Exercise 1

支持向量机背后的基本思想是什么？

答案：

支持向量机背后的基本思想是在类之间拟合尽可能宽的“街道”。换句话说，目标是在分隔两个类的决策边界和训练实例之间具有最大可能的间隔。在执行软边际分类时，SVM 会在完美分离两个类别和拥有尽可能宽的街道（即少数实例可能最终出现在街道上）之间寻找折衷方案。另一个关键思想是在训练非线性数据集时使用内核。还可以调整支持向量机以执行线性和非线性回归，以及异常检测。

Exercise 2

什么是支持向量？

答案：

训练 SVM 后，支持向量是位于“街道”上的任何实例，包括其边界。决策边界完全由支持向量决定。任何不是支持向量的实例（即不在街上）都没有任何影响；您可以删除它们、添加更多实例或四处移动它们，只要它们远离街道，它们就不会影响决策边界。使用内核化 SVM 计算预测仅涉及支持向量，而不涉及整个训练集。

Exercise 3

为什么在使用支持向量机时缩放输入很重要？

答案：

SVM 尝试在类之间拟合最大可能的“街道”，因此如果训练集未按比例缩放，则 SVM 将倾向于忽略小特征。

Exercise 4

当一个 SVM 分类器在对一个实例进行分类时，它能输出一个置信度分数吗？概率如何？

答案：

您可以使用 **decision_function()** 方法来获取置信度分数。这些分数表示实例和决策边界之间的距离。但是，它们不能直接转换为类别概率的估计。如果在创建 SVC 时设置 **probability=True** ，那么在训练结束时它将使用5折交叉验证为训练样本生成样本外分数，并训练一个 **LogisticRegression** 模型来映射这些分数估计概率。**predict_proba()** 和 **predict_log_proba()** 方法将可用。

Exercise 5

如何在 LinearSVC 、SVC 和 SGDClassifier 之间进行选择？

答案：

所有三个类别都可用于大间距线性分类。

SVC 类还支持内核技巧，这使其能够处理非线性任务。然而，这是有代价的：SVC 类不能很好地扩展到具有许多实例的数据集。不过，它确实可以很好地扩展到大量特性。

LinearSVC 类实现了线性 SVM 的优化算法，而 SGDClassifier 使用随机梯度下降。根据数据集，LinearSVC 可能比 SGDClassifier 快一点，但并非总是如此，而且 SGDClassifier 更灵活，而且它支持增量学习。

Exercise 6

假设您已经用 RBF 内核训练了一个 SVM 分类器，但它似乎欠拟合训练集。你应该增加还是减少 γ（**gamma**）？那么 C 呢？

答案：

如果使用 RBF 内核训练的 SVM 分类器欠拟合训练集，则可能存在过多的正则化。要减少它，您需要增加 γ 或 C（或两者）。

Exercise 7

一个模型对 ϵ 不敏感意味着什么？

答案：

SVM 回归模型试图在其预测周围的小范围内拟合尽可能多的实例。如果在此边距内添加实例，则模型根本不会受到影响：它被认为是对 ϵ 不敏感的。

Exercise 8

使用内核技巧的意义是什么？

答案：

内核技巧是一种数学技术，可以训练非线性 SVM 模型。生成的模型相当于使用非线性变换将输入映射到另一个空间，然后在生成的高维输入上训练线性 SVM 。内核技巧给出了相同的结果，而根本不必转换输入。

Exercise 9

在一个线性可分的数据集上训练一个 **LinearSVC** 。然后在同一数据集上训练一个 SVC 和一个 SGDClassifier 。看看你是否不能让它们生产大致相同的模型。

答案：

让我们使用 Iris 数据集：Iris Setosa 和 Iris Versicolor 类是线性可分的。

In [2]:
```
from sklearn import datasets

iris = datasets.load_iris(as_frame=True)
X = iris.data[["petal length (cm)", "petal width (cm)"]].values
y = iris.target

setosa_or_versicolor = (y == 0) | (y == 1)
```

```
X = X[setosa_or_versicolor]

y = y[setosa_or_versicolor]
```

