.....

گزارش .....

عنوان: .....

ارائه‌دهنده:

.....

استاد درس:

.....

زمستان ۱۴۰۰


بسم الله الرحمن الرحیم

چکیده

فرآیند گوسی یک روش یادگیری با نظارت است. رگرسیون فرآیند گوسی و طبقه‌بندی فرآیند گوسی در این گزارش به اختصار مورد بررسی قرار می‌گیرد. در رگرسیون فرآیند گوسی هایپرپارامترهای هسته در طول برازش مدل به‌وسیله حداکثرکردن احتمال شباهت حاشیه‌ای بهینه می‌شوند و در طبقه‌بندی، فرآیند گوسی بر روی یک تابع که عملکرد پنهان دارد پیاده می‌شود. سپس از طریق یک تابع پیوند بی‌اثر می‌شود تا طبقه‌بندی احتمالی به دست آید. تابع لجستیک به‌عنوان تابع پیوند انتخاب می‌شود. هسته‌ها جزء مهم فرآیندهای گوسی هستند که شکل احتمال پسین و احتمال پیشین را تعیین می‌کنند. دو دسته از هسته‌ها را به نام هسته ثابت و غیرثابت نام می‌بریم. با توجه به کاربرد، مزایا و معایب هر یک از روش‌های ارایه شده، می‌توان برای کاربردهای خاص از روش موردنظر استفاده نمود.

کلیدواژه: فرآیند گوسی، طبقه‌بندی فرآیند گوسی، رگرسیون فرآیند گوسی، هسته.

فهرست مطالب

عنوان **صفحه**

فهرست شکل‌ها

عنوان **صفحه**

فهرست علایم و نشانه‌ها

عنوان	علامت اختصاری
طول	L
تناوب	P
ورودی	X
خروجی	Y
فاصله اقلیدسی	d

مقدمه

۱–۱– پیشگفتار

فرآیند گوسی یک روش یادگیری با نظارت است. در این گزارش به توضیح رگرسیون فرآیند گوسی و طبقه‌بندی فرآیند گوسی می‌پردازیم. در رگرسیون فرآیند گوسی ابتدا تابع احتمال پیشین مشخص می‌شود که کوواریانس آن با یک ویژگی از هسته تعیین می‌شود. هایپرپارامترهای هسته در طول برازش رگرسیون فرآیند گوسی به‌وسیله حداکثرکردن احتمال شباهت حاشیه‌ای بهینه می‌شوند. استفاده از هسته‌ای به نام هسته سفید، علاوه بر مزایای تخمین به روش استاندارد رگرسیون گوسی، امکان پیش‌بینی، بدون برازش احتمال پیشین را فراهم می‌کند.

در طبقه‌بندی، فرآیند گوسی برای اهداف طبقه‌بندی پیاده‌سازی می‌شود. فرآیند گوسی را بر روی یک تابع که عملکرد پنهان دارد، اعمال می‌کند. سپس از طریق یک تابع پیوند بی‌اثر می‌شود تا طبقه‌بندی احتمالی به دست آید. در طبقه‌بندی فرآیند گوسی تابع لجستیک به‌عنوان تابع پیوند، انتخاب می‌شود. این روش از طبقه‌بندی چند کلاسه استفاده می‌کند که آموزش و پیش‌بینی مبتنی بر دو روش، یک در مقابل استراحت یا یک در مقابل یک انجام می‌شود.

هسته‌ها یک جزء مهم فرآیندهای گوسی هستند که شکل احتمال پسین و احتمال پیشین را تعیین می‌کنند. دودسته از هسته‌ها را به نام هسته ثابت و غیرثابت نام می‌بریم. هسته‌های ثابت را می‌توان به هسته‌های همسانگرد و ناهمسانگرد تقسیم کرد. درنهایت با توجه به نوع کاربرد و مزایا و معایب هر یک از روش‌های ارایه شده و با توجه به مثال‌ها می‌توان برای کاربردهای خاص از روش موردنظر استفاده نمود.

۱–۲– هدف

هدف اساسی از تهیه این گزارش آشنایی با فرآیندهای گوسی است، که به‌طور خاص به طبقه‌بندی و رگرسیون فرآیندهای گوسی می‌پردازیم. مزایا، معایب و مثال‌هایی از هرکدام ارایه می‌شود که با توجه به نیاز و کاربرد در مسائل مختلف مورداستفاده قرار می‌گیرند.

فصل ۲– مقدمه‌ای بر روش‌های طبقه‌بندی و رگرسیون مبتنی بر فرآیند گوسی

۲–۱– مقدمه

فرآیندهای گوسی یک روش یادگیری با نظارت است که در جهت رفع مشکلات رگرسیون و طبقه‌بندی طراحی‌شده است. در رگرسیون فرآیند گوسی ابتدا تابع احتمال پیشین مشخص می‌شود. کوواریانس آن با یک ویژگی از هسته تعیین می‌شود. هایپرپارامترهای هسته در طول برازش رگرسیون فرآیند گوسی به‌وسیله حداکثر کردن احتمال شباهت حاشیه‌ای بهینه می‌شوند. هسته‌ها شکل احتمال پسین و احتمال پیشین را تعیین می‌کنند. دودسته از هسته‌ها را به نام هسته ثابت و غیرثابت نام می‌بریم.

در طبقه‌بندی، فرآیند گوسی را برای اهداف طبقه‌بندی پیاده‌سازی می‌کند. فرآیند گوسی را بر روی یک تابع که عملکرد پنهان دارد، قرار می‌دهد، سپس از طریق یک تابع پیوند f بی‌اثر می‌شود تا طبقه‌بندی احتمالی به دست آید. در طبقه‌بندی فرآیند گوسی تابع پیوند، تابع لجستیک انتخاب می‌شود. طبقه‌بندی فرآیند گوسی، از طبقه‌بندی چند کلاسه استفاده می‌کند که آموزش و پیش‌بینی مبتنی بر دو روش، یک در مقابل استراحت یا یک در مقابل یک انجام می‌شود.

۲–۲– فرآیندهای گوسی

فرآیندهای گوسی (GP[1]) یک روش یادگیری با نظارت است که برای حل مشکلات رگرسیون و طبقه‌بندی طراحی‌شده است.

مزایای فرآیندهای گوسی عبارت‌اند از:

۱- استفاده از درون‌یابی برای پیش‌بینی‌ها(درون‌یابی حداقل برای هسته‌ها[2] انجام می‌شود)

۲- پیش‌بینی‌های مبتنی بر فرآیند گوسی به این طریق انجام می‌شود که ابتدا فواصل اطمینان محاسبه می‌شود و سپس بر اساس آن‌ها تصمیم‌گیری می‌شود. یعنی بر اساس فواصل اطمینان مشخص می‌شود که باید در منطقه‌ای که مدنظر ما است، دوباره فرایند پیش‌بینی را انجام دهیم یا خیر.

---

1 Gaussian Processes

2 kernels

۳- هسته‌های مشترکی برای دسته‌ها ارایه می‌دهند اما این امکان وجود دارد که برای هر دسته یک هسته خاص خودش را تعیین کنیم.

معایب فرآیندهای گوسی عبارت‌اند از:

۴- از کل اطلاعات نمونه‌ها(ویژگی‌ها) برای پیش‌بینی استفاده می‌کنند.

۵- فرآیندهای گوسی در فضاهای با ابعاد بالا کارایی خود را از دست می‌دهند، یعنی زمانی که تعداد ویژگی‌ها زیاد شود نمی‌توان از فرایندهای گوسی استفاده کرد.

۲-۳- رگرسیون فرآیند گوسی

رگرسیون فرآیند گوسی(GPR[1]) یعنی این‌که فرآیندهای گوسی (GP)را برای اهداف رگرسیون پیاده‌سازی می‌کند. برای این کار باید ابتدا احتمال پیشین[2] مشخص شود. میانگین احتمال پیشین صفر یا یک عدد ثابت(برای normalize_y=False) یا میانگین داده‌های آموزشی (برای normalize_y=True) فرض می‌شود.

کوواریانس احتمال پیشین با یک ویژگی از هسته مشخص می‌شود. هایپرپارامتر[3]های هسته در طول برازش رگرسیون فرآیند گوسی به‌وسیله حداکثرکردن احتمال شباهت حاشیه‌ای(LML[4]) بهینه می‌شوند. یعنی بیشترین مقدار شباهت حاشیه‌ای را به دست می‌آوریم و از این طریق بهینه‌سازی هم صورت می‌گیرد. ازآنجایی‌که LML ممکن است چندین نقطه بهینه محلی داشته باشد، بهینه‌ساز می‌تواند بارها و بارها با مشخص کردن نقطه بهینه جدید راه‌اندازی شود[5]. اولین اجرا همیشه با شروع از مقادیر اولیه از هایپرپارامتر هسته انجام می‌شود. در اجراهای بعدی به‌طور تصادفی از هایپرپارامترهایی استفاده می‌شود که در محدوده مجاز قرار دارند.

سطح نویز را در هدف[6]، می‌توان از طریق پارامتر آلفا مشخص کرد. که به‌صورت سراسری[7] به صورت اسکالر یا در نقطه داده[8] مشخص می‌شود. باید به این موضوع هم توجه کرد که سطح نویز متوسط نیز

1 Gaussian Process Regression
2 Prior
3 Hyperparameter
4 Log-marginal-likelihood
5 N_restarts_optimizer
6 Target
7 Globally
8 Data point

می‌تواند برای حل مسائل عددی مفید باشد، زیرا به‌عنوان منظم سازی تیخونوف[1] اجرا می‌شود(با اضافه کردن آن به قطر ماتریس هسته).

یک روش جایگزین برای مشخص کردن سطح نویز این است که یک جزء به نام هسته سفید[2]، در هسته قرار دهیم که می‌تواند سطح نویز را از روی‌داده‌ها تخمین بزند که علاوه بر مزایای تخمین به روش استاندارد رگرسیون گوسی، امکان پیش‌بینی بدون برازش احتمال پیشین را فراهم می‌کند (بر اساس احتمال پیشین فرآیندهای گوسی).

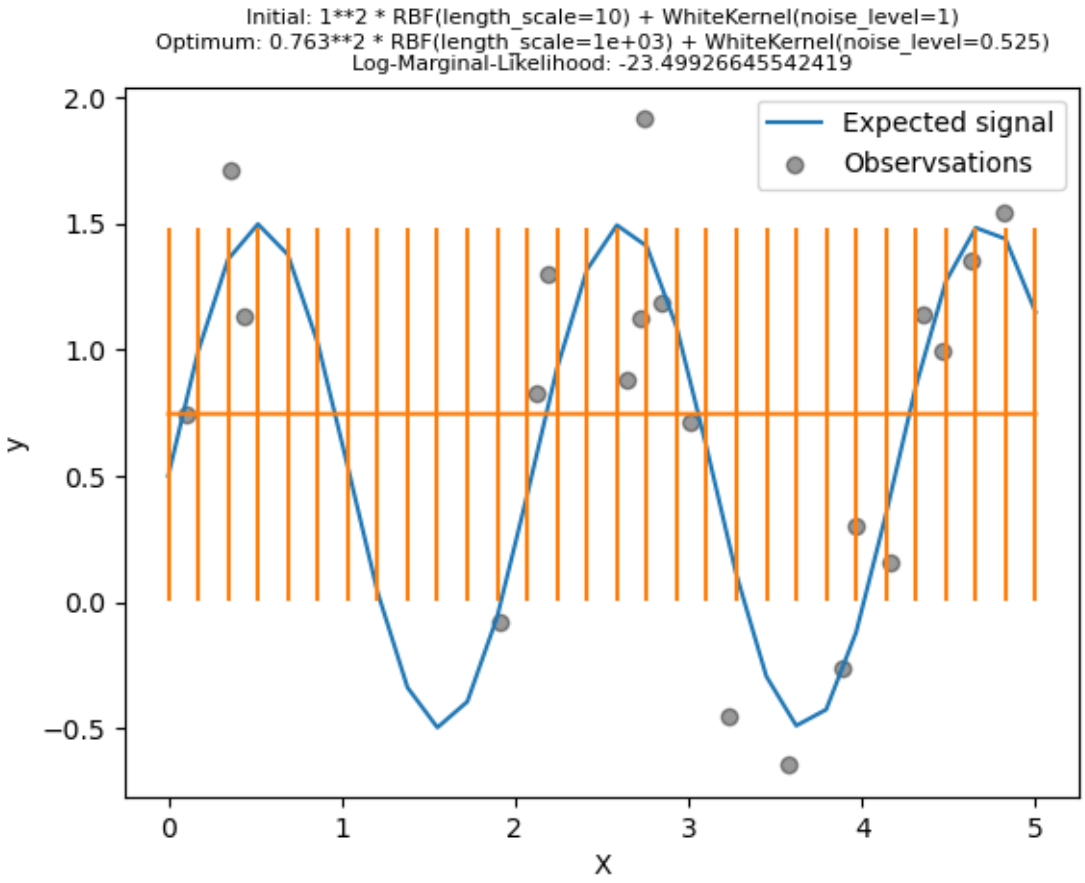
روش sample_y(x)، نمونه‌های گرفته‌شده از رگرسیون فرآیند گوسی (احتمال پیشین یا پسین) را در ورودی‌های داده‌شده ارزیابی می‌کند. یک روش شباهت حاشیه‌ای برحسب تتا[3] را محاسبه می‌کند، که می‌تواند به‌صورت خارجی[4] برای انتخاب‌هایپرپارامترها استفاده شود، به‌عنوان‌مثال، از طریق زنجیره مارکوف مونت کارلو[5].

