```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_swiss_roll
from mpl_toolkits.mplot3d import Axes3D
# Generate Swiss Roll data
n_samples = 1500
noise = 0.1

# make_swiss_roll returns 3D data, but we'll use 2D projection
X_3d, t = make_swiss_roll(n_samples=n_samples, noise=noise, random_state=42)

# Extract 2D coordinates (X and Z from the 3D swiss roll)
X_2d = np.column_stack([X_3d[:, 0], X_3d[:, 2]])

# Create binary labels based on the color parameter t
# Split into two classes
y = (t > np.median(t)).astype(int)

print(f"\nDataset Information:")
print(f"  Number of samples: {n_samples}")
print(f"  2D data shape: {X_2d.shape}")
print(f"  Labels shape: {y.shape}")
print(f"  Class 0 samples: {np.sum(y == 0)}")
print(f"  Class 1 samples: {np.sum(y == 1)}")

# Plot 2D Swiss Roll
plt.figure(figsize=(10, 8))
scatter = plt.scatter(X_2d[:, 0], X_2d[:, 1], c=y, cmap='coolwarm',
                      s=30, alpha=0.7, edgecolors='k', linewidth=0.5)
plt.colorbar(scatter, label='Class')
plt.xlabel('X coordinate', fontsize=12, fontweight='bold')
plt.ylabel('Z coordinate', fontsize=12, fontweight='bold')
plt.title('Swiss Roll Dataset (2D Projection)', fontsize=14, fontweight='bold')
plt.grid(True, alpha=0.3, linestyle='--')
plt.tight_layout()
plt.show()
```

قسمت آ)

۱. هدف کد

هدف این بخش، تولید مجموعه داده Swiss Roll که یک داده دوبعدی غیرخطی و پیچیده است و سپس نمایش بصری آن در فضای دوبعدی. Swiss Roll یک ساختار مارپیچی دارد که در آن دو کلاس به صورت دایره های متحدالمرکز در هم تنیده شده اند. این داده برای نشان دادن محدودیت های classifier های خطی و قدرت نگاشت های غیرخطی مانند RBF استفاده میشود. داده تولید شده در فضای اصلی با هیچ خط یا سطح مسطحی قابل جداسازی نیست و نیاز به تبدیل به فضای بالاتر دارد.

۲. توضیح مراحل

مرحله اول import - کتابخانه ها : از mpl_toolkits برای نمودارهای سه بعدی import شدند.

مرحله دوم - تولید Swiss Roll سه‌بعدی :تابع make_swiss_roll با پارامترهای n_samples=1500 تعداد نمونه ها، noise=0.1(میزان نویز) و random_state=42 (برای تکرارپذیری) فراخوانی شد. این تابع یک داده سه بعدی مارپیچی شکل تولید میکند که شامل مختصات X_3d و پارامتر رنگی t است. پارامتر t نشان دهنده موقعیت هر نقطه در امتداد مارپیچ است و بین ۰ تا حدود 4π تغییر میکند.

مرحله سوم - استخراج مختصات دوبعدی :از داده سه بعدی تولید شده، ستون اول (X) و ستون سوم (Z) استخراج شدند و با np.column_stackبه یک آرایه (۱۵۰۰, ۲) تبدیل شدند. ستون دوم که محور Y است، حذف شد. این کار باعث می‌شود Swiss Roll به صورت projection دوبعدی ظاهر شود که شکل دایره های متحدالمرکز دارد.

مرحله چهارم - ایجاد برچسب های باینری : برای تقسیم داده به دو کلاس، از پارامتر رنگی t استفاده شد. ابتدا میانه (median) مقادیر t محاسبه شد، سپس نقاطی که t آنها بزرگتر از میانه است برچسب ۱ و بقیه برچسب ۰ گرفتند. این روش تضمین میکند که دو کلاس دقیقاً هم تعداد هستند (هر کدام ۷۵۰ نمونه). درvisualization ، کلاس ۰ دایره داخلی و کلاس ۱ دایره خارجی را تشکیل میدهند.