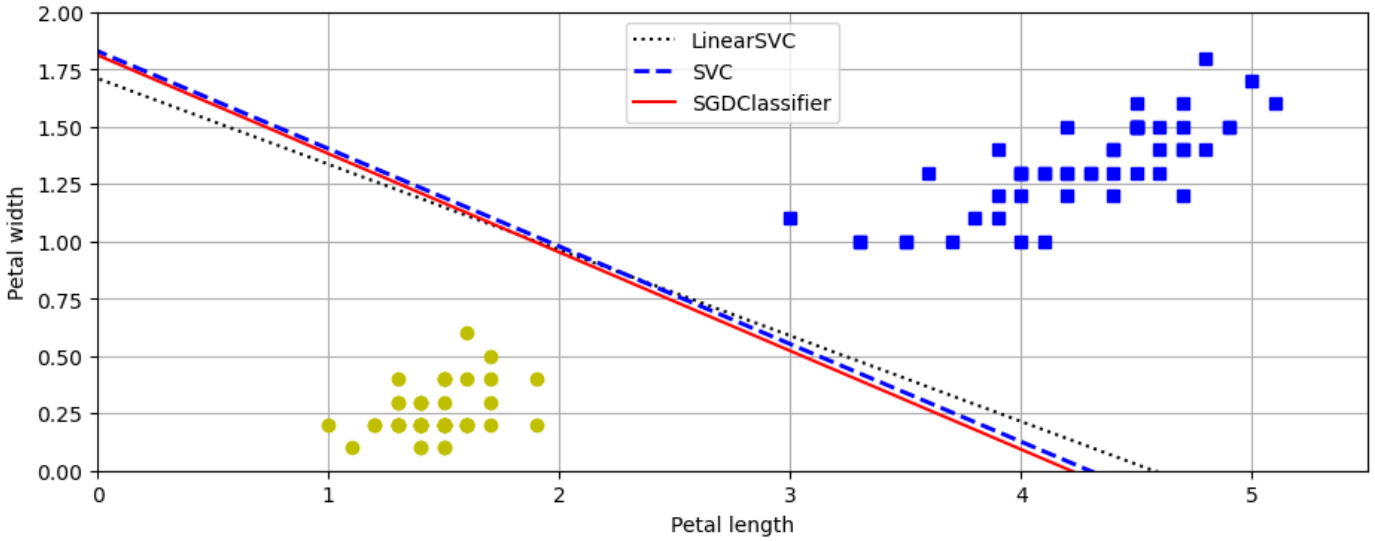
现在让我们构建和训练 3 个模型：

- 请记住，`LinearSVC` 默认使用 `loss="squared_hinge"`，因此如果我们希望所有 3 个模型产生相似的结果，我们需要设置 `loss="hinge"`。
- 此外，`SVC` 类默认使用 RBF 内核，因此我们需要设置 `kernel="linear"` 以获得与其他两个模型相似的结果。
- 最后，`SGDClassifier` 类没有 `C` 超参数，但它有另一个称为 `alpha` 的正则化超参数，因此我们可以调整它以获得与其他两个模型相似的结果。

In [3]:
```
from sklearn.svm import SVC, LinearSVC
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler

C = 5
alpha = 0.05

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

lin_clf = LinearSVC(loss="hinge", C=C, random_state=42).fit(X_scaled, y)

svc_clf = SVC(kernel="linear", C=C).fit(X_scaled, y)

sgd_clf = SGDClassifier(alpha=alpha, random_state=42).fit(X_scaled, y)
```

让我们绘制这三个模型的决策边界：

In [5]:
```
import matplotlib.pyplot as plt

def compute_decision_boundary(model):
    w = -model.coef_[0, 0] / model.coef_[0, 1]
    b = -model.intercept_[0] / model.coef_[0, 1]
    return scaler.inverse_transform([[-10, -10 * w + b], [10, 10 * w + b]])

lin_line = compute_decision_boundary(lin_clf)
svc_line = compute_decision_boundary(svc_clf)
sgd_line = compute_decision_boundary(sgd_clf)

# Plot all three decision boundaries
plt.figure(figsize=(11, 4))
plt.plot(lin_line[:, 0], lin_line[:, 1], "k:", label="LinearSVC")
plt.plot(svc_line[:, 0], svc_line[:, 1], "b--", linewidth=2, label="SVC")
plt.plot(sgd_line[:, 0], sgd_line[:, 1], "r-", label="SGDClassifier")
plt.plot(X[:, 0][y==1], X[:, 1][y==1], "bs") # label="Iris versicolor"
plt.plot(X[:, 0][y==0], X[:, 1][y==0], "yo") # label="Iris setosa"
plt.xlabel("Petal length")
plt.ylabel("Petal width")
plt.legend(loc="upper center")
plt.axis([0, 5.5, 0, 2])
plt.grid()

plt.show()
```



