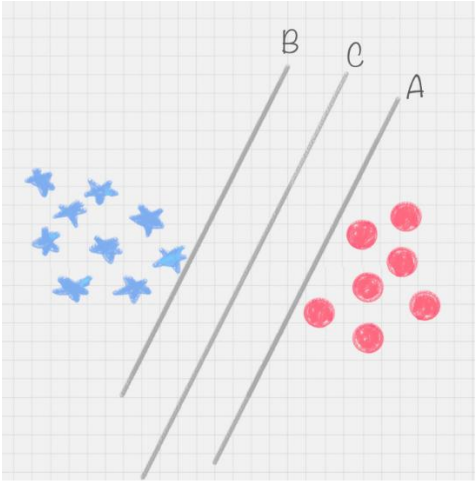
作业 15/26 支持向量机调研报告

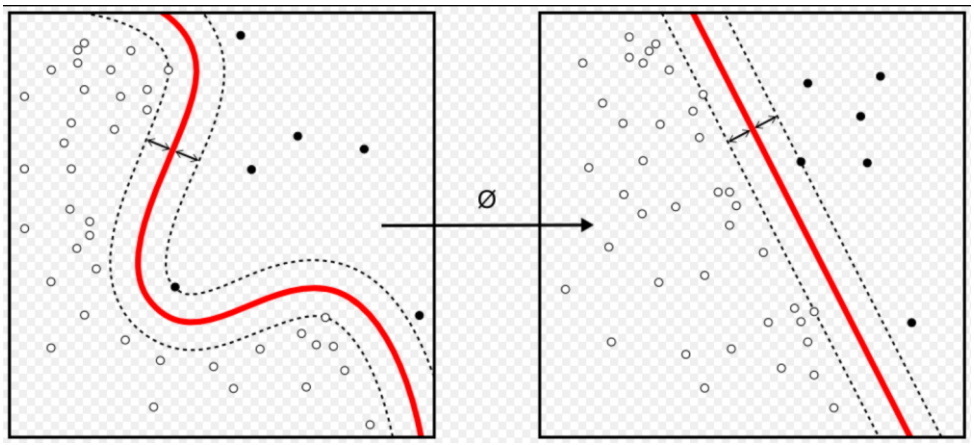
1　支持向量机（SVM）介绍

SVM 是一种应用广泛的监督学习算法。虽然主要用于分类任务（判断数据点属于哪个类别），但它也可以用于回归问题。它的基本思想非常直观，目标是寻找最优分界线（超平面），想象你要在两类不同的数据点（比如红色和蓝色的点）之间画一条分界线。SVM 的目标不仅仅是简单地将它们分开，而是要找到**最能**分开两类数据的那条线（在更高维度，是一个平面或超平面）。也就是说，这条线离两边最近的样本点都要尽可能远。两边最近点到这条分界线的距离形成的“隔离带”就叫间隔，最大化间隔是 SVM 算法的核心，通过最大化间隔，可以获得更强的泛化能力，因为间隔越大，意味着分界线离数据点的“缓冲地带”越宽，模型对于新数据、噪声数据的判断通常会更稳定，降低过拟合风险。



定义这个最优分界线的数据点就是支持向量。它们是离分界线最近的那些训练样本点。SVM 的学习过程只依赖于这些关键的支持向量，删掉其他点并不影响分界线的位置。这是 SVM 高效的原因之一。

现实中很多数据不是一条直线（或超平面）就能完美分开的（称为线性不可分）。SVM 解决此问题的方法是核技巧。思路是不直接在原始数据空间里硬分，而是把原始数据点巧妙地映射到一个更高维的空间。在这个新的高维空间里，原本在低维空间纠缠在一起、无法用平面分开的数据点，很可能就变得线性可分了。实现这种映射不需要我们手动进行复杂的计算转换。SVM 通过一个核函数来隐式地计算高维空间中点的关系，从而高效地在高维空间中找到线性分界线。核技巧是 SVM 处理复杂非线性问题的核心。



SVM 的优点是，在高维特征空间中工作良好，由于其依赖的支持向量通常较少，对中小规模数据集很有效，鲁棒性强，泛化能力好。

2 向量运算

向量运算是 SVM 算法的数学基础。

2.1 超平面定义

公式： $\mathbf{W} \cdot \mathbf{X} - \mathbf{b} = 0$ (有时也写作 $\mathbf{W} \cdot \mathbf{X} + \mathbf{b} = 0$，取决于 b 符号定义)

涉及的向量运算：点积 $(W \cdot X)$，这是最核心的向量运算。

W = [w1, w2, ..., wn]表示法向量，它是一个 n 维向量 (n 是输入特征 X 的维度)。它的方向决定了超平面的朝向。想象一个平面，法向量就是垂直于这个平面的箭头。箭头指向的一侧通常定义为正类方向。

