM) – это расширение метода опорных векторов, оно позволяет получать более сложные модели, которые не сводятся к построению простых гиперплоскостей в пространстве. Несмотря на то что метод опорных векторов можно применять для задач классификации регрессии, мы ограничимся классификацией, реализованной в SVC. Аналогичные принципы применяются в опорных векторах для регрессии реализованы в SVR.

Линейные модели и нелинейные признаки

В низкоразмерных пространствах линейные модели накладывают весьма жесткие ограничения, поскольку линии и гиперплоскости имеют ограниченную гибкость. Один из способов сделать линейную модель более гибкой – добавить новые признаки, например, добавить взаимодействия или полиномы входных признаков.

Давайте взглянем на синтетический набор данных, который мы использовали в разделе «Важность признаков в деревьях».

```
X, y = make_blobs(centers=4, random_state=8)
y = y % 2
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
```

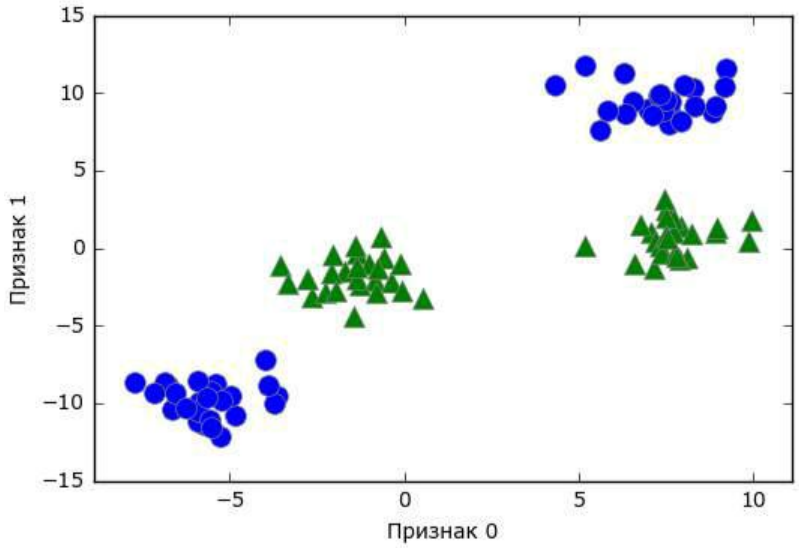

Рис. 10.1 Набор данных с двух классовой классификацией, в котором классы линейно неразделимы

Линейная модель классификации может отделить точки только с помощью прямой линии и не может дать хорошее качество для этого набора данных (см. рис. 10.2):

```
from sklearn.svm import LinearSVC
linear_svm = LinearSVC().fit(X, y)

mglearn.plots.plot_2d_separator(linear_svm, X)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
```

Теперь давайте расширим набор входных признаков, скажем, добавим качестве нового признака feature1 ** 2, квадрат второго признака. Теперь каждую точку данных мы представим не в виде точки двумерного пространства (feature0, feature1), а виде точки трехмерного пространства (feature0, feature1, feature1 ** 2).18 Новое пространство признаков показано на рис. 10.3 в виде трехмерной диаграммы рассеяния:

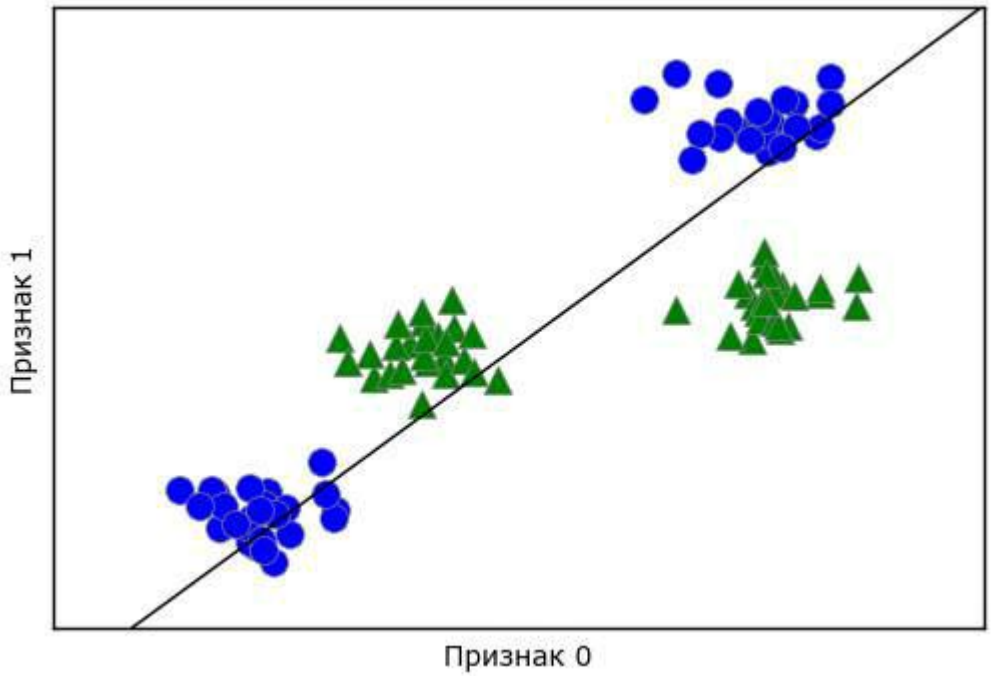

Рис. 10.2 Граница принятия решений, найденная с помощью линейного SVM

```
# добавляем второй признак, возведенный в квадрат
X_new = np.hstack([X, X[:, 1:] ** 2])

from mpl_toolkits.mplot3d import Axes3D, axes3d
figure = plt.figure()
# визуализируем в 3D
ax = Axes3D(figure, elev=-152, azim=-26)
# сначала размещаем на графике все точки с y == 0, затем с y == 1 mask = y == 0
```

```
ax.scatter(
    X_new[mask, 0],
    X_new[mask, 1],
    X_new[mask, 2],
    c='b',
    cmap=mglearn.cm2,
    s=60
)
ax.scatter(
    X_new[~mask, 0],
    X_new[~mask, 1],
    X_new[~mask, 2],
    c='r',
    marker='^',
    cmap=mglearn.cm2,
    s=60
)
ax.set_xlabel("признак0")
ax.set_ylabel("признак1")
ax.set_zlabel("признак1 ** 2")
```

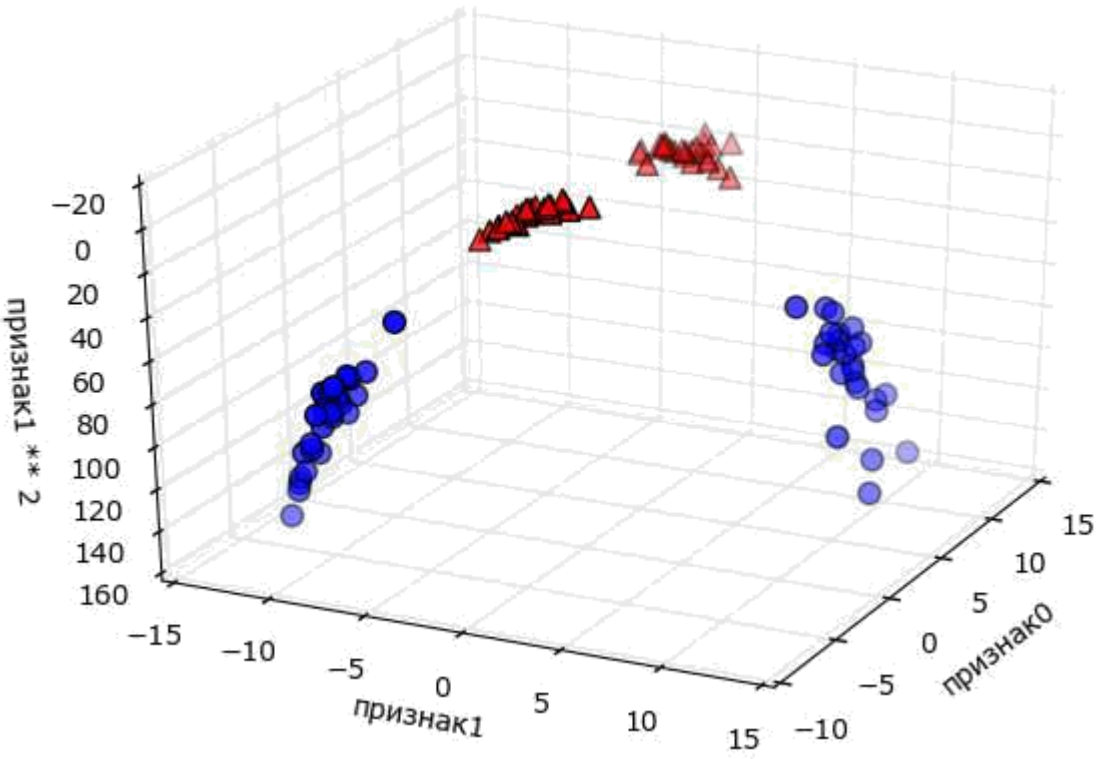

Рис. 10.3 Расширение набора данных, показанного на рис. 10.2, за счет добавления третьего признака, полученного на основе признака 1

В новом представлении данных уже можно отделить два класса друг от друга, используя линейную модель, плоскость в трехмерном пространстве. Мы можем убедиться в этом, подогнав линейную модель к дополненным данным (см. рис. 10.4):

```
linear_svm_3d = LinearSVC().fit(X_new, y)
coef, intercept = linear_svm_3d.coef_.ravel(), linear_svm_3d.intercept_

# показать границу принятия решений линейной модели
figure = plt.figure()
ax = Axes3D(figure, elev=-152, azim=-26)

xx = np.linspace(
    X_new[:, 0].min() - 2,
    X_new[:, 0].max() + 2, 50
)
yy = np.linspace(
    X_new[:, 1].min() - 2,
    X_new[:, 1].max() + 2, 50
)

XX, YY = np.meshgrid(xx, yy)
= (coef[0] * XX + coef[1] * YY + intercept) / -coef[2]
ax.plot_surface(XX, YY, ZZ, rstride=8, cstride=8, alpha=0.3)
ax.scatter(
    X_new[mask, 0],
    X_new[mask, 1],
    X_new[mask, 2],
    c='b',
    cmap=mglearn.cm2,
     s=60
)
ax.scatter(
    X_new[~mask, 0],
    X_new[~mask, 1],
    X_new[~mask, 2],
    c='r',
    marker='^',
    cmap=mglearn.cm2,
     s=60
)

ax.set_xlabel("признак0")
ax.set_ylabel("признак1")
ax.set_zlabel("признак1 ** 2")
```