۲-۴- مثال‌های رگرسیون فرآیند گوسی

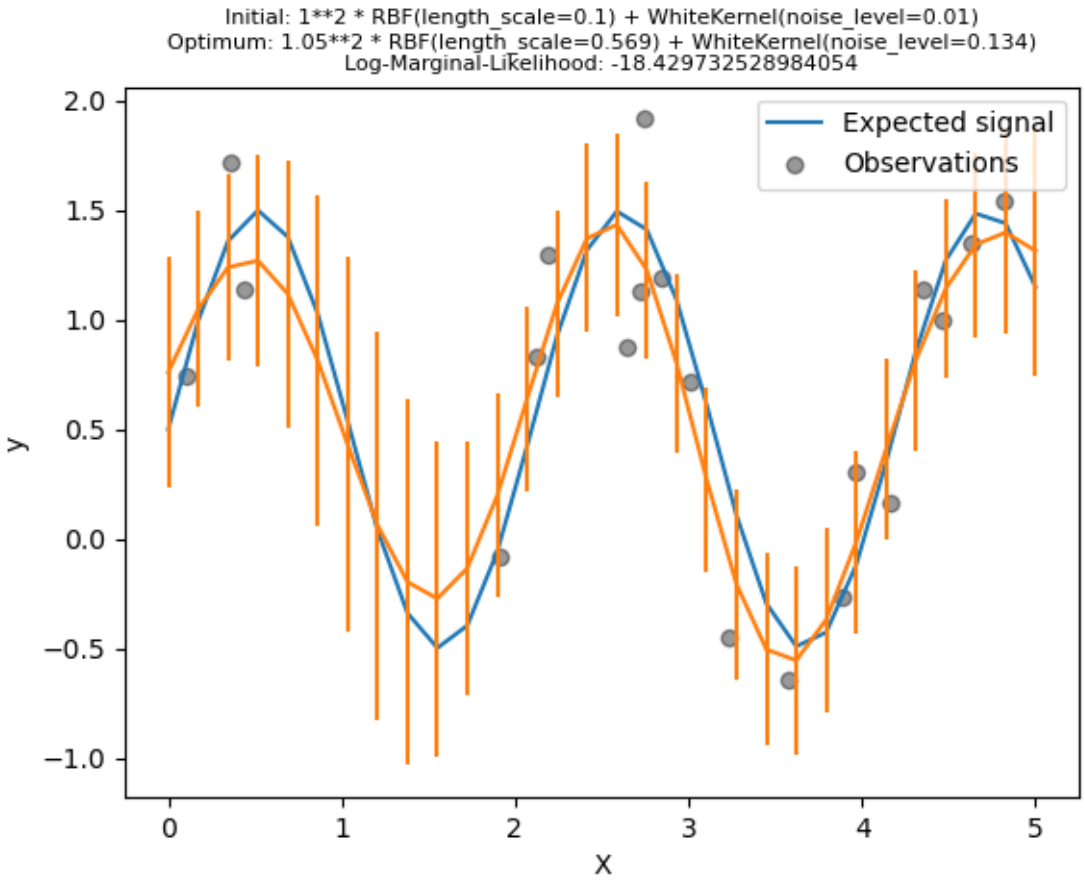
۲-۴-۱- رگرسیون فرآیند گوسی با تخمین سطح نویز:

این مثال توانایی هسته سفید برای تخمین سطح نویز در داده‌ها را نشان می‌دهد. علاوه بر این، اهمیت انتخاب هایپرپارامترهای هسته را نشان می‌دهیم[۱].

1 Tikhonov
2 White Kernel
3 Log_marginal_likelihood(theta)
4 Externally
5 Markov chain Monte Carlo

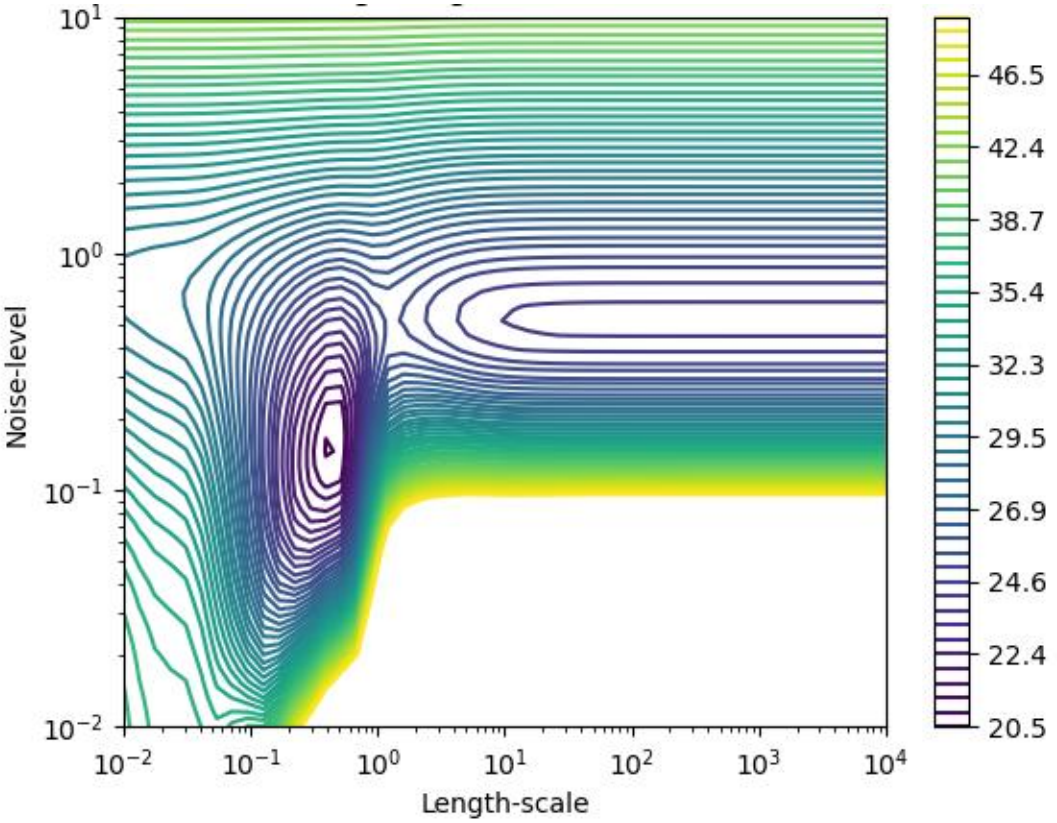


شکل ۲: مدلی با سطح نویز بالا[۱]



شکل ۳: مدلی سطح نویز کوچک[۲]

شکل ۲ مربوط به مدلی با سطح نویز بالا که تمام تغییرات داده تا را با نویز توضیح می‌دهد. شکل ۳ دارای سطح نویز کوچک‌تر و مقیاس طول کوتاه‌تر است، که بیشتر تغییرات را با رابطه عملکردی بدون نویز توضیح می‌دهد. مدل دوم احتمال بیشتری دارد. با این‌حال، بسته به مقدار اولیه برای هایپرپارامترها، بهینه‌سازی مبتنی بر گرادیان نیز ممکن است به راه‌حل پر نویز همگرا شود. بنابراین مهم است که بهینه‌سازی چندین بار برای مقداردهی‌های اولیه مختلف تکرار شود. شکل ۴ احتمال حاشیه‌ای باوجود دو ماکزیمم محلی را نشان می‌دهد.



شکل ۴: لگاریتم احتمال حاشیه‌ای[۳]

۲-۴-۲- مقایسه رگرسیون فرآیند گوسی و رگرسیون ریج هسته

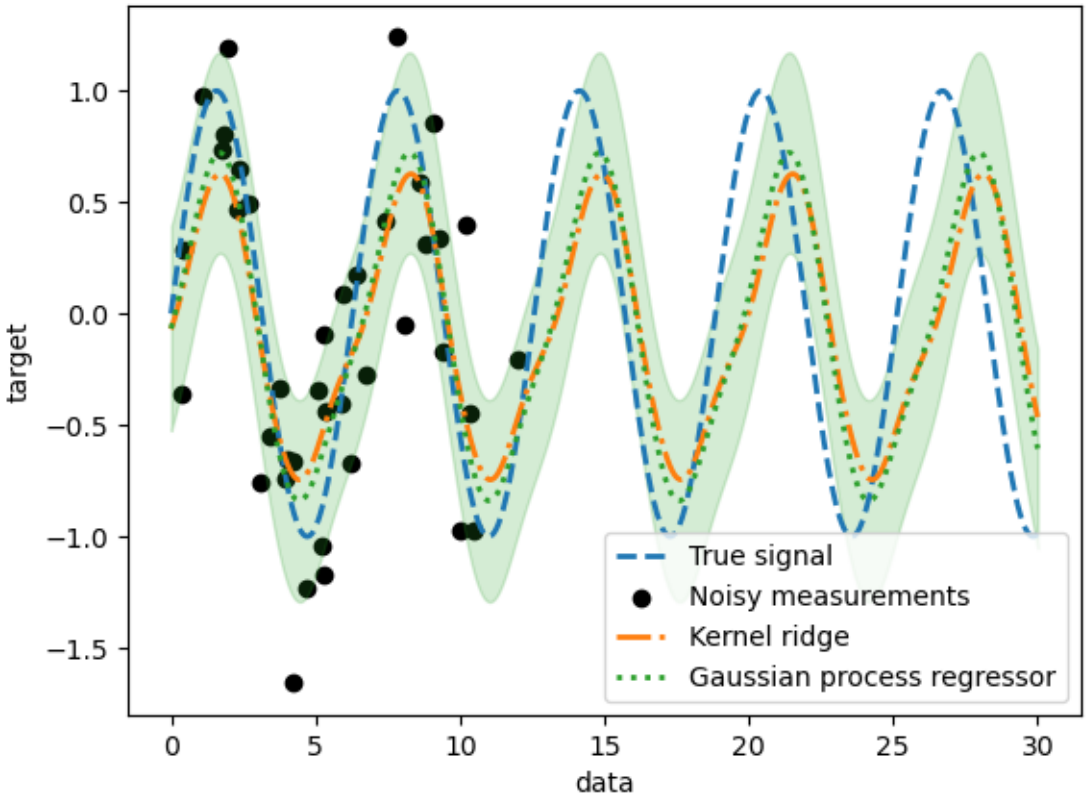
هم رگرسیون ریج هسته (KRR[۱]) و هم رگرسیون فرآیند گوسی یک تابع هدف را با استفاده از قانون هسته[۲] به‌صورت داخلی یاد می‌گیرند. KRR یک تابع خطی را در فضایی که توسط هسته‌ایجادشده است را یاد می‌گیرد که با تابع غیرخطی در فضای اصلی مطابقت دارد. تابع خطی در فضای کرنل بر اساس خطای میانگین مربع[۳] با تنظیم مرز[۴] انتخاب می‌شود. GPR از هسته برای تعریف کوواریانس توزیع احتمال پیشین بر روی توابع هدف استفاده می‌کند و از داده‌های آموزشی مشاهده‌شده برای تعریف یک تابع شباهت[۵] استفاده می‌کند. بر اساس قضیه بیز، یک توزیع پسین[۶](گوسی) بر روی توابع هدف تعریف‌شده است که میانگین آن برای پیش‌بینی استفاده می‌شود.