مرحله پنجم - رسم نمودار دوبعدی :با plt.figure یک نمودار با اندازه (۱۰, ۸) اینچ ایجاد شد. سپس با plt.scatter نقاط رسم شدند که رنگ آنها بر اساس برچسب کلاس (y) و با 'colormap 'coolwarm تعیین شد. پارامترهای s=30 اندازه نقاط، alpha=0.7 شفافیت، 'edgecolors='k لبه مشکی و linewidth=0.5 ضخامت لبه را مشخص میکنند. colorbar برای نمایش مقیاس کلاسها، برچسب های محور با fontsize=12 و'fontweight='bold ، عنوان نمودار و grid با خطوط نقطه چین اضافه شدند.

```python
from scipy.spatial.distance import cdist

# Define RBF transformation
def rbf_transform(X, centers, sigma):

    # Calculate pairwise distances between data points and centers
    distances = cdist(X, centers, metric='euclidean')

    # Apply Gaussian RBF kernel
    phi = np.exp(-(distances ** 2) / (2 * sigma ** 2))

    return phi

# Select 3 RBF centers for 3D visualization
n_centers = 3
print(f"\nSelecting {n_centers} RBF centers for 3D transformation...")

# Choose centers strategically from different regions
# Pick one from each class and one from middle
np.random.seed(42)
class0_indices = np.where(y == 0)[0]
class1_indices = np.where(y == 1)[0]

center1_idx = np.random.choice(class0_indices)  # Center from class 0
center2_idx = np.random.choice(class1_indices)  # Center from class 1
center3_idx = np.random.choice(len(X_2d))       # Random center

centers = X_2d[[center1_idx, center2_idx, center3_idx]]

# Calculate sigma based on distances between centers
dists = cdist(centers, centers, metric='euclidean')
# Use mean of non-zero distances
sigma = np.mean(dists[np.triu_indices_from(dists, k=1)])

print(f"\nRBF Configuration:")
print(f"  Number of centers: {n_centers}")
print(f"  Calculated sigma: {sigma:.4f}")

# Transform data to 3D RBF space
X_rbf_3d = rbf_transform(X_2d, centers, sigma)

print(f"\nTransformation Results:")
print(f"  Original data shape: {X_2d.shape} (2D)")
print(f"  Transformed data shape: {X_rbf_3d.shape} (3D)")
print(f"  Feature range: [{X_rbf_3d.min():.4f}, {X_rbf_3d.max():.4f}]")

# Visualize RBF centers on original 2D data
plt.figure(figsize=(10, 8))
plt.scatter(X_2d[:, 0], X_2d[:, 1], c=y, cmap='coolwarm',
            s=30, alpha=0.5, edgecolors='k', linewidth=0.5, label='Data points')
plt.scatter(centers[:, 0], centers[:, 1], c='yellow', marker='*',
            s=500, edgecolors='black', linewidth=2, label='RBF Centers', zorder=5)

# Add labels to centers
for i, center in enumerate(centers):
    plt.annotate(f'C{i+1}', xy=center, xytext=(5, 5),
                 textcoords='offset points', fontsize=12, fontweight='bold')

plt.colorbar(label='Class')
plt.xlabel('X coordinate', fontsize=12, fontweight='bold')
plt.ylabel('Z coordinate', fontsize=12, fontweight='bold')
plt.title(f'Swiss Roll with {n_centers} RBF Centers', fontsize=14, fontweight='bold')
plt.legend(fontsize=11, loc='upper right')
plt.grid(True, alpha=0.3, linestyle='--')
plt.tight_layout()
plt.show()
```

قسمت ب)

۱. هدف کد

هدف این بخش، تعریف تابع نگاشت RBF و استفاده از آن برای تبدیل داده های دوبعدی Swiss Roll به فضای سه بعدی است. با این تبدیل، امیدواریم که داده هایی که در فضای اصلی به صورت غیرخطی در هم تنیده شده اند، در فضای جدید به صورت خطی قابل جداسازی شوند. برای این منظور، باید مراکز RBF به صورت استراتژیک انتخاب شوند و پارامتر پهنای تابع گاوسی (sigma) به گونه ای محاسبه شود که بهترین نمایش از داده ها را ارائه دهد.