Exercise 10

在葡萄酒数据集上训练一个 SVM 分类器，您可以使用 **sklearn.datasets.load_wine()** 加载它。该数据集包含了由 3 个不同的培养者生产的 178 个葡萄酒样品的化学分析：目标是训练一个能够基于葡萄酒的化学分析来预测培养者的分类模型。由于 SVM 分类器是二值分类器，因此需要使用 OVA 来分类所有三个类。你能达到什么准确度？

答案：

首先，让我们获取数据集，查看其描述，然后将其拆分为训练集和测试集：

In [8]:
```
from sklearn.datasets import load_wine

wine = load_wine(as_frame=True)
```

In [9]:
```
print(wine.DESCR)
```

```
.. _wine_dataset:

Wine recognition dataset
------------------------

**Data Set Characteristics:**

    :Number of Instances: 178
    :Number of Attributes: 13 numeric, predictive attributes and the class
    :Attribute Information:
                - Alcohol
                - Malic acid
                - Ash
                - Alcalinity of ash
                - Magnesium
                - Total phenols
                - Flavanoids
                - Nonflavanoid phenols
                - Proanthocyanins
                - Color intensity
                - Hue
                - OD280/OD315 of diluted wines
                - Proline

    - class:
            - class_0
```

- class_1
- class_2

:Summary Statistics:

	Min	Max	Mean	SD
Alcohol:	11.0	14.8	13.0	0.8
Malic Acid:	0.74	5.80	2.34	1.12
Ash:	1.36	3.23	2.36	0.27
Alcalinity of Ash:	10.6	30.0	19.5	3.3
Magnesium:	70.0	162.0	99.7	14.3
Total Phenols:	0.98	3.88	2.29	0.63
Flavanoids:	0.34	5.08	2.03	1.00
Nonflavanoid Phenols:	0.13	0.66	0.36	0.12
Proanthocyanins:	0.41	3.58	1.59	0.57
Colour Intensity:	1.3	13.0	5.1	2.3
Hue:	0.48	1.71	0.96	0.23
OD280/OD315 of diluted wines:	1.27	4.00	2.61	0.71
Proline:	278	1680	746	315

:Missing Attribute Values: None
:Class Distribution: class_0 (59), class_1 (71), class_2 (48)
:Creator: R.A. Fisher
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
:Date: July, 1988

This is a copy of UCI ML Wine recognition datasets.
https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data

The data is the results of a chemical analysis of wines grown in the same region in Italy by three different cultivators. There are thirteen different measurements taken for different constituents found in the three types of wine.

Original Owners:

Forina, M. et al, PARVUS -
An Extendible Package for Data Exploration, Classification and Correlation.
Institute of Pharmaceutical and Food Analysis and Technologies,
Via Brigata Salerno, 16147 Genoa, Italy.

Citation:

Lichman, M. (2013). UCI Machine Learning Repository
[https://archive.ics.uci.edu/ml]. Irvine, CA: University of California,
School of Information and Computer Science.

.. topic:: References

(1) S. Aeberhard, D. Coomans and O. de Vel,
Comparison of Classifiers in High Dimensional Settings,
Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of
Mathematics and Statistics, James Cook University of North Queensland.
(Also submitted to Technometrics).

The data was used with many others for comparing various
classifiers. The classes are separable, though only RDA
has achieved 100% correct classification.
(RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
(All results using the leave-one-out technique)

(2) S. Aeberhard, D. Coomans and O. de Vel,
"THE CLASSIFICATION PERFORMANCE OF RDA"

```
  Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Journal of Chemometrics).
```

In [10]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    wine.data, wine.target, random_state=42)
```

In [11]:
```
X_train.head()
```

Out[11]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	pro
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	
100	12.08	2.08	1.70	17.5	97.0	2.23	2.17	0.26	
122	12.42	4.43	2.73	26.5	102.0	2.20	2.13	0.43	
154	12.58	1.29	2.10	20.0	103.0	1.48	0.58	0.53	
51	13.83	1.65	2.60	17.2	94.0	2.45	2.99	0.22	

In [14]:
```
y_train.head()
```

Out[14]:
```
2      0
100    1
122    1
154    2
51     0
Name: target, dtype: int32
```

让我们从简单的线性 SVM 分类器开始。它会自动使用 One-vs-All（也称为 One-vs-the-Rest，OvR）策略，因此我们不需要做任何特别的事情来处理多个类。容易，对吧？

In [24]:
```
lin_clf = LinearSVC(random_state=42)
lin_clf.fit(X_train, y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\_base.py:1225: ConvergenceWarnin
g: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