1 Kernel ridge regression
2 Kernel trick
3 Mean-squared error
4 Ridge
5 Likelihood
6 Posterior distribution

یک تفاوت عمده این است که GPR می‌تواند هایپرپارامترهای هسته را بر اساس افزایش گرادیان بر روی تابع احتمال حاشیه‌ای انتخاب کند، درحالی‌که KRR باید جستجوی شبکه‌ای[1]را بر روی یک تابع هزینه[2](خطای میانگین مربع) اعتبار سنجی کند(به‌وسیله اعتبار سنجی متقابل).

یک تفاوت دیگر این است که GPR یک مدل مولد[3] و احتمالاتی از تابع هدف را می‌آموزد و بنابراین می‌تواند فواصل اطمینان معنی‌دار و نمونه‌های پسین را همراه با پیش‌بینی ارائه دهد درحالی‌که KRR فقط پیش‌بینی ارائه می‌دهد. شکل ۵ هر دو روش را بر روی یک مجموعه داده مصنوعی نشان می‌دهد که از یک تابع هدف سینوسی و نویز شدید تشکیل‌شده است. شکل ۵، مدل KRR و GPR را بر اساس یک هسته سینوسی نمایی[4]، که برای یادگیری توابع متناوب[5] مناسب است، مقایسه می‌کند. هایپرپارامترهای هسته، صاف بودن[6](طول_مقیاس) و تناوب هسته را کنترل می‌کنند. شکل ۵ نشان می‌دهد که هر دو روش مدل‌های منطقی بر تابع هدف را یاد می‌گیرند. GPR به‌درستی دوره تناوب تابع را شناسایی می‌کند(تقریبا$6.28 * 2\pi$)، درحالی‌که KRR دوره تناوب دو برابر را انتخاب می‌کند($4 * \pi$). علاوه بر این، GPR مرزهای اطمینان معقولی را در مورد پیش‌بینی ارائه می‌کند که برای KRR در دسترس نیست. تفاوت عمده بین این دو روش زمان موردنیاز برای برازش و پیش‌بینی است. برازش KRR در اصل سریع‌تر است، اما جستجوی شبکه‌ای برای بهینه‌سازی هایپرپارامترها به‌صورت تصاعدی با افزایش تعداد پارامترها اضافه می‌شود(نفرین ابعاد[7]). بهینه‌سازی مبتنی بر گرادیان در GPR در این مثال زمان‌بر نیست و بنابراین با فضای ابر پارامتر سه‌بعدی بسیار سریع است. زمان پیش‌بینی در هر دو روش مشابه است، بااین‌حال، برای ایجاد واریانس توزیع پیش‌بینی GPR به‌طور قابل‌توجهی به‌وقت بیشتری در مقایسه با پیش‌بینی میانگین نیاز داریم.

1 Grid search
2 Loss function
3 Generative
4 ExpSineSquared
5 Periodic
6 Smoothness
7 Curse of dimensionality



شکل ۵: مقایسه رگرسیون گوسی و رگرسیون ریج هسته [۴]

۲-۴-۲-۱- شبیه‌سازی و تشریح روش ریج هسته و رگرسیون گوسی

در هر دو روش از اصطلاحی به نام قانون هسته استفاده می‌شود تا مدل‌های خود را به‌اندازه کافی با داده‌های آموزشی مطابقت دهند. بااین‌حال، مشکلات یادگیری ماشینی که با این دو روش حل می‌شوند، به‌شدت متفاوت هستند. رگرسیون ریج هسته، تابع هدفی را پیدا می‌کند که تابع میانگین مربعات خطا را به حداقل می‌رساند. رگرسیون فرآیند گوسی به‌جای یافتن یک تابع هدف واحد، از یک رویکرد احتمالی استفاده می‌کند، یعنی یک احتمال پسین با توزیع گوسی بر روی توابع هدف بر اساس قضیه بیز تعریف می‌شود. بنابراین احتمالات پیشین در توابع هدف با یک تابع احتمال که توسط داده‌های آموزشی تعریف‌شده‌اند، ترکیب می‌شوند تا تخمین‌هایی از توزیع‌های پسین ارائه شود. این تفاوت‌ها را با یک مثال نشان داده می‌شود و همچنین بر تنظیم هایپرپارامترهای هسته تمرکز خواهد شد.

تولید مجموعه داده

یک مجموعه داده مصنوعی ایجاد می‌کنیم. فرآیند مولد بردار یک‌بعدی می‌گیرد و سینوس آن را محاسبه می‌کند که دوره‌این سینوس 2π است.

```
import numpy as np

rng = np.random.RandomState(0)
data = np.linspace(0, 30, num=1_000).reshape(-1, 1)
```

```
target = np.sin(data).ravel()
```

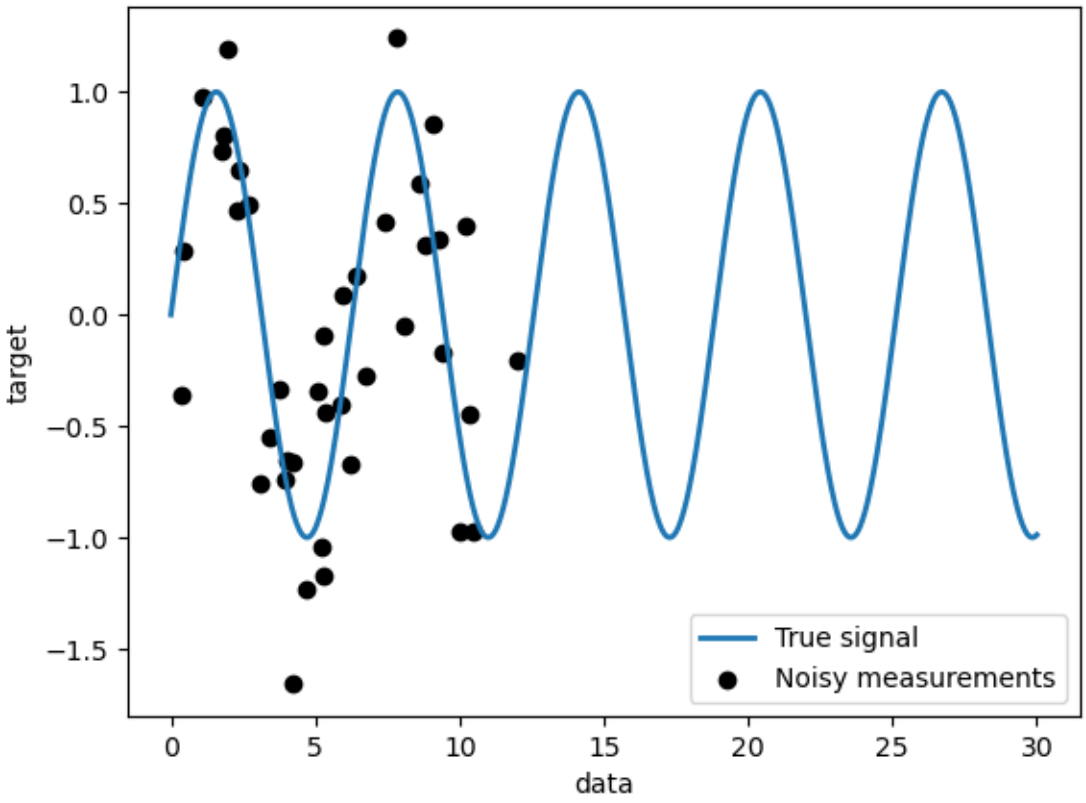
چند چالش اضافه خواهیم کرد:

- ✓ اندازه‌گیری همراه با نویز است.
- ✓ فقط نمونه‌هایی از ابتدای سیگنال موجود است.

```
training_sample_indices = rng.choice(np.arange(0, 400), size=40, replace=False)
training_data = data[training_sample_indices]
training_noisy_target = target[training_sample_indices] + 0.5 * rng.randn(
    len(training_sample_indices)
)
```

سیگنال واقعی و اندازه‌گیری‌های نویز موجود برای آموزش را ترسیم می‌کنیم.

```
import matplotlib.pyplot as plt

plt.plot(data, target, label="True signal", linewidth=2)
plt.scatter(
    training_data,
    training_noisy_target,
    color="black",
    label="Noisy measurements",
)
plt.legend()
plt.xlabel("data")
plt.ylabel("target")
_ = plt.title(
    "Illustration of the true generative process and \n"
    "noisy measurements available during training"
)
```



شکل ۶: فرآیند تولید و اندازه‌گیری‌های پر نویز موجود در طول آموزش[۴]

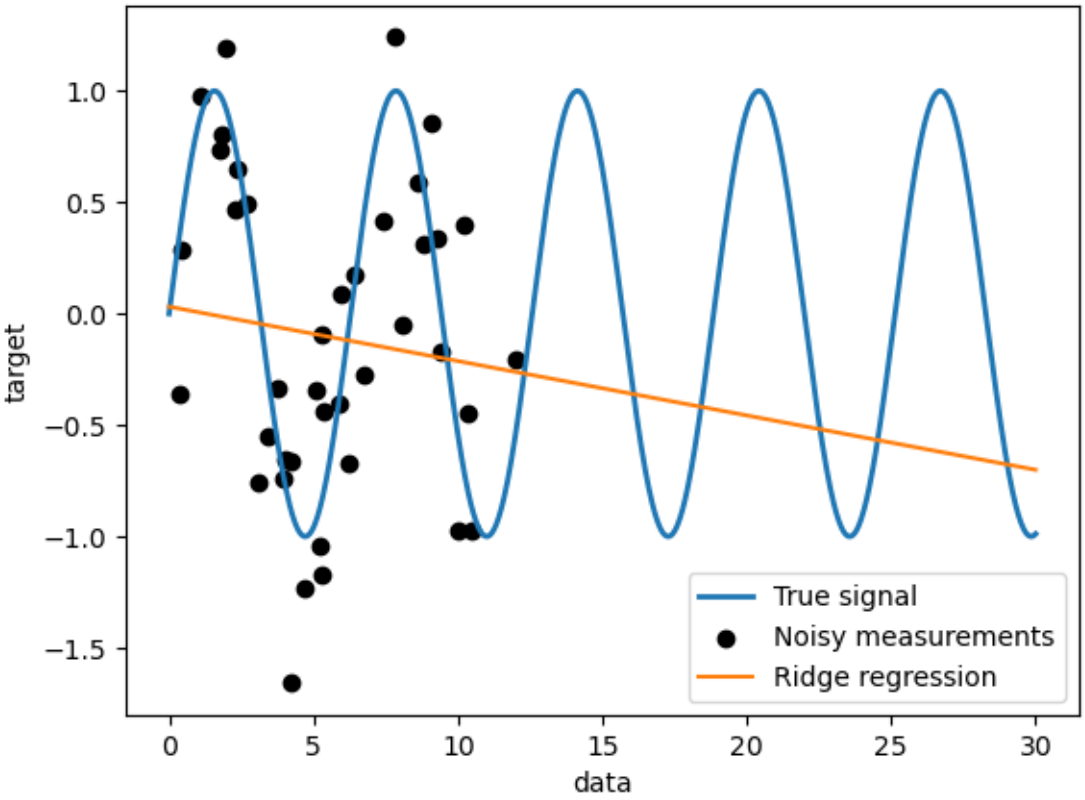
محدودیت‌های یک مدل خطی ساده

ابتدا، می‌خواهیم محدودیت‌های یک مدل خطی را با توجه به مجموعه داده‌هایمان برجسته کنیم. یک ریج را قرار می‌دهیم و پیش‌بینی‌های این مدل را در مجموعه داده خود بررسی می‌کنیم.

```
from sklearn.linear_model import Ridge

ridge = Ridge().fit(training_data, training_noisy_target)

plt.plot(data, target, label="True signal", linewidth=2)
plt.scatter(
    training_data,
    training_noisy_target,
    color="black",
    label="Noisy measurements",
)
plt.plot(data, ridge.predict(data), label="Ridge regression")
plt.legend()
plt.xlabel("data")
plt.ylabel("target")
_ = plt.title("Limitation of a linear model such as ridge")
```

محدودیت یک مدل خطی مانند ریج هسته

چنین رگرسیون‌ای داده‌ها را از بین می‌برد زیرا به‌اندازه کافی معنادار نیست(شکل ۷). از روش‌های انتخاب هسته می‌توان به ریج هسته و فرآیند گوسی اشاره کرد که فعلا از ریج استفاده شد.