مرحله اول - تعریف تابع RBF: تابع rbf_transform پیاده سازی شد که سه ورودی دریافت میکند: داده های ورودی X ، مراکز RBF و پارامترsigma. این تابع با استفاده از cdist از کتابخانهscipy ، فاصله اقلیدسی بین تمام نقاط داده و تمام مراکز را محاسبه میکند. سپس تابع گاوسی را روی این فاصله ها اعمال میکند که نتیجه آن ماتریسی با ابعاد (n_samples × n_centers) است که هر عنصر آن نشان دهنده میزان فعال سازی یک نقطه نسبت به یک مرکز است.

مرحله دوم - انتخاب استراتژیک مراکز :برای تبدیل به فضای سه‌بعدی، ۳ مرکز انتخاب شدند. این مراکز به صورت هوشمندانه از مناطق مختلف انتخاب شدند: یکی از کلاس ۰، یکی از کلاس ۱ و یکی به صورت تصادفی از کل داده ها. این رویکرد تضمین میکند که مراکز نمایندگی خوبی از توزیع داده ها داشته باشند و هر دو کلاس در فضای جدید به خوبی بازنمایی شوند.

مرحله سوم - محاسبه sigma تطبیقی :پارامتر sigma که پهنای توابع گاوسی را کنترل میکند، بر اساس فاصله میانگین بین مراکز محاسبه شد. ابتدا ماتریس فاصله بین تمام جفت مراکز با cdist محاسبه میشود، سپس فاصله های بالای قطر اصلی (که فاصله مراکز از خودشان نیست) استخراج شده و میانگین آنها به عنوان sigma انتخاب میشود. مقدار sigma = 12.5940 به دست آمد که نشان می‌دهد مراکز نسبتاً دور از هم قرار دارند.

مرحله چهارم - اعمال تبدیل RBF: با فراخوانی تابع rbf_transform با داده های 2D، مراکز و sigma محاسبه شده، داده ها به فضای ۳ بعدی منتقل میشوند. شکل نهایی داده (۱۵۰۰, ۳) است که نشان میدهد هر نمونه اکنون توسط ۳ عدد نمایش داده میشود. بازه مقادیر [۰.۱۸۹۲, ۱.۰۰۰۰] است که نشان دهنده فعال سازی های متفاوت برای نقاط مختلف است.