Out[24]:
```
LinearSVC(random_state=42)
```

不好了！ 它未能收敛。 你能猜出为什么吗？ 你认为我们必须增加训练迭代次数吗？ 让我们来看看：

In [25]:
```
lin_clf = LinearSVC(max_iter=1_000_000, random_state=42)
lin_clf.fit(X_train, y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\_base.py:1225: ConvergenceWarnin
g: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

Out[25]:
```
LinearSVC(max_iter=1000000, random_state=42)
```

即使经过一百万次迭代，它仍然没有收敛。一定还有别的问题。

让我们仍然用 cross_val_score 评估这个模型，它将作为一个基线：

In [26]:
```python
from sklearn.model_selection import cross_val_score

cross_val_score(lin_clf, X_train, y_train).mean()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\_base.py:1225: ConvergenceWarnin
g: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\_base.py:1225: ConvergenceWarnin
g: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\_base.py:1225: ConvergenceWarnin
g: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\_base.py:1225: ConvergenceWarnin
g: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\_base.py:1225: ConvergenceWarnin
g: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

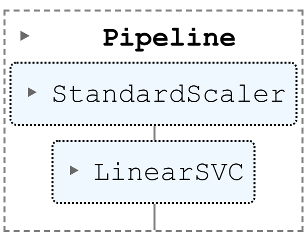
Out[26]:
```
0.90997150997151
```

这个数据集 91% 的准确率并不高。那么你猜到问题是什么了吗？

没错，我们忘记缩放特征了！在使用 SVM 时，永远记得缩放特征：

In [28]:
```python
from sklearn.pipeline import make_pipeline

lin_clf = make_pipeline(StandardScaler(),
                        LinearSVC(random_state=42))
lin_clf.fit(X_train, y_train)
```

Out[28]:



现在它收敛没有任何问题。 让我们来衡量一下它的性能：

In [29]:
```python
from sklearn.model_selection import cross_val_score

cross_val_score(lin_clf, X_train, y_train).mean()
```

Out[29]:
```
0.9774928774928775
```

好的！ 我们得到了 97.7% 的准确率，这要好得多。

让我们看看内核化 SVM 是否会做得更好。我们现在将使用默认的 SVC：

In [30]:
```python
svm_clf = make_pipeline(StandardScaler(), SVC(random_state=42))

cross_val_score(svm_clf, X_train, y_train).mean()
```

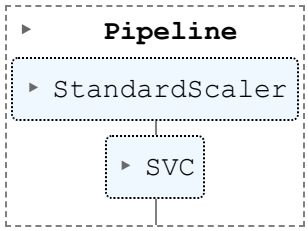
Out[30]:
```
0.9698005698005698
```

这并没有更好，但也许我们需要做一些超参数调整：

In [32]:
```python
from sklearn.model_selection import RandomizedSearchCV
```

```
from scipy.stats import loguniform, uniform

param_distrib = {
    "svc__gamma": loguniform(0.001, 0.1),
    "svc__C": uniform(1, 10)
}

rnd_search_cv = RandomizedSearchCV(svm_clf, param_distrib, n_iter=100, cv=5,
                                   random_state=42)

rnd_search_cv.fit(X_train, y_train)
rnd_search_cv.best_estimator_
```

Out[32]:



In [33]:
```
rnd_search_cv.best_score_
```

Out[33]:
```
0.9925925925925926
```

啊，这看起来棒极了！让我们选择这个模型。现在我们可以在测试集上测试它：

In [34]:
```
rnd_search_cv.score(X_test, y_test)
```

Out[34]:
```
0.9777777777777777
```

这种经过调整的内核化 SVM 比 LinearSVC 模型表现更好，但我们在测试集上得到的分数低于我们使用交叉验证测量的分数。这很常见：由于我们进行了如此多的超参数调整，最终导致交叉验证测试集略微过度拟合。在测试集上获得更好的结果之前，我们很想稍微调整超参数，但这可能无济于事，因为我们会开始过度拟合测试集。反正这个分数一点都不差，就此打住吧。