شکل ۷: محدودیت مدل خطی ریج[۴]

ریج هسته

ما می‌توانیم مدل خطی قبلی را گویاتر کنیم که به‌وسیله هسته انجام می‌شود. با استفاده از هسته می‌توان به‌جای فضای ویژگی اصلی، یک فضای دیگر قرارداد. یعنی از هسته، برای نگاشت داده‌های اصلی در فضای ویژگی‌های جدیدتر استفاده می‌شود. این فضای جدید با انتخاب هسته تعریف‌شده است. در این مثال، می‌دانیم که فرآیند مولد واقعی یک تابع پریودیک است. می‌توانیم از یک هسته مربع سینوسی نمایی[۱] استفاده کنیم که امکان بازیابی تناوب را فراهم می‌کند. برای استفاده از این روش یک هسته انتخاب می‌شود و سپس به‌وسیله نگاشت هسته، فضای ویژگی را به فضای جدید تبدیل می‌کنیم سپس رگرسیون ریج را اعمال می‌کنیم. در عمل، داده‌ها به‌طور خیلی واضح نگاشت نمی‌شوند. اما یک تابع نقطه‌ای بین نمونه‌ها در فضای ویژگی با ابعاد بالاتر با استفاده از قانون هسته محاسبه می‌شود. بنابراین، از ریج هسته زیر استفاده می‌شود.

```
import time
```

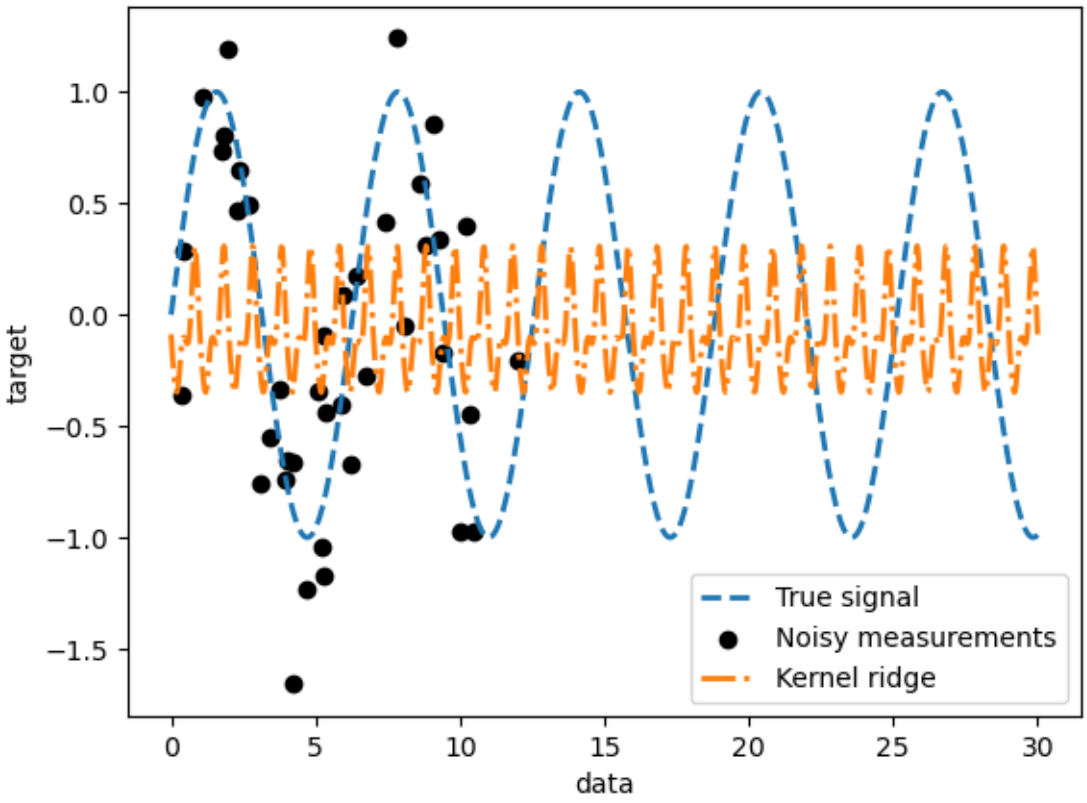
1 ExpSineSquared

```
from sklearn.gaussian_process.kernels import ExpSineSquared
from sklearn.kernel_ridge import KernelRidge

kernel_ridge = KernelRidge(kernel=ExpSineSquared())

start_time = time.time()
kernel_ridge.fit(training_data, training_noisy_target)
print(
    f"Fitting KernelRidge with default kernel: {time.time() - start_time:.3f} seconds"
)
```

خروجی

```
Fitting KernelRidge with default kernel: 0.001 seconds
```

```
plt.plot(data, target, label="True signal", linewidth=2, linestyle="dashed")
plt.scatter(
    training_data,
    training_noisy_target,
    color="black",
    label="Noisy measurements",
)
plt.plot(
    data,
    kernel_ridge.predict(data),
    label="Kernel ridge",
    linewidth=2,
    linestyle="dashdot",
)
plt.legend(loc="lower right")
plt.xlabel("data")
plt.ylabel("target")
_ = plt.title(
    "Kernel ridge regression with an exponential sine squared\n "
    "kernel using default hyperparameters"
)
```



شکل ۸: رگرسیون ریج هسته با هسته مربع سینوسی نمایی با استفاده از هایپرپارامترهای پیش‌فرض[۴]

این مدل دقیق نیست. درواقع، ما پارامترهای هسته را تنظیم نکردیم و در عوض از پارامترهای پیش‌فرض استفاده کردیم اما می‌توانیم آن‌ها را تنظیم کنیم.

```
kernel_ridge.kernel
```

خروجی

```
ExpSineSquared(length_scale=1, periodicity=1)
```

هسته انتخاب‌شده دو پارامتر دارد، مقیاس طول و تناوب. برای مجموعه داده خود، از سینوسی به‌عنوان فرآیند مولد استفاده کردیم که دوره تناوب آن 2π است. مقدار پیش‌فرض پارامتر، فرکانس بالایی را در مدل نشان می‌دهد بنابراین، پارامترهای هسته باید تنظیم شوند. از یک جستجوی تصادفی برای تنظیم پارامترهای مختلف مدل ریج هسته استفاده خواهیم کرد(پارامتر آلفا و پارامترهای هسته).

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.utils.fixes import loguniform

param_distributions = {
    "alpha": loguniform(1e0, 1e3),
    "kernel__length_scale": loguniform(1e-2, 1e2),
    "kernel__periodicity": loguniform(1e0, 1e1),
}
```

```
kernel_ridge_tuned = RandomizedSearchCV(
    kernel_ridge,
    param_distributions=param_distributions,
    n_iter=500,
    random_state=0,
)
start_time = time.time()
kernel_ridge_tuned.fit(training_data, training_noisy_target)
print(f"Time for KernelRidge fitting: {time.time() - start_time:.3f} seconds")
```

خروجی

```
Time for KernelRidge fitting: 5.736 seconds
```

برازش مدل اکنون ازنظر محاسباتی پرهزینه‌تر است زیرا باید چندین ترکیب از هایپرپارامترها را امتحان کنیم. با نگاهی به پارامترهایی که تنظیم کردیم، می‌بینیم که آن‌ها با پارامترهای پیش‌فرض متفاوت هستند. همچنین می‌بینیم که تناوب به مقدار موردنظر نزدیک‌تر است. اکنون می‌توانیم پیش‌بینی‌های هسته تنظیم شده را بررسی کنیم.

```
start_time = time.time()
predictions_kr = kernel_ridge_tuned.predict(data)
print(f"Time for KernelRidge predict: {time.time() - start_time:.3f} seconds")
```

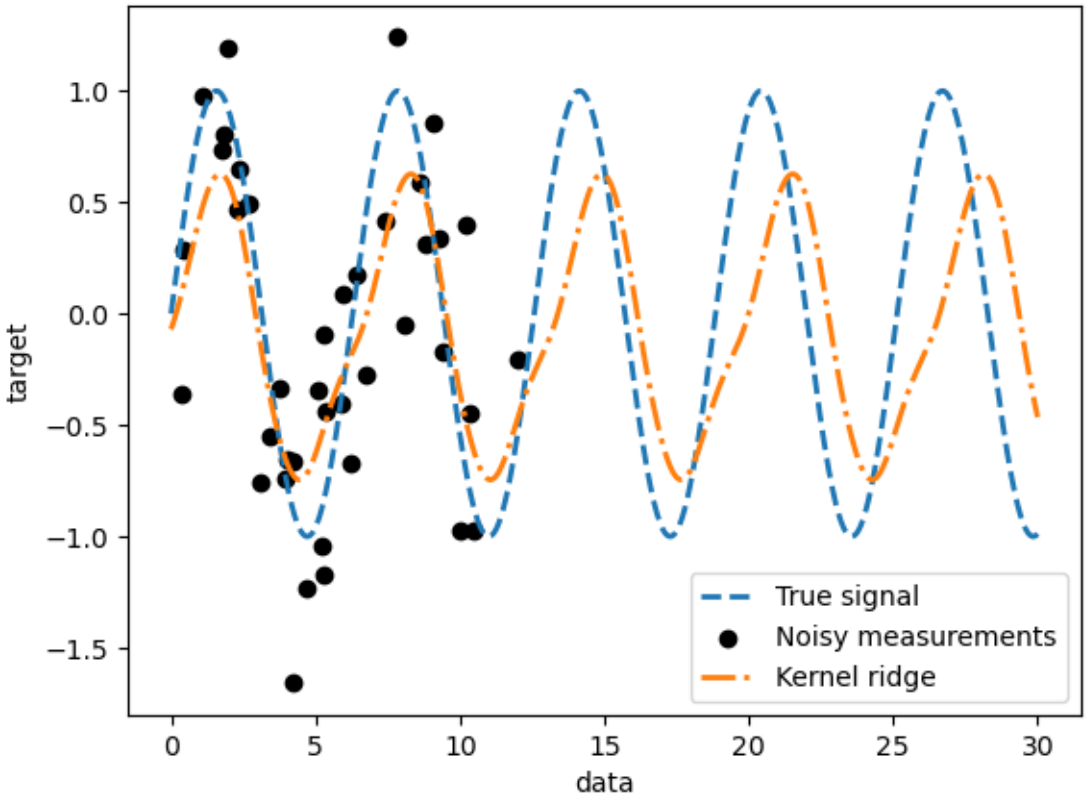
خروجی

```
Time for KernelRidge predict: 0.003 seconds
```

```
plt.plot(data, target, label="True signal", linewidth=2, linestyle="dashed")
plt.scatter(
    training_data,
    training_noisy_target,
    color="black",
    label="Noisy measurements",
)
plt.plot(
    data,
    predictions_kr,
    label="Kernel ridge",
    linewidth=2,
    linestyle="dashdot",
)
plt.legend(loc="lower right")
plt.xlabel("data")
plt.ylabel("target")
_ = plt.title(
```

```
    "Kernel ridge regression with an exponential sine squared\n "
    "kernel using tuned hyperparameters"
)
```

به‌این‌ترتیب مدل بسیار دقیق‌تری دریافت می‌کنیم(شکل ۹). اما هنوز برخی از خطاها را عمدتاً به دلیل نویز اضافه‌شده به مجموعه داده مشاهده می‌کنیم.



شکل ۹: رگرسیون ریج هسته با یک هسته مربع سینوسی نمایی با استفاده از هایپرپارامترهای تنظیم‌شده[۴]

رگرسیون فرآیند گوسی

اکنون، از یک GPR برای فیت کردن همان مجموعه داده استفاده خواهیم کرد. هنگام آموزش فرآیند گوسی، هایپرپارامترهای هسته بهینه می‌شوند. در این قسمت یک هسته، کمی پیچیده‌تر از روش رگرسیون ریج هسته ایجاد می‌کنیم. درواقع یک هسته سفید اضافه می‌شود که برای تخمین نویز در مجموعه داده استفاده مورداستفاده قرار می‌گیرد.

```
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import WhiteKernel

kernel = 1.0 * ExpSineSquared(1.0, 5.0, periodicity_bounds=(1e-2, 1e1)) + WhiteKernel(
    1e-1
)
gaussian_process = GaussianProcessRegressor(kernel=kernel)
start_time = time.time()
```

```
gaussian_process.fit(training_data, training_noisy_target)
print(
    f"Time for GaussianProcessRegressor fitting: {time.time() - start_time:.3f} seconds"
)
```

خروجی

```
Time for GaussianProcessRegressor fitting: 0.058 seconds
```

هزینه محاسباتی آموزش یک فرآیند گوسی بسیار کمتر از ریج هسته است. چون در روش ریج هسته از جستجوی تصادفی استفاده می‌کند. ما می‌توانیم پارامترهایی را که محاسبه کرده‌ایم بررسی کنیم.