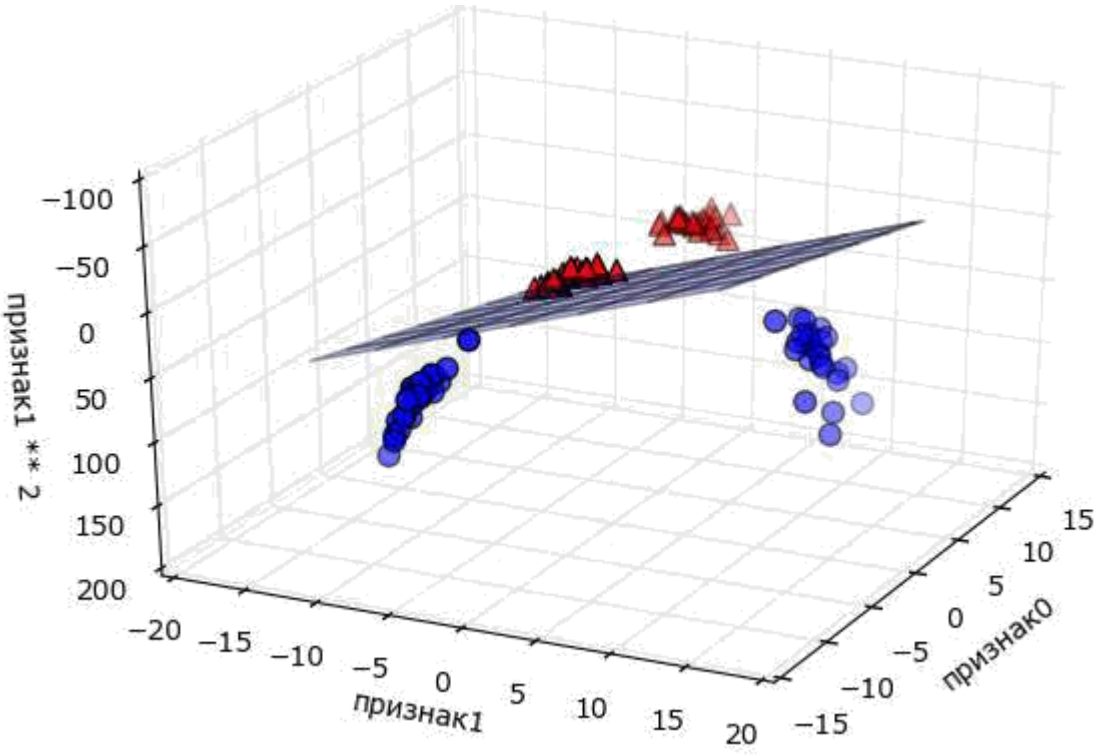

Рис. 10.4 Граница принятия решений, найденная линейным SVM для расширенного трехмерного набора данных

Фактически модель линейного SVM как функция исходных признаков не является больше линейной. Это скорее эллипс, как можно увидеть на графике, построенном ниже (рис. 10.5):

```
ZZ = YY**2
dec = linear_svm_3d.decision_function(
    np.c_[
        XX.ravel(),
        YY.ravel(),
        ZZ.ravel()
    ]
)
plt.contourf(XX, YY, dec.reshape(XX.shape), levels=[dec.min(), 0, dec.max()],
    cmap=mglearn.cm2, alpha=0.5)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.show()
```

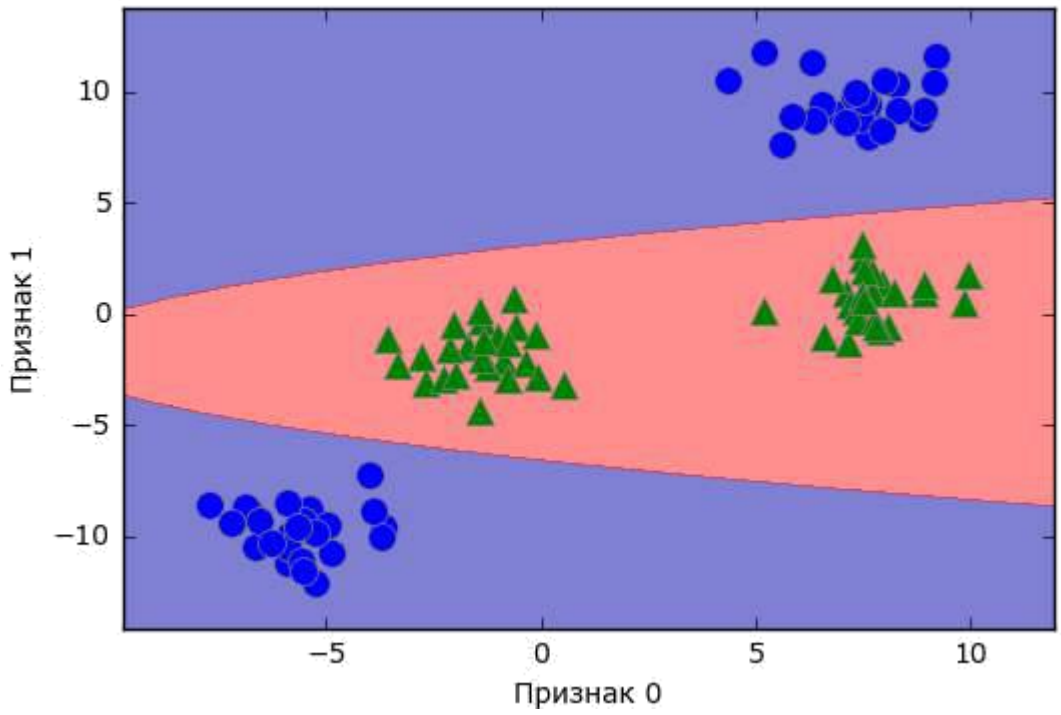

Рис. 10.5 Граница принятия решений для рис. 10.4 как функция от двух исходных признаков