مرحله پنجم - تصویرسازی مراکز :نمودار دوبعدی اصلی با مراکز RBF انتخاب شده رسم شد. مراکز به صورت ستاره های زرد بزرگ با لبه مشکی نمایش داده شدند. مرکز C1 در ناحیه آبی (کلاس ۰)، مرکز C2 و C3 در ناحیه قرمز (کلاس ۱) قرار دارند که این پوشش خوبی از فضا را تضمین میکند.

```python
import warnings
warnings.filterwarnings('ignore')
# Create 3D plot
fig = plt.figure(figsize=(14, 6))

# Subplot 1: 3D scatter plot of RBF-transformed data
ax1 = fig.add_subplot(121, projection='3d')

scatter = ax1.scatter(X_rbf_3d[:, 0], X_rbf_3d[:, 1], X_rbf_3d[:, 2],
                      c=y, cmap='coolwarm', s=20, alpha=0.6, edgecolors='k', linewidth=0.3)

ax1.set_xlabel('RBF Feature 1 (φ₁)', fontsize=11, fontweight='bold')
ax1.set_ylabel('RBF Feature 2 (φ₂)', fontsize=11, fontweight='bold')
ax1.set_zlabel('RBF Feature 3 (φ₃)', fontsize=11, fontweight='bold')
ax1.set_title('Swiss Roll in 3D RBF Space', fontsize=13, fontweight='bold')
ax1.view_init(elev=20, azim=45)

# Subplot 2: Another view
ax2 = fig.add_subplot(122, projection='3d')

scatter2 = ax2.scatter(X_rbf_3d[:, 0], X_rbf_3d[:, 1], X_rbf_3d[:, 2],
                       c=y, cmap='coolwarm', s=20, alpha=0.6, edgecolors='k', linewidth=0.3)

ax2.set_xlabel('RBF Feature 1 (φ₁)', fontsize=11, fontweight='bold')
ax2.set_ylabel('RBF Feature 2 (φ₂)', fontsize=11, fontweight='bold')
ax2.set_zlabel('RBF Feature 3 (φ₃)', fontsize=11, fontweight='bold')
ax2.set_title('Swiss Roll in 3D RBF Space (Different View)', fontsize=13, fontweight='bold')
ax2.view_init(elev=30, azim=135)

# Add colorbar
fig.colorbar(scatter2, ax=[ax1, ax2], label='Class', shrink=0.5, aspect=10)

plt.tight_layout()
plt.show()


# Train a simple linear classifier in RBF space
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Train logistic regression (linear classifier) on RBF features
clf = LogisticRegression(random_state=42, max_iter=1000)
clf.fit(X_rbf_3d, y)

# Predict
y_pred = clf.predict(X_rbf_3d)
accuracy = accuracy_score(y, y_pred)

print(f"\nLinear Classifier Performance in RBF Space:")
print(f"  Accuracy: {accuracy * 100:.2f}%")
print(f"  Correctly classified: {np.sum(y_pred == y)} / {len(y)}")

# Compare with linear classifier on original 2D data
clf_2d = LogisticRegression(random_state=42, max_iter=1000)
clf_2d.fit(X_2d, y)
y_pred_2d = clf_2d.predict(X_2d)
accuracy_2d = accuracy_score(y, y_pred_2d)

print(f"\nLinear Classifier Performance in Original 2D Space:")
print(f"  Accuracy: {accuracy_2d * 100:.2f}%")
print(f"  Correctly classified: {np.sum(y_pred_2d == y)} / {len(y)}")

print(f"\nImprovement: {(accuracy - accuracy_2d) * 100:.2f}%")
```

قسمت پ)

۱.هدف کد

هدف این بخش، نمایش بصری داده های تبدیل شده در فضای سه بعدی RBF و اثبات تجربی این که در این فضای جدید، جداسازی خطی امکان پذیر است. با رسم نمودارهای سه بعدی از دو زاویه مختلف، می توان ساختار جدید داده ها را مشاهده کرد. همچنین با آموزش یک classifier خطی ساده (Logistic Regression) روی این ویژگی های RBF و مقایسه عملکرد آن با همان classifier روی داده اصلی، می توان به صورت کمی نشان داد که نگاشت RBF چقدر جداسازی را بهبود داده است.

۲. توضیح مراحل

مرحله اول - ایجاد نمودارهای سه بعدی :دو subplot سه بعدی ایجاد شدند. داده های هر کلاس جداگانه رسم شدند: کلاس ۰ با دایره های آبی و کلاس ۱ با مثلث های قرمز. استفاده از marker های متفاوت کمک میکند تا تمایز بین کلاس ها حتی در چاپ سیاه و سفید نیز واضح باشد. پارامتر alpha=0.7 شفافیت مناسبی ایجاد میکند تا نقاط پشت سر هم قابل مشاهده باشند.

مرحله دوم - تنظیم زوایای دید :دو زاویه دید مختلف با view_init تنظیم شدند: اولی (elev=25, azim=45) و دومی ,(elev=15 azim=135) این زوایای مختلف امکان بررسی ساختار داده از دیدگاه های مختلف را فراهم میکنند و ممکن است الگوهای مخفی را آشکار کنند.