```
gaussian_process.kernel_
```

خروجی

```
0.675**2 * ExpSineSquared(length_scale=1.34, periodicity=6.57) +
WhiteKernel(noise_level=0.182)
```

درواقع، می‌بینیم که پارامترها بهینه‌شده‌اند. دوره‌ای نزدیک به مقدار نظری 2π پیدا کردیم. اکنون می‌توانیم پیش‌بینی‌های مدل را بررسی کنیم.

```
start_time = time.time()
mean_predictions_gpr, std_predictions_gpr = gaussian_process.predict(
    data,
    return_std=True,
)
print(
    f"Time for GaussianProcessRegressor predict: {time.time() - start_time:.3f} seconds"
)
```

خروجی

```
Time for GaussianProcessRegressor predict: 0.004 seconds
```

و در پایان به مقایسه بین رگرسیون گوسی و ریج هسته می‌پردازیم.

```
plt.plot(data, target, label="True signal", linewidth=2, linestyle="dashed")
plt.scatter(
    training_data,
    training_noisy_target,
    color="black",
    label="Noisy measurements",
)
# Plot the predictions of the kernel ridge
plt.plot(
    data,
    predictions_kr,
```

```
    label="Kernel ridge",
    linewidth=2,
    linestyle="dashdot",
)
# Plot the predictions of the gaussian process regressor
plt.plot(
    data,
    mean_predictions_gpr,
    label="Gaussian process regressor",
    linewidth=2,
    linestyle="dotted",
)
plt.fill_between(
    data.ravel(),
    mean_predictions_gpr - std_predictions_gpr,
    mean_predictions_gpr + std_predictions_gpr,
    color="tab:green",
    alpha=0.2,
)
plt.legend(loc="lower right")
plt.xlabel("data")
plt.ylabel("target")

_ = plt.title("Comparison between kernel ridge and gaussian process regressor")
```

با توجه به شکل ۵ مشاهده می‌کنیم که نتایج رگرسیون ریج هسته و رگرسیون فرآیند گوسی نزدیک هستند. در رگرسیون فرآیند گوسی اطلاعات عدم قطعیتی ارایه می‌شود که در روش رگرسیون ریج هسته دسترس نیست. با توجه به فرمول‌بندی احتمالی توابع هدف، خروجی فرآیند گوسی می‌تواند انحراف استاندارد (یا کوواریانس) همراه با میانگین پیش‌بینی‌شده توابع هدف باشد[۳].

در این مثال فقط شروع سیگنال را به‌عنوان یک مجموعه آموزشی ارائه کردیم. استفاده از یک هسته متناوب، مدل را مجبور می‌کند تا الگوی موجود در مجموعه آموزشی را تکرار کند. با استفاده از اطلاعات هسته و به‌کارگیری هر دو مدل برای پیش‌بینی(برون‌یابی[۱])، مشاهده می‌کنیم که مدل‌ها به پیش‌بینی الگوی سینوسی ادامه می‌دهند. فرآیند گوسی اجازه می‌دهد تا هسته‌ها را باهم ترکیب کنیم. بنابراین، می‌توانیم هسته مربعی سینوسی نمایی را با یک هسته تابعی بر مبنای شعاعی[۲] ترکیب کنیم.

```
from sklearn.gaussian_process.kernels import RBF

kernel = 1.0 * ExpSineSquared(1.0, 5.0, periodicity_bounds=(1e-2, 1e1)) * RBF(
```

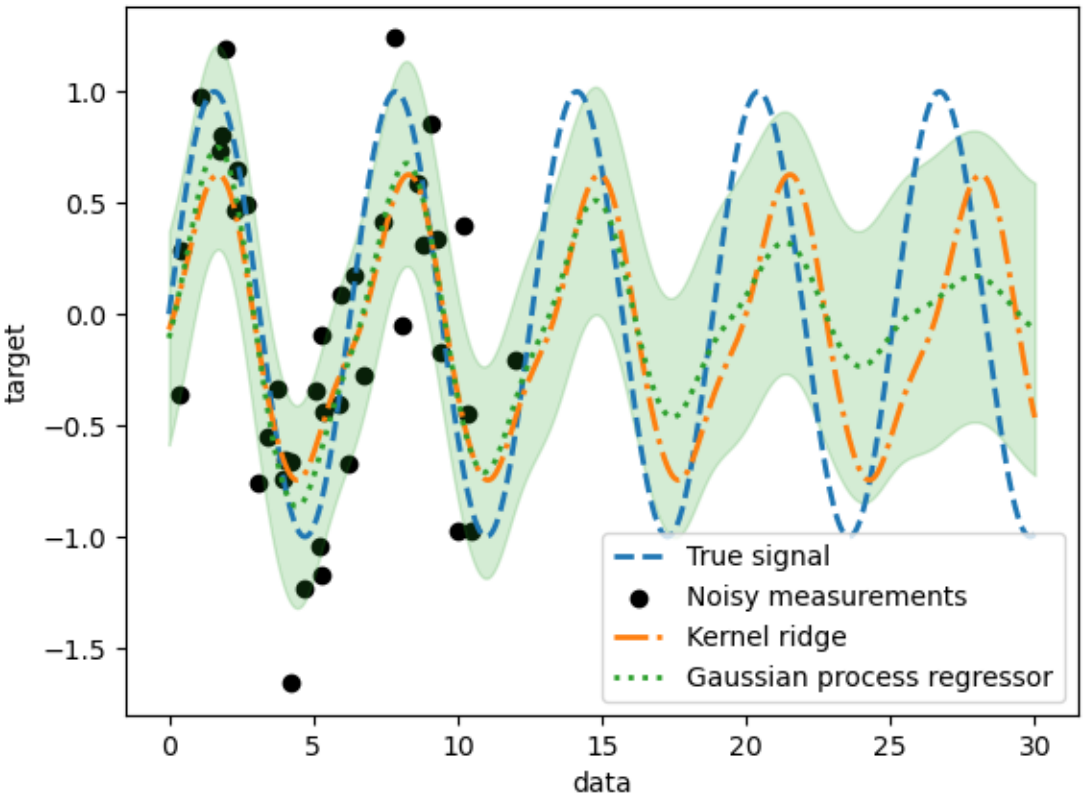
1 Extrapolate

2 Radial basis function kernel