Ядерный трюк (kernel trick)

Из вышесказанного можно сделать вывод, что добавление нелинейных признаков может улучшить прогнозную силу линейной модели. Однако часто мы не знаем, какие признаки необходимо добавить, и добавление большего числа признаков (например, рассмотрение всех возможных взаимодействий в 100-мерном пространстве признаков) может очень сильно увеличить стоимость вычислений. К счастью, есть хитрый математический трюк, который позволяет нам обучить классификатор в многомерном пространстве, фактически не прибегая к вычислению нового, возможно, очень высокоразмерного пространства. Этот трюк известен под названием «ядерный трюк» (**kernel trick**) и он непосредственно вычисляет евклидовы расстояния (более точно, скалярные произведения точек данных), чтобы получить расширенное пространство признаков без фактического добавления новых признаков.

Существуют два способа поместить данные в высокоразмерное пространство, которые чаще всего используются методом опорных векторов: полиномиальное ядро, которое вычисляет все возможные полиномиальные комбинации исходных признаков до определенной степени, и ядро RBF (радиальная базисная функция), также известное как гауссовское ядро. Гауссовское ядро немного сложнее объяснить, поскольку оно соответствует бесконечному пространству признаков. Объяснить гауссовское ядро можно так: оно рассматривает все возможные полиномы всех степеней, однако важность признаков снижается с возрастанием степени. Хотя на практике математические детали ядерного SVM не столь важны и можно легко понять, каким образом SVM с помощью ядра RBF делает прогнозы, мы рассмотрим их в следующем разделе.

Понимание принципов SVM

В ходе обучения SVM вычисляет важность каждой точки обучающих данных с точки зрения определения решающей границы между двумя классами. Обычно лишь часть точек обучающего набора важна для определения границы принятия решений: точки, которые лежат на границе между классами. Они называются опорными векторами (**support vectors**) и дали свое название машине опорных векторов.

Чтобы получить прогноз для новой точки, измеряется расстояние до каждого опорного вектора. Классификационное решение принимается, исходя из расстояний до опорных векторов, а также важности опорных векторов, полученных в процессе обучения (хранятся в атрибуте **dual_coef_** класса SVC).

Расстояние между точками данных измеряется с помощью гауссовского ядра:

$$k_{rbf}(x_1, x_2) \ \exp\left(|| \ x_1 \ x_2||^2 \right)$$

Здесь x_1 и x_2 – точки данных, $|| \ x_1 \ x_2||^2$ – означает евклидово расстояние, а (гамма) – параметр, который регулирует ширину гауссовского ядра.

Рис. 10.7 показывает результат обучения машины опорных векторов на двумерном 2-ух классовом наборе данных. Граница принятия решений показана черным цветом, а опорные векторы – это точки большего размера, обведенные широким контуром. Программный код строит этот график, обучая SVM на наборе данных forge:

```
from sklearn.svm import SVC
X, y = mglearn.tools.make_handcrafted_dataset()
svm = SVC(kernel='rbf', C=10, gamma=0.1).fit(X, y)
mglearn.plots.plot_2d_separator(svm, X, eps=.5)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)

# размещаем на графике опорные векторы sv = svm.support_vectors_
# метки классов опорных векторов определяются знаком дуальных коэффициентов

sv_labels = svm.dual_coef_.ravel() > 0

mglearn.discrete_scatter(sv[:, 0], sv[:, 1], sv_labels, s=15, markeredgewidth=3
)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
```