X = [x1, x2, ..., xn]表示输入样本的特征向量，代表一个数据点。

$W \cdot X$ 的结果是一个标量。从几何上看，点积 $W \cdot X$ 等于向量 W 和 X 的模长乘以它们之间夹角的余弦，当 W 是单位向量时，$|W \cdot X|$ 的几何意义就是点 X 到超平面的垂直距离。

偏置项 b 表示一个标量值。它的作用是将超平面平移远离原点。改变 b 的值会让整个决策平面沿着法向量 W 的方向平行移动，从而决定超平面的具体位置。

这个公式利用点积定义了一个 n 维空间中的超平面。任何一个输入点 X 代入这个公式：

若 $W \cdot X - b > 0$，则判定为正类。

若 $W \cdot X - b < 0$，则判定为负类。

若 $W \cdot X - b = 0$，则位于超平面上。

2.2 间隔最大化

找到参数 W 和 b，使得所有训练样本都满足分类正确，且使得间隔最大化。

对于线性可分的情况，决策超平面到其两侧的“边界”（$W \cdot X - b = \pm 1$）的垂直距离是相等的。

在这个过程中，使用到了点到平面的距离公式（推广到 n 维）：对于一个点 X_0 和超平面 $W \cdot X - b = 0$，距离 $d = |W \cdot X\_0 - b| / ||W||$。

然后转化为一个优化问题，最大化间隔 => 最小化 $\|W\|$，等价于最小化 W 的模的平方 $\|W\|^2$（平方方便求导）。约束条件 $(y\_i\ (W \cdot X\_i + b) >= 1)$。对于所有训练样本 i ，分类正确且保证最小间隔。

2.3 核函数计算

对于非线性可分的数据，需要在原始输入空间找到一个超平面通常是不可能的。将原始输入特征 X 通过一个非线性映射 $\Phi(X)$ 映射到一个更高维甚至无限维的特征空间。在这个高维空间中，数据可能变得线性可分。

直接计算映射后的高维特征 $\Phi(X)$ 及其点积 $\Phi(X\_i) \cdot \Phi(X\_j)$ 可能计算量巨大或难以显式进行。

而核函数 $K(X\_i, X\_j)$ 的本质就是 $\Phi(X\_i) \cdot \Phi(X\_j)$ 的结果，因此不需要显式计算映射函数 Φ 和高维特征向量本身。我们只需要在原始输入空间中计算一个函数 $K(X\_i, X\_j)$，这个函数的结果等价于在高维特征空间中计算那两个映射后的向量的点积。

最核心的运算仍然是点积 X_i • X_j。

3 使用 sk-learn 实现一个简单的 SVM

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report


# 加载乳腺癌数据集
cancer = datasets.load_breast_cancer()
X = cancer.data
y = cancer.target

# 标准化
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 划分训练测试集
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42
)


# 创建SVM分类器
svm_model = SVC(kernel='rbf', C=1.0, gamma='scale', random_state=42)

# 训练模型
svm_model.fit(X_train, y_train)

# 预测与评估
y_pred = svm_model.predict(X_test)

print("准确率:", accuracy_score(y_test, y_pred))
print("\n分类报告:")
print(classification_report(y_test, y_pred))

```

```python
# 选择前两个特征进行可视化
X_2d = X_scaled[:, :2]

# 重新训练模型
svm_2d = SVC(kernel='rbf', C=1.0, gamma=0.5)
svm_2d.fit(X_2d, y)

# 创建网格点
x_min, x_max = X_2d[:, 0].min() - 1, X_2d[:, 0].max() + 1
y_min, y_max = X_2d[:, 1].min() - 1, X_2d[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

# 预测网格点类别
Z = svm_2d.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# 绘制
plt.contourf(xx, yy, Z, alpha=0.8)
plt.scatter(X_2d[:, 0], X_2d[:, 1], c=y, edgecolors='k')
plt.xlabel('Feature 1 (Standardized)')
plt.ylabel('Feature 2 (Standardized)')
plt.title('SVM Decision Boundary with RBF Kernel')
plt.show()
```

(variable) y: Any

运行代码，得到的结果如下：

```
(E:\conda_envs\AIMath) PS E:\conda_python\AIMath> python .\question15.py
准确率: 0.9707602339181286

分类报告:
              precision    recall  f1-score   support

           0       0.95      0.97      0.96        63
           1       0.98      0.97      0.98       108

    accuracy                           0.97       171
   macro avg       0.97      0.97      0.97       171
weighted avg       0.97      0.97      0.97       171

(E:\conda_envs\AIMath) PS E:\conda_python\AIMath>
```