```
    length_scale=15, length_scale_bounds="fixed"
) + WhiteKernel(1e-1)
gaussian_process = GaussianProcessRegressor(kernel=kernel)
gaussian_process.fit(training_data, training_noisy_target)
mean_predictions_gpr, std_predictions_gpr = gaussian_process.predict(
    data,
    return_std=True,
)
plt.plot(data, target, label="True signal", linewidth=2, linestyle="dashed")
plt.scatter(
    training_data,
    training_noisy_target,
    color="black",
    label="Noisy measurements",
)
# Plot the predictions of the kernel ridge
plt.plot(
    data,
    predictions_kr,
    label="Kernel ridge",
    linewidth=2,
    linestyle="dashdot",
)
# Plot the predictions of the gaussian process regressor
plt.plot(
    data,
    mean_predictions_gpr,
    label="Gaussian process regressor",
    linewidth=2,
    linestyle="dotted",
)
plt.fill_between(
    data.ravel(),
    mean_predictions_gpr - std_predictions_gpr,
    mean_predictions_gpr + std_predictions_gpr,
    color="tab:green",
    alpha=0.2,
)
plt.legend(loc="lower right")
plt.xlabel("data")
plt.ylabel("target")
_ = plt.title("Effect of using a radial basis function kernel")
```

تأثیر استفاده از هسته تابع پایه شعاعی

استفاده از یک هسته تابع پایه شعاعی(RBF[1]) اثر تناوب را زمانی که هیچ نمونه‌ای در آموزش موجود نباشد کاهش می‌دهد. همان‌طور که نمونه‌های آزمایشی از نمونه‌های آموزشی دورتر می‌شوند، پیش‌بینی‌ها به سمت میانگین آن‌ها همگرا می‌شوند و انحراف معیار آن‌ها نیز افزایش می‌یابد(شکل ۱۰).



شکل ۱۰: تأثیر استفاده از هسته تابع پایه شعاعی[۵]

۲-۵- طبقه‌بندی فرآیند گوسی

در طبقه‌بندی فرآیند گوسی (GPC[2])، GP را برای اهداف طبقه‌بندی پیاده‌سازی می‌کند. GP را بر روی یک تابع که عملکرد پنهان دارد، اعمال می‌کند سپس از طریق یک تابع پیوند f بی‌اثر می‌شود تا طبقه‌بندی احتمالی به دست آید. تابع پنهان f به‌اصطلاح یک تابع مزاحم[3] است که مقادیر آن مشاهده نمی‌شود. هدف این است که فرمول مناسب مدل اجرا شود و در طول پیش‌بینی حذف می‌شود (ادغام می‌شود). در GPC تابع لجستیک به‌عنوان تابع پیوند انتخاب می‌شود، که برای آن انتگرال را نمی‌توان به‌صورت تحلیلی محاسبه کرد، اما در حالت باینری به‌راحتی تقریب زده می‌شود.

1 Radial-basis function
2 Gaussian Process Classification
3 Nuisance function

برخلاف رگرسیون، احتمال پسین برای تابعی که عملکرد پنهان دارد، گوسی نیست زیرا احتمال گوسی برای برچسب‌های کلاس گسسته مناسب نیست. که به‌جای آن، یک احتمال که غیر گوسی است (مربوط به تابع پیوند لجستیک) استفاده می‌شود. در احتمال پسین برای طبقه‌بندی فرآیند گوسی، غیر گوسی را با گوسی، بر اساس تقریب لاپلاس تقریب می‌زنند[۶].

طبقه‌بندی فرآیند گوسی، از طبقه‌بندی چند کلاسه استفاده می‌کند که آموزش و پیش‌بینی مبتنی بر دو روش، یک در مقابل استراحت[1]یا یک در مقابل یک[2] انجام می‌شود. در روش یک در مقابل استراحت، یک طبقه‌بندی کننده گوسی باینری برای هر کلاس تعبیه‌شده است که برای جدا کردن یک کلاس از بقیه کلاس‌ها آموزش داده‌شده است. در روش یک در مقابل یک، یک طبقه‌بندی کننده گوسی باینری برای هر جفت کلاس تعبیه‌شده است که برای جدا کردن دو کلاس از بقیه کلاس‌ها آموزش داده‌شده است. این پیش‌بینی باینری در کاربردهای چند کلاسه ترکیب می‌شوند(طبقه‌بندی فرآیند گوسی بر اساس تقریب لاپلاس انجام می‌شود)[۷].

طبقه‌بندی فرآیند گوسی به روش یک در مقابل یک، ممکن است ازنظر محاسباتی ارزان‌تر باشد، زیرا مسائلی را حل می‌کند که فقط شامل یک زیرمجموعه از کل مجموعه آموزشی است. باید توجه کرد که روش یک در مقابل یک، از پیش‌بینی‌های احتمالی پشتیبانی نمی‌کند و در پیش‌بینی‌های ساده استفاده می‌شود. علاوه بر این، باید توجه کرد که GPC یک تقریب لاپلاس چند کلاسه واقعی را به‌صورت داخلی پیاده‌سازی نمی‌کند، اما همان‌طور که در بالا بحث شد مبتنی بر چندین طبقه‌بندی باینری است، که با استفاده از روش یک در مقابل استراحت، یا یک در مقابل یک، ترکیب می‌شوند. اما در کل زمان محاسبه پیش‌بینی‌ها با فرآیند گوسی بیشتر است[۸].

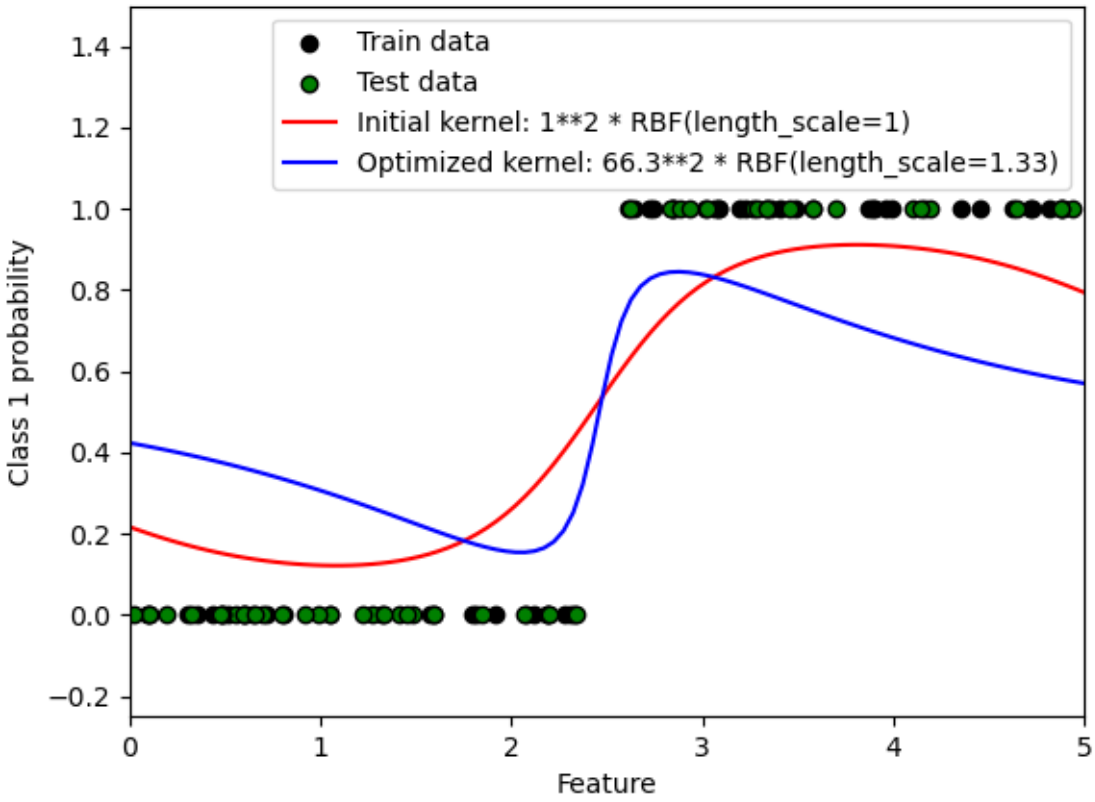
۲-۶- مثال‌های طبقه‌بندی فرآیند گوسی

۲-۶-۱- پیش‌بینی‌های احتمالی با طبقه‌بندی فرآیند گوسی

این مثال، احتمال پیش‌بینی‌شده GPC را برای یک هسته RBF با انتخاب‌های مختلف هایپرپارامترها نشان می‌دهد.شکل ۱۱ احتمال پیش‌بینی‌شده GPC، که با هایپرپارامترهای دلخواه انتخاب‌شده است را نمایش می‌دهد و سپس در شکل ۱۲ با هایپرپارامترهای مربوط، حداکثر لگاریتم احتمال حاشیه‌ای محاسبه شده است.

1 One-versus-rest

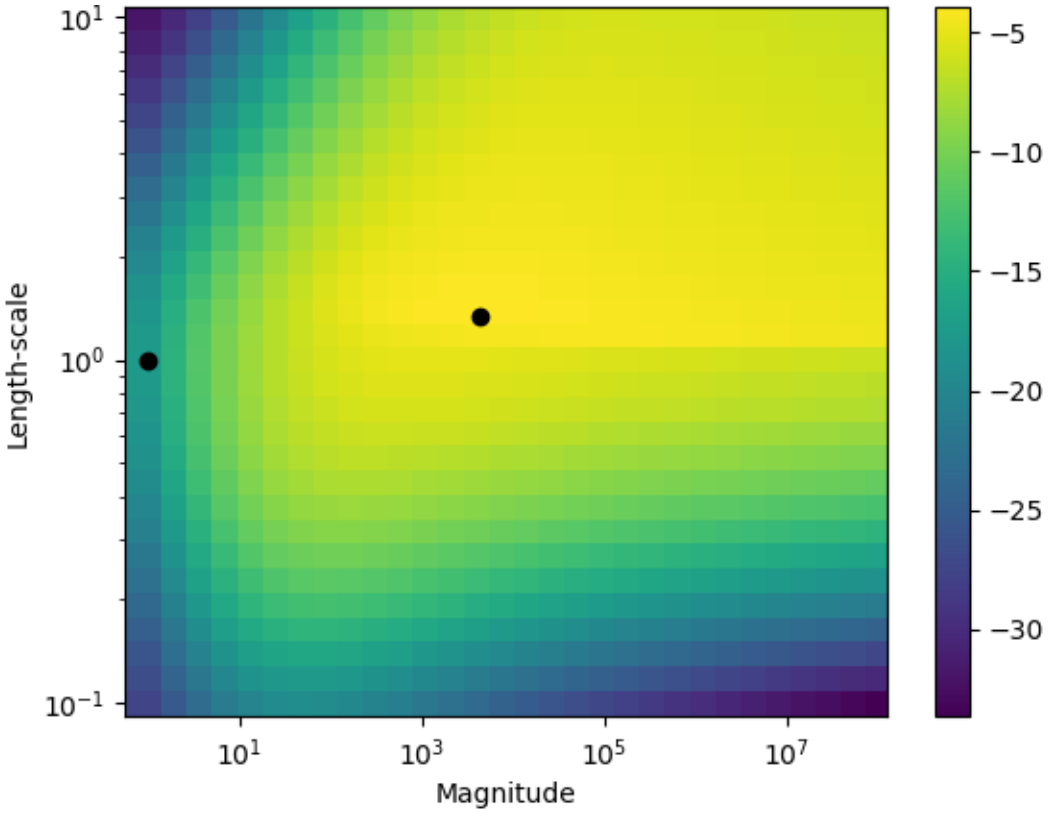
2 One-versus-one



شکل ۱۱ :احتمال پیش‌بینی‌شده GPC[5]

هایپرپارامترهای انتخاب‌شده با استفاده از بهینه‌سازی LML مقدار بزرگ‌تری دارند، اما با توجه به اینکه مقداری از داده‌های آزمایش از بین رفته‌اند، کمی بدتر عمل می‌کنند. شکل ۱۱نشان می‌دهد که یک تغییر شدید در مرزهای کلاس وجود دارد (که خوب است)، اما احتمالات نزدیک به ۰/۵ دور از مرزهای کلاس را پیش‌بینی کرده‌اند (که بد است) این اثر نامطلوب به دلیل تقریب لاپلاس که توسط GPC استفاده می‌شود، ایجادشده است.

شکل ۱۲، احتمال حاشیه‌ای را برای انتخاب‌های مختلف هایپرپارامترهای هسته نشان می‌دهد، و هایپرپارامترهای استفاده‌شده دررا با نقاط سیاه برجسته می‌کند.



شکل ۱۲: احتمال حاشیه‌ای با هایپرپارامترهای مختلف هسته(هایپرپارامترهای شکل۱۰ با نقاط سیاه برجسته شده است)[۵]

۲-۶-۲- طبقه‌بندی فرآیند گوسی روی مجموعه داده XOR

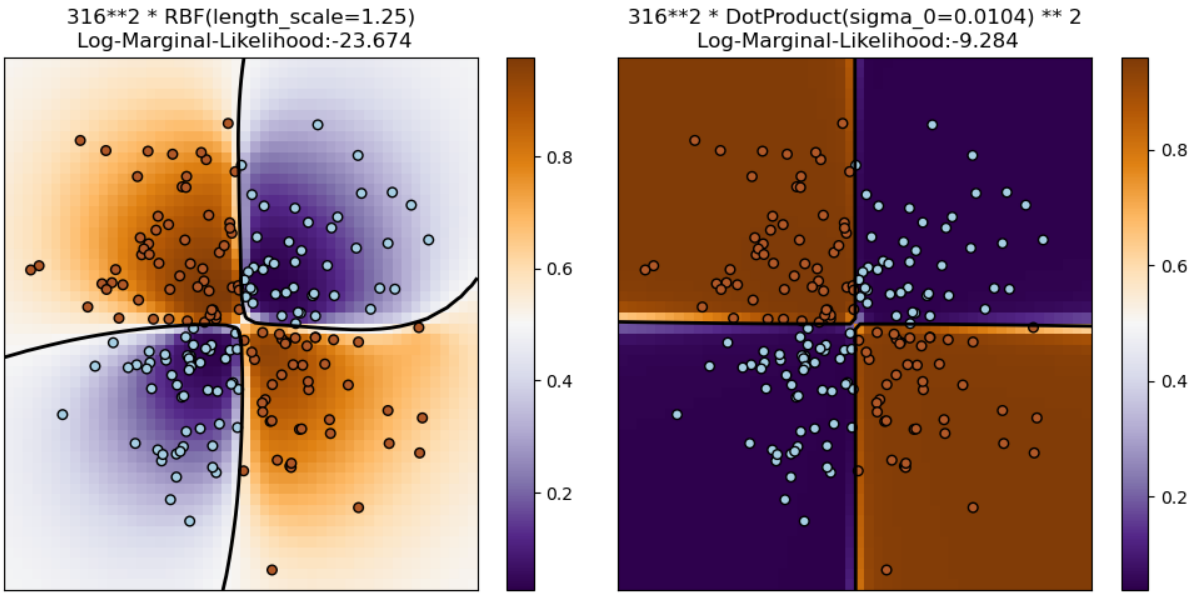
این مثال، روش GPC روی‌داده‌های XOR را نشان می‌دهد. یک هسته ثابت، که از هر سو خواص برابر دارد یعنی همسانگرد[1] است و یک هسته غیر ساکن[2] باهم مقایسه می‌شوند. در این مجموعه داده خاص، هسته غیر ساکن نتایج بهتری به دست می‌آورد زیرا مرزهای کلاس خطی هستند و با محورهای مختصات منطبق هستند. اما در عمل، هسته‌های ثابت مانند RBF اغلب نتایج بهتری به دست می‌آورند.

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

from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF, DotProduct


xx, yy = np.meshgrid(np.linspace(-3, 3, 50), np.linspace(-3, 3, 50))
rng = np.random.RandomState(0)
X = rng.randn(200, 2)
Y = np.logical_xor(X[:, 0] > 0, X[:, 1] > 0)
```

1 Isotropic

2 DotProduct

```
# fit the model
plt.figure(figsize=(10, 5))
kernels = [1.0 * RBF(length_scale=1.0), 1.0 * DotProduct(sigma_0=1.0) ** 2]
for i, kernel in enumerate(kernels):
    clf = GaussianProcessClassifier(kernel=kernel, warm_start=True).fit(X, Y)

    # plot the decision function for each datapoint on the grid
    Z = clf.predict_proba(np.vstack((xx.ravel(), yy.ravel())).T)[:, 1]
    Z = Z.reshape(xx.shape)

    plt.subplot(1, 2, i + 1)
    image = plt.imshow(
        Z,
        interpolation="nearest",
        extent=(xx.min(), xx.max(), yy.min(), yy.max()),
        aspect="auto",
        origin="lower",
        cmap=plt.cm.PuOr_r,
    )
    contours = plt.contour(xx, yy, Z, levels=[0.5], linewidths=2, colors=["k"])
    plt.scatter(X[:, 0], X[:, 1], s=30, c=Y, cmap=plt.cm.Paired, edgecolors=(0, 0, 0))
    plt.xticks(())
    plt.yticks(())
    plt.axis([-3, 3, -3, 3])
    plt.colorbar(image)
    plt.title(
        "%s\n Log-Marginal-Likelihood:%.3f"
        % (clf.kernel_, clf.log_marginal_likelihood(clf.kernel_.theta)),
        fontsize=12,
    )

plt.tight_layout()
plt.show()
```



شکل ۱۳: طبقه‌بندی فرآیند گوسی در مجموعه داده XOR [10]

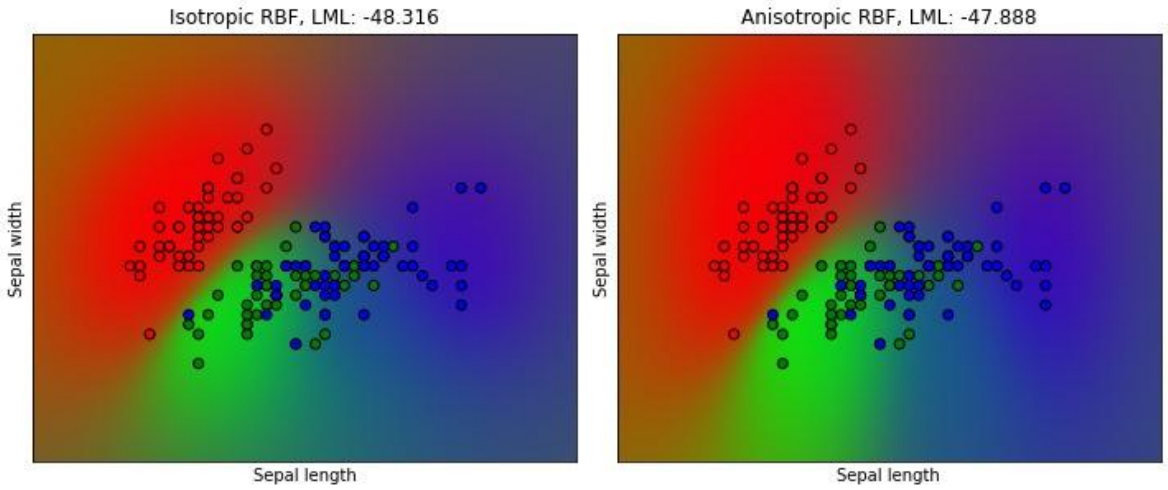
۲-۶-۳- طبقه‌بندی فرآیند گوسی در مجموعه داده عنبیه

این مثال احتمال پیش‌بینی‌شده GPC را برای یک هسته RBF همسانگرد و ناهمسانگرد[1] در یک نسخه دوبعدی برای مجموعه داده عنبیه نشان می‌دهد(شکل ۱۴). که کاربرد GPC برای طبقه‌بندی غیر باینری است. هسته RBF ناهمسانگرد با تخصیص مقیاس‌های طولی مختلف به دو بعد ویژگی، احتمال لگاریتم حاشیه‌ای بالاتری را به دست می‌آورد[۱۱].