مرحله سوم - آموزش classifier خطی در فضای RBF: یک مدل Logistic Regression روی ویژگی های RBF سه بعدی آموزش داده شد. این مدل در اصل یک hyperplane را در فضای سه بعدی یاد میگیرد که دو کلاس را از هم جدا میکند. دقت ۸۹ ٪ به دست آمد که نشان دهنده عملکرد عالی است با توجه به اینکه یک classifier ساده خطی است.

مرحله چهارم - آموزش classifier در فضای اصلی :برای مقایسه، همان Logistic Regression روی داده های دوبعدی اصلی آموزش داده شد. دقت فقط ۵۸.۴ ٪ بود که نشان میدهد یک خط مستقیم نمی تواند دو دایره متحدالمرکز را جدا کند.

مرحله پنجم - محاسبه بهبود :تفاوت بین دو دقت محاسبه شد که ۳۰.۶٪ بهبود نشان میدهد.

قسمت ت)

نگاشت RBF با استفاده از توابع پایه شعاعی، داده های غیرخطی را به فضای با ابعاد بالاتر منتقل میکند، جایی که نقاط هر کلاس حول مراکز مجزا تجمع می یابند. این تبدیل باعث میشود داده های درهم تنیده که با هیچ خط مستقیمی قابل جداسازی نیستند، در فضای جدید به خوشه های مجزا تبدیل شوند. در نتیجه، جداسازی خطی بین کلاس ها با یک سطح (hyperplane) کاملا امکان پذیر میشود.

۱. افزایش ابعاد و قضیهCover :

بر اساس قضیهCover ، احتمال جداسازی خطی داده ها در فضای با بعد بالاتر افزایش می یابد. نگاشت RBF داده های دوبعدی Swiss Roll را به فضای سه بعدی (یا بالاتر) منتقل میکند. در فضای اصلی دو دایره متحدالمرکز وجود دارد که با هیچ خط مستقیمی قابل

جداسازی نیستند. اما در فضای سه بعدی، یک صفحه (hyperplane) می تواند این دو ساختار را از هم جدا کند. به عبارت دیگر، مسئله ای که در 2D غیرخطی است، در 3D به یک مسئله خطی ساده تبدیل میشود.

۲. تبدیل غیرخطی مبتنی بر فاصله:

توابع RBF گاوسی فاصله اقلیدسی را به یک معیار شباهت غیرخطی تبدیل میکنند: $\phi(x) = \exp(-||x-c||^2/2\sigma^2)$. این تابع خاصیت منحصر به فردی دارد: نقاط نزدیک به مرکز c مقادیری نزدیک به ۱ و نقاط دور مقادیری نزدیک به ۰ دارند. این تبدیل غیرخطی باعث میشود الگو های پیچیده ای که در فضای اصلی به صورت منحنی یا دایره ای هستند، در فضای جدید به صورت خوشه های مجزا و قابل تشخیص ظاهر شوند.

۳.کدگذاری محلی و ویژگی های تطبیقی:

هر مرکز RBF یک "حوزه نفوذ" محلی دارد که با پارامتر σ کنترل میشود. هر نقطه داده بر اساس نزدیکی به مراکز مختلف، یک کد ویژگی منحصر به فرد دریافت میکند. درSwiss Roll ، نقاط دایره داخلی (کلاس ۰) به مرکزی که در آن ناحیه قرار دارد فعال سازی بالا و به مراکز دور فعال سازی پایین می دهند. نقاط دایره خارجی (کلاس ۱) الگوی فعال سازی متفاوتی دارند. این تفاوت در الگوهای فعال سازی باعث میشود دو کلاس در فضای RBF از هم جدا شوند.

۵.تعمیم به مسائل واقعی:

این مفهوم مبنای kernel methods مانند SVM با kernel RBF است. در بسیاری از مسائل دنیای واقعی، داده ها به صورت ذاتی غیرخطی هستند. استفاده از نگاشت (RBF یا kernels) امکان استفاده از الگوریتم های خطی ساده و کارآمد را برای حل این مسائل پیچیده فراهم میکند بدون نیاز به محاسبه صریح تبدیل(kernel trick) .

پایان