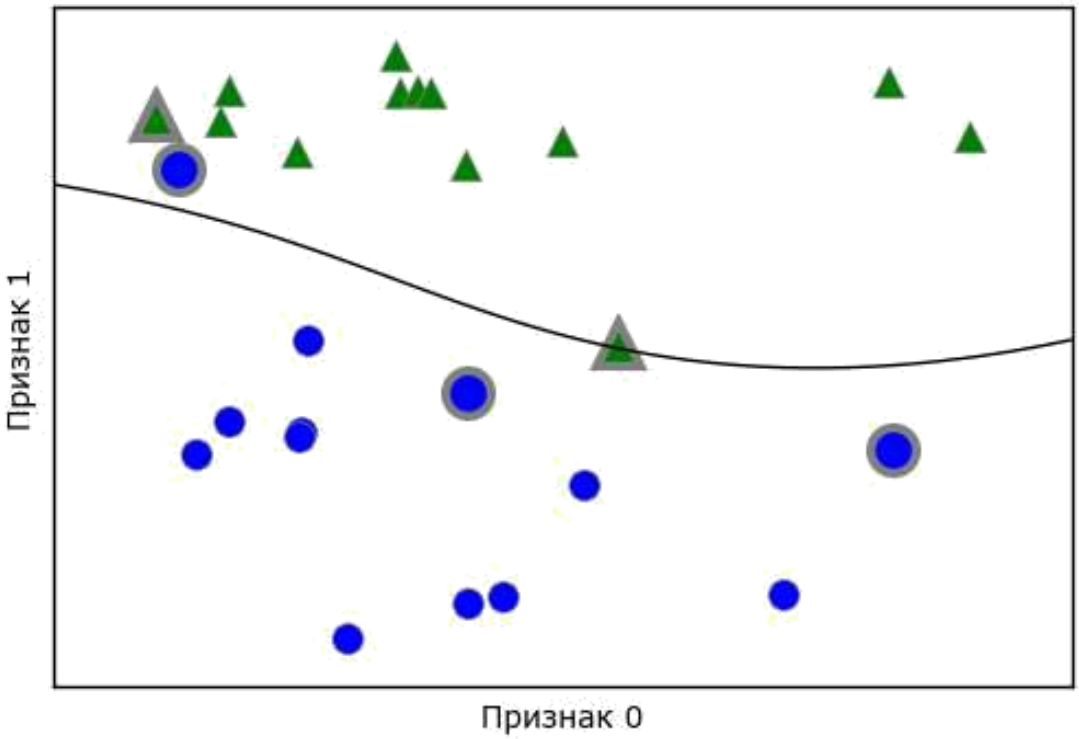

Рис. 10.6 Граница принятия решений для рис. 10.4 как функция от двух исходных признаков

В данном случае SVM дает очень гладкую и нелинейную (непрямую) границу. Здесь мы скорректировали параметр **C** и параметр **gamma**, которые сейчас подробно обсудим.

Настройка параметров SVM

Параметр **gamma** – это параметр формулы, приведенной в предыдущем разделе. Он регулирует ширину гауссовского ядра. Параметр gamma задает степень близости расположения точек. Параметр **C** представляет собой параметр регуляризации, аналогичный тому, что использовался в линейных моделях. Он ограничивает важность каждой точки (точнее, ее **dual_coef_**).

Давайте посмотрим, что происходит при изменении этих параметров (рис. 10.8):

```
fig, axes = plt.subplots(3, 3, figsize=(15, 10))
for ax, C in zip(axes, [-1, 0, 3]):
for a, gamma in zip(ax, range(-1, 2)):
mglearn.plots.plot_svm(log_C=C, log_gamma=gamma, ax=a)
axes[0, 0].legend(
    ["class 0", "class 1",  "sv class 0", "sv class 1"],
    ncol=4,
    loc=(.9, 1.2)
)
```