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF

# import some data to play with
iris = datasets.load_iris()
X = iris.data[:, :2]  # we only take the first two features.
y = np.array(iris.target, dtype=int)

h = 0.02  # step size in the mesh

kernel = 1.0 * RBF([1.0])
gpc_rbf_isotropic = GaussianProcessClassifier(kernel=kernel).fit(X, y)
kernel = 1.0 * RBF([1.0, 1.0])
gpc_rbf_anisotropic = GaussianProcessClassifier(kernel=kernel).fit(X, y)
```

1 Anisotropic

```
# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

titles = ["Isotropic RBF", "Anisotropic RBF"]
plt.figure(figsize=(10, 5))
for i, clf in enumerate((gpc_rbf_isotropic, gpc_rbf_anisotropic)):
    # Plot the predicted probabilities. For that, we will assign a color to
    # each point in the mesh [x_min, m_max]x[y_min, y_max].
    plt.subplot(1, 2, i + 1)

    Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape((xx.shape[0], xx.shape[1], 3))
    plt.imshow(Z, extent=(x_min, x_max, y_min, y_max), origin="lower")

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=np.array(["r", "g", "b"])[y], edgecolors=(0, 0, 0))
    plt.xlabel("Sepal length")
    plt.ylabel("Sepal width")
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks(())
    plt.yticks(())
    plt.title(
        "%s, LML: %.3f" % (titles[i], clf.log_marginal_likelihood(clf.kernel_.theta))
    )

plt.tight_layout()
plt.show()
```



شکل ۱۴: احتمال پیش‌بینی‌شده GPC برای هسته RBF همسانگرد و ناهمسانگرد[11]

۲-۷- هسته در فرآیندهای گوسی

هسته‌ها (که توابع کوواریانس نیز نامیده می‌شوند) یک جزء مهم فرآیندهای گوسی هستند که شکل احتمال پسین و احتمال پیشین فرآیند گوسی را تعیین می‌کنند. که با استفاده از دونقطه از دیتا، با این فرض که نقاط داده مشابه، باید مقادیر هدف مشابهی داشته باشند، عملیات کدگذاری را روی دیتا انجام می‌دهند. دودسته از هسته‌ها را می‌توان نام برد، هسته‌های ثابت و غیرثابت. هسته‌های ثابت فقط به فاصله دونقطه داده وابسته هستند و به مقادیر مطلق آن‌ها بستگی ندارند($k(x_i, x_j)=d(k(x_i, x_j))$) و بنابراین نسبت به انتقال در فضای ورودی ثابت هستند، درحالی‌که هسته‌های غیرثابت به مقادیر داده نیز بستگی دارند. هسته‌های ثابت را می‌توان به هسته‌های همسانگرد و ناهمسانگرد تقسیم کرد، که در آن هسته‌های همسانگرد نیز نسبت به چرخش در فضای ورودی ثابت هستند[۱۲].

۲-۷-۱- هسته فرآیند گوسی API

کاربرد اصلی کرنل، محاسبه کوواریانس فرآیند گوسی بین نقاط دیتا است. برای این کار می‌توان روش __call__ را برای هر هسته فراخوانی کرد. این روش می‌تواند برای محاسبه کوواریانس خودکار[1] همه جفت نقاط داده در یک آرایه دوبعدی x یا کوواریانس متقاطع همه ترکیبات نقاط یک آرایه دوبعدی X با نقاط داده یک آرایه دوبعدی Y استفاده شود. معادله k(X) = K(X, Y=X) برای همه هسته‌های k صادق است (به‌جز هسته سفید).اگر فقط از قطر کوواریانس خودکار استفاده شود، می‌توان روش diag را برای هسته فراخوانی کرد که ازنظر محاسباتی کارآمدتر از فراخوانی __call__ است((np.diag(k(X, X)) = k.diag (X)).

1 Auto-covariance

هسته‌ها توسط یک بردار از هایپرپارامترها نوشته می‌شوند. هایپرپارامترها برای مثال می‌توانند مقیاس طول یا تناوب یک هسته را کنترل کنند. همه هسته‌ها از طریق تنظیم eval_gradient=True در روش __call__، از محاسبه گرادیان‌های تحلیلی کوواریانس خودکار هسته با توجه به $\log\theta$ محاسبه می‌شوند[۱۳].

این گرادیان توسط فرآیند گوسی (هم رگرسیون و هم طبقه‌بندی کننده) در محاسبه گرادیان احتمال لگاریتم حاشیه‌ای استفاده می‌شود. برای هر هایپرپارامتر، مقدار اولیه و مرزها باید هنگام ایجاد یک نمونه از هسته مشخص شوند. مقدار فعلی را می‌توان از طریق ویژگی θ دریافت و تنظیم کرد. علاوه بر این، محدوده هایپرپارامترها توسط مرزهای هسته قابل‌دسترس هستند. مشخصات هر هایپرپارامتر به‌صورت نمونه‌ای در هسته مربوطه ذخیره می‌شود. هسته‌ای که از یک هایپرپارامتر بانام x استفاده می‌کند باید دارای ویژگی‌های self.x و self.x_bounds باشد.

کلاس پایه برای همه هسته‌ها Kernel است. Kernel یک رابط مشابه به‌عنوان تخمین زننده، پیاده‌سازی می‌کند و متدهای ()get_params، ()set_params و ()clone را ارائه می‌کند. این امر موجب می‌شود تا مقادیر کرنل را از طریق تخمین زننده‌های متا[1] مانند Pipeline یا GridSearch نیز تنظیم کند. باید توجه کرد که به دلیل ساختار تودرتوی هسته‌ها (با اعمال عملگرهای هسته)، نام پارامترهای هسته ممکن است نسبتاً پیچیده شود.

مثال[۶]:

```
>>> from sklearn.gaussian_process.kernels import ConstantKernel, RBF
>>> kernel = ConstantKernel(constant_value=1.0, constant_value_bounds=(0.0, 10.0)) *
RBF(length_scale=0.5, length_scale_bounds=(0.0, 10.0)) + RBF(length_scale=2.0,
length_scale_bounds=(0.0, 10.0))
>>> for hyperparameter in kernel.hyperparameters: print(hyperparameter)
Hyperparameter(name='k1__k1__constant_value', value_type='numeric', bounds=array([[
0., 10.]]), n_elements=1, fixed=False)
Hyperparameter(name='k1__k2__length_scale', value_type='numeric', bounds=array([[ 0.,
10.]]), n_elements=1, fixed=False)
Hyperparameter(name='k2__length_scale', value_type='numeric', bounds=array([[ 0.,
10.]]), n_elements=1, fixed=False)
>>> params = kernel.get_params()
>>> for key in sorted(params): print("%s : %s" % (key, params[key]))
k1 : 1**2 * RBF(length_scale=0.5)
k1__k1 : 1**2
k1__k1__constant_value : 1.0
k1__k1__constant_value_bounds : (0.0, 10.0)
```

1 Meta-estimators