Exercise 11

在加州住房数据集上训练和微调一个 SVM 回归器。您可以使用原始数据集，而不是我们在第2章中使用的调整版本，您可以使用 **sklearn.datasets.fetch_california_housing()** 进行加载。这些目标是数十万美元。由于有 20000 个实例，SVM 可能会很慢，因此对于超参数调优，您应该使用更少的实例（例如 2000 个）来测试更多的超参数组合。你最好模型的 RMSE 是什么？

答案：

让我们加载数据集：

In [35]:
```
from sklearn.datasets import fetch_california_housing

housing = fetch_california_housing()

X = housing.data
y = housing.target
```

将其拆分为训练集和测试集：

In [37]:
```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42)
```

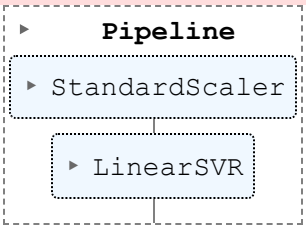
不要忘记缩放数据！

让我们先训练一个简单的 LinearSVR：

In [38]:
```
from sklearn.svm import LinearSVR

lin_svr = make_pipeline(StandardScaler(),
                        LinearSVR(random_state=42))

lin_svr.fit(X_train, y_train)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\svm\_base.py:1225: ConvergenceWarnin
g: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

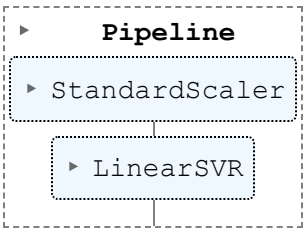
Out[38]:



它没有收敛，所以让我们增加 max_iter：

In [39]:
```
lin_svr = make_pipeline(StandardScaler(),
                        LinearSVR(max_iter=5000, random_state=42))

lin_svr.fit(X_train, y_train)
```

Out[39]:



让我们看看它在训练集上的表现：

In [40]:
```
from sklearn.metrics import mean_squared_error

y_pred = lin_svr.predict(X_train)

mse = mean_squared_error(y_train, y_pred)

mse
```

Out[40]:
```
0.9595484665813284
```

让我们看看 RMSE：

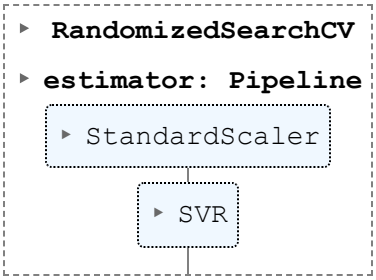
In [48]:
```
import numpy as np

np.sqrt(mse)
```

Out[48]:
```
0.979565447829459
```

在此数据集中，目标代表数十万美元。RMSE 给出了您应该预期的错误类型的粗略概念（较大错误的权重较高）：因此使用此模型我们可以预期接近 98000 美元的错误！不是很好。让我们看看我们是否可以使用 RBF 内核做得更好。我们将使用带有交叉验证的随机搜索来为 C 和 $gamma$ 找到合适的超参数值：

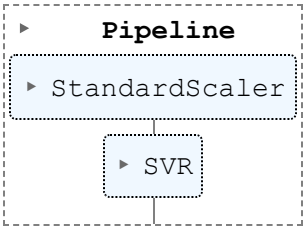
In [45]:
```
from sklearn.svm import SVR
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import loguniform, uniform

svm_clf = make_pipeline(StandardScaler(), SVR())

param_distrib = {
    "svr__gamma": loguniform(0.001, 0.1),
    "svr__C": uniform(1, 10)
}
rnd_search_cv = RandomizedSearchCV(svm_clf, param_distrib,
                                   n_iter=100, cv=3, random_state=42)

rnd_search_cv.fit(X_train[:2000], y_train[:2000])
```

Out[45]:



In [46]:
```
rnd_search_cv.best_estimator_
```

Out[46]:



In [47]:
```
-cross_val_score(rnd_search_cv.best_estimator_,
                 X_train, y_train,
                 scoring="neg_root_mean_squared_error")
```

Out[47]:
```
array([0.58835648, 0.57468589, 0.58085278, 0.57109886, 0.59853029])
```

看起来比线性模型好得多。 让我们选择这个模型并在测试集上对其进行评估：

In [50]:
```
y_pred = rnd_search_cv.best_estimator_.predict(X_test)

rmse = mean_squared_error(y_test, y_pred, squared=False)

rmse
```

Out[50]:
```
0.5854732265172228
```

所以支持向量机在 Wine 数据集上工作得很好，但在 California Housing 数据集上就没那么好了。在第 2 章中，我们发现随机森林对该数据集的效果更好。