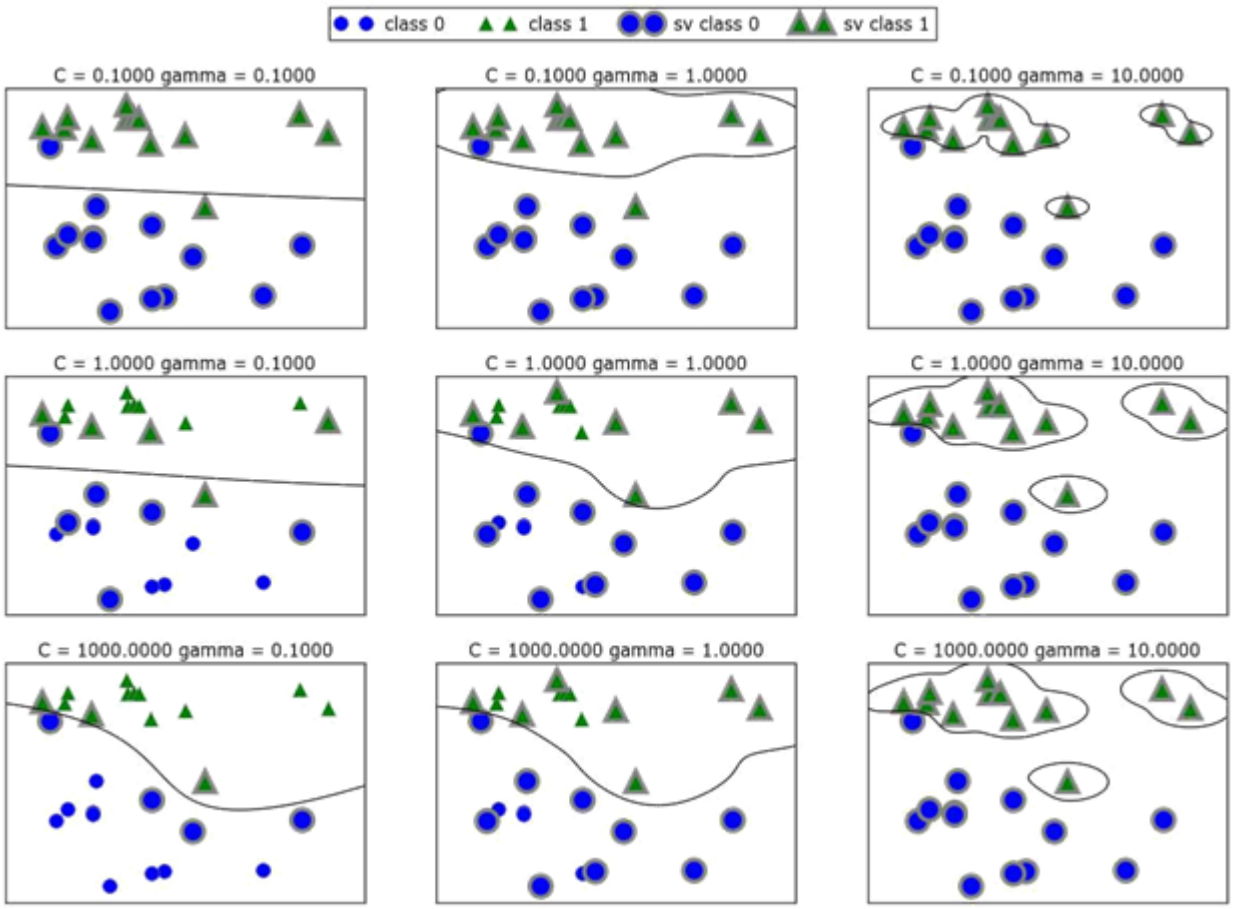

Рис. 10.7 Границы принятия решений и опорные векторы для различных значений параметров C и gamma

Перемещаясь слева направо, мы увеличиваем значение параметра gamma с 0.1 до 10. Небольшое значение gamma соответствует большому радиусу гауссовского ядра, это означает, что многие точки рассматриваются как расположенные поблизости. Это приводит к получению очень гладких границ принятия решений, показанных в левой части графика, а границы, которые больше фокусируются на отдельных точках, расположились в правой части графика. Низкое значение **gamma** означает медленное изменение решающей границы, которое дает модель низкой сложности, в то время как высокое значение gamma дает более сложную модель.

Перемещаясь сверху вниз, мы увеличиваем параметр **C** с 0.1 до 1000. Как и в случае с линейными моделями, небольшое значение **C** соответствует модели с весьма жесткими ограничениями, в которой каждая точка данных может иметь лишь очень ограниченное влияние. В левом верхнем углу рис. можно увидеть, что граница принятия решений выглядит как почти линейная, неправильно классифицированные точки почти не влияют на линию. Увеличение значения **C**, как показано в левом нижнем углу, позволяет этим точкам оказывать более сильное влияние на модель и делает решающую границу изогнутой, позволяя правильно классифицировать данные точки.

Давайте применим SVM с RBF-ядром к набору данных Breast Cancer. По умолчанию используются **C**=1 и **gamma**=1/**n_features**:

```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data,
    cancer.target,
    random_state=0
)
svc = SVC()
svc.fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.2f}".format(
    svc.score(X_train, y_train)
))
print("Правильность на тестовом наборе: {:.2f}".format(
    svc.score(X_test, y_test)
))
```

```
Правильность на обучающем наборе: 1.00
Правильность на тестовом наборе: 0.63
```

Переобучение модели весьма существенно: идеальная правильность на обучающем наборе и лишь 63%-ная правильность на тестовом наборе. Хотя SVM часто дает хорошее качество модели, он очень чувствителен к настройкам параметров и масштабированию данных. В частности, SVM требует, чтобы все признаки были измерены в одном и том же масштабе. Давайте посмотрим на минимальное и максимальное значения каждого признака в log-пространстве (рис. 10.8):

```
plt.plot(X_train.min(axis=0), 'o', label="min")
plt.plot(X_train.max(axis=0), '^', label="max")
plt.legend(loc=4)
plt.xlabel("Индекс признака")
plt.ylabel("Величина признака")
plt.yscale("log")
```

Исходя из этого графика, мы можем заключить, что признаки в наборе данных Breast Cancer имеют совершенно различные порядки величин. Для ряда моделей (например, для линейных моделей) данный факт может быть в некоторой степени проблемой, однако для ядерного SVM он будет иметь разрушительные последствия. Давайте рассмотрим некоторые способы решения этой проблемы.

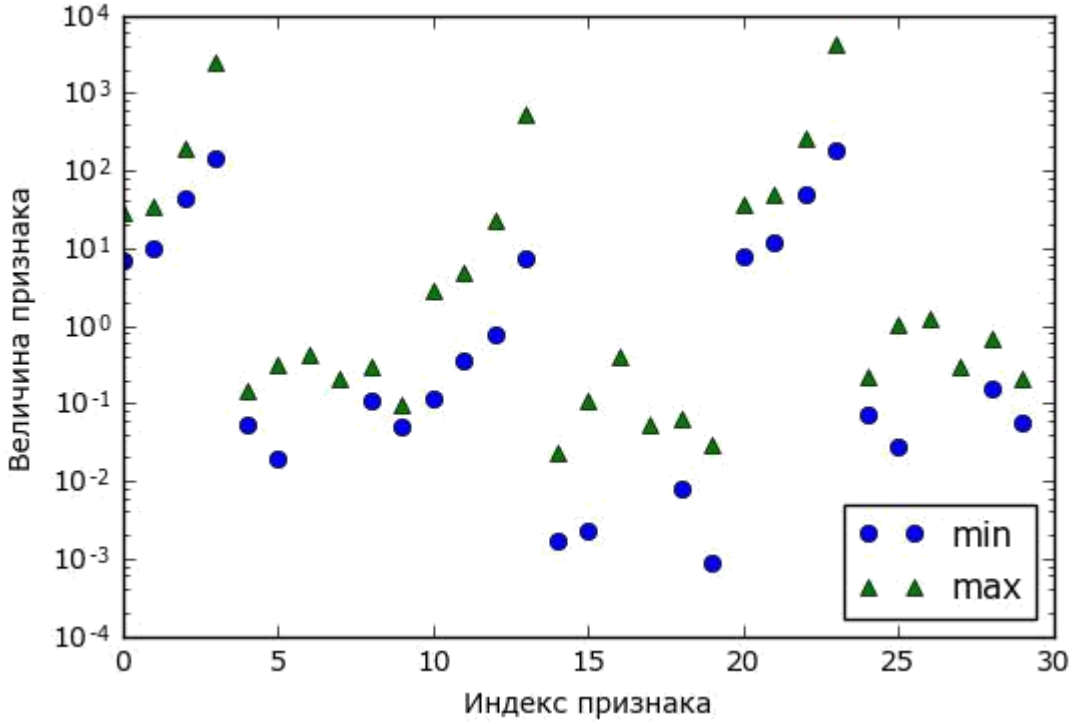

Рис. 10.8 Диапазоны значений признаков для набора данных Breast Cancer (обратите внимание, что ось y имеет логарифмическую шкалу)

Предварительная обработка данных для SVM

Один из способов решения этой проблемы – масштабирование всех признаков таким образом, чтобы все они имели примерно один и тот же масштаб. Общераспространенный метод масштабирования для ядерного SVM заключается в масштабировании данных так, чтобы все признаки принимали значения от 0 до 1. Мы увидим, как это делается с помощью метода предварительной обработки MinMaxScaler в главе 3, в которой дадим более подробную информацию. А сейчас давайте сделаем это «вручную»:

```
# вычисляем минимальное значение для каждого признака обучающего набора
min_on_training = X_train.min(axis=0)
# вычисляем ширину диапазона для каждого признака (max - min) обучающего набора
range_on_training = (X_train - min_on_training).max(axis=0)
# вычитаем минимальное значение и затем делим на ширину диапазона
# min=0 и max=1 для каждого признака
X_train_scaled = (X_train - min_on_training) / range_on_training
print("Минимальное значение для каждого признака\n{}"
    .format(X_train_scaled.min(axis=0))
)
print("Максимальное значение для каждого признака\n {}"
    .format(X_train_scaled.max(axis=0))
)
```

```
Минимальное значение для каждого признака
[0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
Максимальное значение для каждого признака
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

```
# используем ТО ЖЕ САМОЕ преобразование для тестового набора,
# используя минимум и ширину диапазона из обучающего набора
X_test_scaled = (X_test - min_on_training) / range_on_training

svc = SVC()
svc.fit(X_train_scaled, y_train)

print("Правильность на обучающем наборе: {:.3f}"
    .format( svc.score(X_train_scaled, y_train))
)
print("Правильность на тестовом наборе: {:.3f}"
    .format(svc.score(X_test_scaled, y_test))
)
```

```
Правильность на обучающем наборе: 0.948
Правильность на тестовом наборе: 0.951
```

Масштабирование данных привело к огромной разнице! На самом деле наша модель имеет признаки недообучения, когда качество модели на обучающем и тестовом наборе весьма схоже, но все еще далеко от 100%-ной правильности. Исходя из этого, мы можем попытаться увеличить C или gamma, чтобы подогнать более сложную модель. Например:

```
svc = SVC(C=1000)
svc.fit(X_train_scaled, y_train)

print("Правильность на обучающем наборе: {:.3f}"
    .format( svc.score(X_train_scaled, y_train))
)
print("Правильность на тестовом наборе: {:.3f}"
    .format(svc.score(X_test_scaled, y_test))
)
```

```
Правильность на обучающем наборе: 0.988
Правильность на тестовом наборе: 0.972
```

Параметры, недостатки и параметры

Ядерный метод опорных векторов – это модели, обладающие мощной прогнозной силой и хорошо работающие на различных наборах данных. SVM позволяет строить сложные решающие границы, даже если данные содержат лишь несколько признаков. Они хорошо работают на низкоразмерных и высокоразмерных данных (то есть когда у нас мало или, наоборот, много признаков), однако плохо масштабируются с ростом объема данных. Запуск **SVM** на наборе данных объемом 10000 наблюдений не составляет проблем, однако работа с наборами данных объемом 100000 наблюдений и больше может стать сложной задачей с точки зрения времени вычислений и использования памяти.

Другим недостатком является то, что **SVM** требует тщательной предварительной обработки данных и настройки параметров. Именно поэтому сейчас многие специалисты в различных сферах вместо SVM используют модели на основе дерева, например, случайные леса или градиентный бустинг (который практически не требуют предварительную обработки данных). Кроме того, модели **SVM** трудноисследуемы, тяжело понять, почему был сделан именно такой прогноз и довольно сложно объяснить модель неспециалисту.

Однако все же стоит попробовать **SVM**, особенно в тех случаях, когда все ваши признаки имеют одинаковые единицы измерения (например, все признаки являются интенсивностями пикселей) и измерены в одном и том же масштабе.

Важными параметрами ядерного **SVM** являются параметр регуляризации **C**, тип ядра, а также параметры, определяемые ядром. Хотя мы в основном сосредоточились на ядре **RBF**, в scikit-learn доступны и другие типы ядер. Ядро **RBF** имеет лишь один параметр **gamma**, который является обратной величиной ширины гауссовского ядра. **gamma** и **C** регулируют сложность модели, более высокие значения этих параметров дают более сложную модель. Таким образом, оптимальные настройки обоих параметров, как правило, сильно взаимосвязаны между собой и поэтому **C** и **gamma** должны быть отрегулированы вместе.

Код к лабораторной работе:

```
import sklearn
import mglearn
import matplotlib.pyplot as plt
import numpy as np

from sklearn.datasets import make_blobs
X, y = make_blobs(centers=4, random_state=8)
y = y % 2

mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.show()

from sklearn.svm import LinearSVC
linear_svm = LinearSVC().fit(X, y)

mglearn.plots.plot_2d_separator(linear_svm, X)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.show()

X_new = np.hstack([X, X[:, 1:] ** 2])
from mpl_toolkits.mplot3d import Axes3D, axes3d
figure = plt.figure()
ax = Axes3D(figure, elev=-152, azim=-26)
mask = y == 0

ax.scatter(X_new[mask, 0], X_new[mask, 1], X_new[mask, 2], c='b', cmap=mglearn.cm2,
s=60)
ax.scatter(X_new[~mask, 0], X_new[~mask, 1], X_new[~mask, 2], c='r', marker='^',
cmap=mglearn.cm2, s=60)
ax.set_xlabel("признак0")
ax.set_ylabel("признак1")
ax.set_zlabel("признак1 ** 2")
plt.show()

linear_svm_3d = LinearSVC().fit(X_new, y)
coef, intercept = linear_svm_3d.coef_.ravel(), linear_svm_3d.intercept_
figure = plt.figure()

ax = Axes3D(figure, elev=-152, azim=-26)

xx = np.linspace(X_new[:, 0].min() - 2, X_new[:, 0].max() + 2, 50)
yy = np.linspace(X_new[:, 1].min() - 2, X_new[:, 1].max() + 2, 50)

XX, YY = np.meshgrid(xx, yy)
ZZ = (coef[0] * XX + coef[1] * YY + intercept) / -coef[2]
```

```
ax.plot_surface(XX, YY, ZZ, rstride=8, cstride=8, alpha=0.3)
ax.scatter(X_new[mask, 0], X_new[mask, 1], X_new[mask, 2], c='b', cmap=mglearn.cm2,
s=60)
ax.scatter(X_new[~mask, 0], X_new[~mask, 1], X_new[~mask, 2], c='r', marker='^',
cmap=mglearn.cm2, s=60)

ax.set_xlabel("признак0")
ax.set_ylabel("признак1")
ax.set_zlabel("признак1 ** 2")
plt.show()

ZZ = YY**2
dec = linear_svm_3d.decision_function(np.c_[XX.ravel(), YY.ravel(), ZZ.ravel()])
plt.contourf(XX, YY, dec.reshape(XX.shape), levels=[dec.min(), 0, dec.max()],
    cmap=mglearn.cm2, alpha=0.5)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.show()

from sklearn.svm import SVC
X, y = mglearn.tools.make_handcrafted_dataset()
svm = SVC(kernel='rbf', C=10, gamma=0.1).fit(X, y)
mglearn.plots.plot_2d_separator(svm, X, eps=.5)
mglearn.discrete_scatter(X[:, 0], X[:, 1], y)
sv = svm.support_vectors_
sv_labels = svm.dual_coef_.ravel() > 0

mglearn.discrete_scatter(sv[:, 0], sv[:, 1], sv_labels, s=15, markeredgewidth=3)
plt.xlabel("Признак 0")
plt.ylabel("Признак 1")
plt.show()


fig, axes = plt.subplots(3, 3, figsize=(15, 10))

for ax, C in zip(axes, [-1, 0, 3]):
    for a, gamma in zip(ax, range(-1, 2)):
        mglearn.plots.plot_svm(log_C=C, log_gamma=gamma, ax=a)

axes[0, 0].legend(["class 0", "class 1", "sv class 0", "sv class 1"], ncol=4,
loc=(.9, 1.2))
plt.show()

X_train, X_test, y_train, y_test = train_test_split( cancer.data, cancer.target,
random_state=0)
svc = SVC()
svc.fit(X_train, y_train)
print("Правильность на обучающем наборе: {:.2f}".format(svc.score(X_train,
y_train)))
print("Правильность на тестовом наборе: {:.2f}".format(svc.score(X_test, y_test)))
```

```
plt.show()

plt.plot(X_train.min(axis=0), 'o', label="min")
plt.plot(X_train.max(axis=0), '^', label="max")
plt.legend(loc=4)
plt.xlabel("Индекс признака")
plt.ylabel("Величина признака")
plt.yscale("log")
plt.show()

min_on_training = X_train.min(axis=0)
range_on_training = (X_train - min_on_training).max(axis=0)
X_train_scaled = (X_train - min_on_training) / range_on_training
print("Минимальное значение для каждого
признака\n{}".format(X_train_scaled.min(axis=0)))
print("Максимальное значение для каждого признака\n
{}".format(X_train_scaled.max(axis=0)))

X_test_scaled = (X_test - min_on_training) / range_on_training
svc = SVC()
svc.fit(X_train_scaled, y_train)
print("Правильность на обучающем наборе: {:.3f}".format( svc.score(X_train_scaled,
y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(svc.score(X_test_scaled,
y_test)))

svc = SVC(C=1000)
svc.fit(X_train_scaled, y_train)
print("Правильность на обучающем наборе: {:.3f}".format( svc.score(X_train_scaled,
y_train)))
print("Правильность на тестовом наборе: {:.3f}".format(svc.score(X_test_scaled,
y_test)))
```