```
k1__k2 : RBF(length_scale=0.5)
k1__k2__length_scale : 0.5
k1__k2__length_scale_bounds : (0.0, 10.0)
k2 : RBF(length_scale=2)
k2__length_scale : 2.0
k2__length_scale_bounds : (0.0, 10.0)
>>> print(kernel.theta)  # Note: log-transformed
[ 0.         -0.69314718  0.69314718]
>>> print(kernel.bounds)  # Note: log-transformed
[[      -inf 2.30258509]
 [      -inf 2.30258509]
 [      -inf 2.30258509]]
```

۲-۷-۲- هسته‌های پایه

هسته ثابت[1]را می‌توان به‌عنوان نوعی از هسته نام برد که به‌عنوان بخشی از یک هسته مجموع[2]، میانگین فرآیند گوسی را تغییر می‌دهد و به پارامتر constant-value بستگی دارد و به این صورت معادله (۲-۱)تعریف می‌شود:

$$k(x_i, x_j) = constant\_value \forall x_1, x_2 \quad (۱-۲)$$

مهم‌ترین استفاده هسته سفید این است که در آن مؤلفه نویز سیگنال را توضیح می‌دهد. تنظیم پارامتر nois_level مربوط به تخمین سطح نویز است و به این صورت تعریف می‌شود[۱۴]:

$$k(x_i, x_j) = noise_{level} \quad if\ x_i = x_j \ \ else\ 0 \quad (۲-۲)$$

۲-۷-۳- اپراتورهای هسته

عملگرهای کرنل یک یا دو هسته پایه را می‌گیرند و آن‌ها را به یک هسته جدید ترکیب می‌کنند. هسته مجموع، دو هسته k1 و k2 را می‌گیرد و را از طریق $k_{sum}(x,y) = k_1(x,y) + k_2(x,y)$ ترکیب می‌کند. هسته غیرثابت، دو هسته k1 و k2 را می‌گیرد و آن‌ها را از طریق $k_{\text{Product}}(x,y) = k_1(x,y) * k_2(x,y)$ ترکیب می‌کند. هسته توان، یک هسته پایه و یک پارامتر اسکالر p را می‌گیرد و آن‌ها را از طریق $k_{exp}(x,y) = k(x,y)^p$ ترکیب می‌کند[۵].

1 ConstantKernel

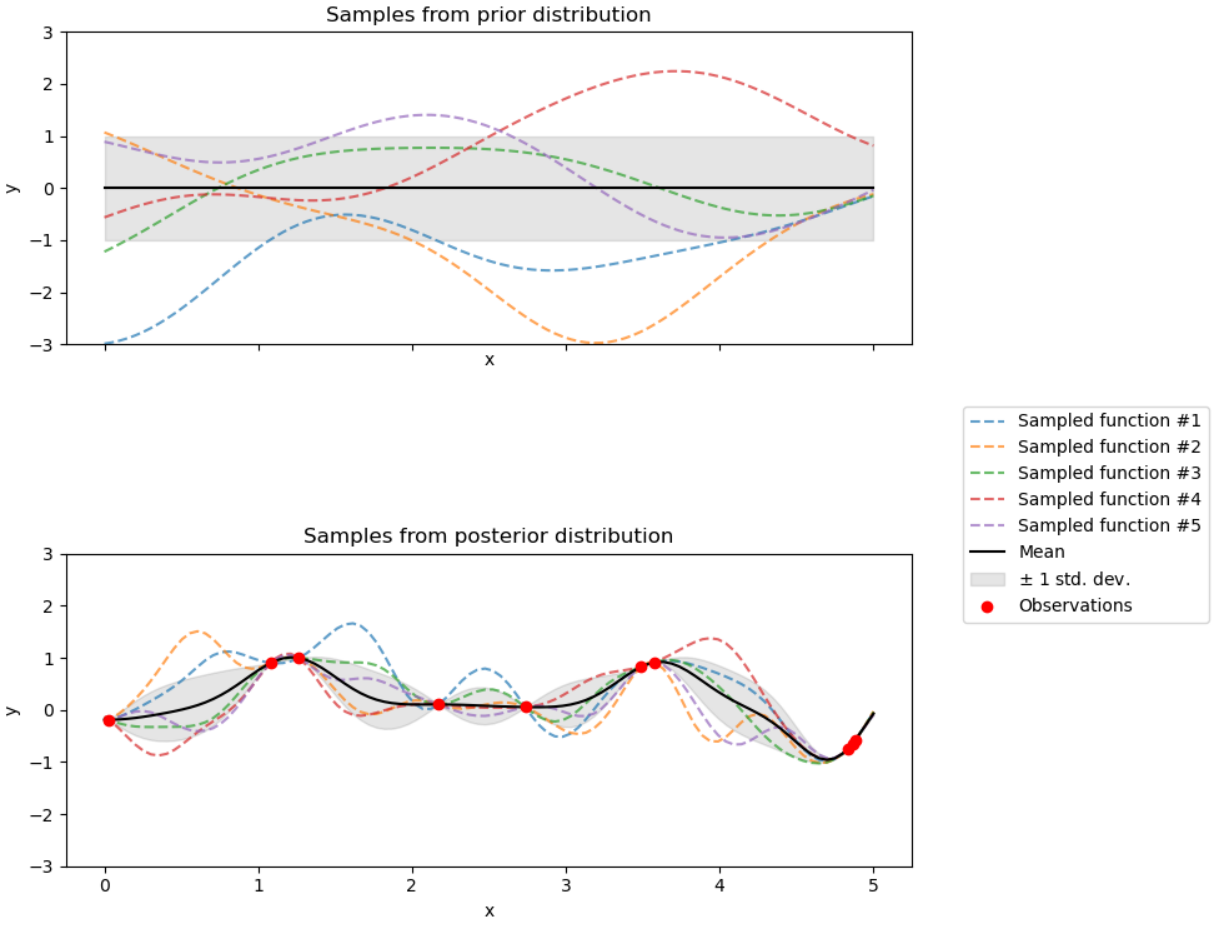
2 Sum kernel

۲-۷-۳-۱- هسته تابع پایه شعاعی

هسته تابع پایه شعاعی(RBF) یک هسته ثابت است. و به آن هسته مربع نمایی[1] هم گفته می‌شود که با پارامتر L که می‌تواند یک اسکالر (نوع همسانگرد هسته) یا یک بردار، برابر با تعداد ابعاد ورودی X (نوع ناهمسانگرد هسته) باشد، تعیین می‌شود. هسته توسط معادله (۲-۳) تعریف می‌شود[۱۵].

$$K(x_i, x_j) = exp\left(-\frac{d(x_i, x_j)^2}{2l^2}\right) \quad (۲-۳)$$

در معادله (۲-۳)، $d(x_i, x_j)$ نشان‌دهنده فاصله اقلیدسی است. احتمال پیشین و احتمال پسین مربوط به فرآیند گوسی در شکل ۱۵ نمایش داده‌شده است.



شکل ۱۵: احتمال پیشین و احتمال پسین فرآیند گوسی با هسته RBF [۱۵]

1 Squared exponential

۲-۷-۳-۲- هسته ماترن

هسته ماترن[1] یک هسته ثابت و تعمیم هسته RBF است. دارای یک پارامتر v است که صاف و هموار بودن تابع حاصل را کنترل می‌کند. با یک پارامترl که یک اسکالر (نوع همسانگرد هسته) یا یک بردار برابر با تعداد ابعاد ورودی x (نوع ناهمسانگرد هسته) تعیین می‌شود و توسط معادله (۲-۴) بیان می‌شود[۱۲]:

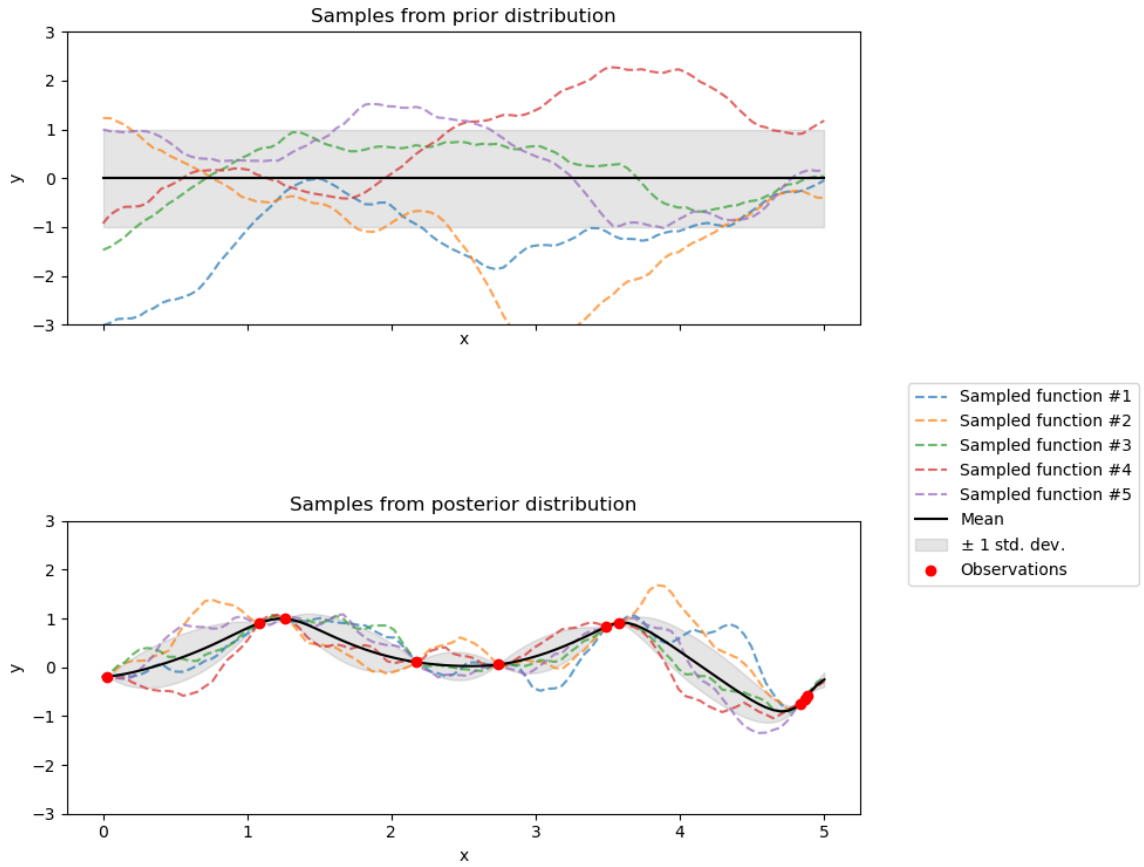
$$K(x_i, x_j) = \frac{1}{\Gamma(v)2^{v-1}}\left(\frac{\sqrt{2v}}{l}d(x_i, x_j)\right)^{v} k_v\left(\frac{\sqrt{2v}}{l}d(x_i, x_j)\right) \quad (۴-۲)$$

که در آن $d(x_i, x_j)$ فاصله اقلیدسی است، k_v یک تابع بسل اصلاح‌شده و $\Gamma(v)$ تابع گاما است. وقتی v به سمت بی‌نهایت برود، هسته ماترن به هسته RBF همگرا می‌شود. وقتی $v = \frac{1}{2}$، هسته ماترن با هسته نمایی مطلق یکسان می‌شود، یعنی:

$$K(x_i, x_j) = exp\left(-\frac{d(x_i, x_j)}{l}\right) \quad (۵-۲)$$

احتمال پیشین و احتمال پسین فرآیند گوسی با استفاده از هسته ماترن در شکل ۱۶ نمایش داده‌شده است.

1 Matern kernel



شکل ۱۶: احتمال پیشین و احتمال پسین فرآیند گوسی با هسته ماترن [۱۲]

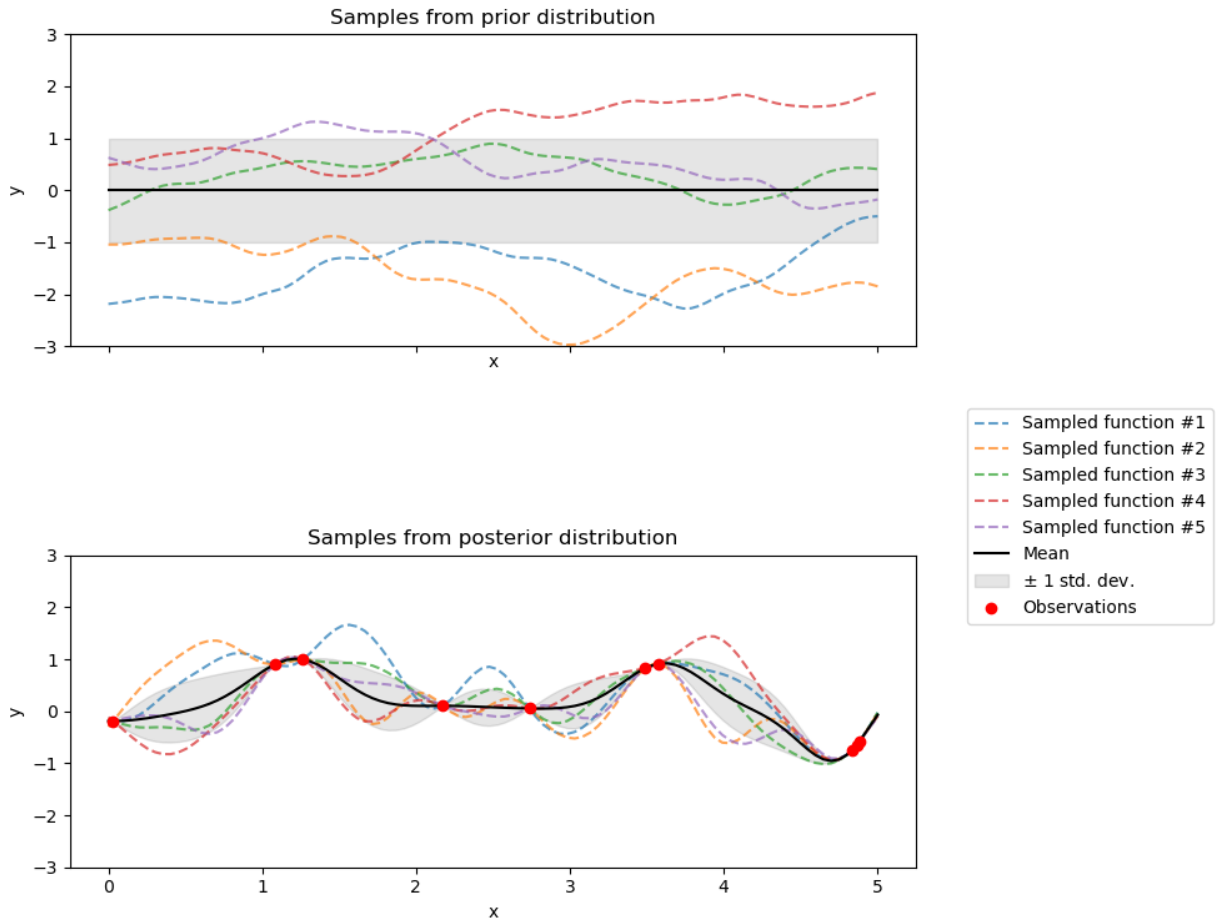
۲-۷-۳-۳- هسته درجه دوم کسری

هسته درجه دوم کسری[1] را می‌توان به‌عنوان مجموع نامتناهی از هسته‌های RBF با مقیاس‌های طول متفاوت مشاهده کرد. توسط یک پارامتر طول L و یک پارامتر α مطابق معادله (۲-۶) مشخص می‌شود.

$$K(x_i, x_j) = \left(1 + \frac{d(x_i, x_j)^2}{2\alpha l^2}\right)^{-\alpha} \quad (۲-۶)$$

احتمال پیشین و پسین فرآیند گوسی که از هسته درجه دوم کسری حاصل می‌شود در شکل ۱۷ نشان داده‌شده است[۷].

1 Rational quadratic kernel



شکل ۱۷: احتمال پیشین و پسین فرآیند گوسی با هسته درجه دوم کسری[۷]

۲-۷-۳-۴- هسته مربع سینوسی نمایی

هسته مربع سینوسی نمایی[1] برای مدل‌سازی توابع پریودیک استفاده می‌شود. با استفاده از پارامتر طول L و پارامتر تناوب P توسط معادله (۲-۷) تعریف می‌شود[۱۱].

$$K(x_i, x_j) = exp\left(\frac{2sin^2\left(\pi d(x_i, x_j)/p\right)}{2l^2}\right) \quad (۷-۲)$$

1 ExpSineSquared

۲-۷-۳-۵- هسته نقطه حاصل‌ضرب

هسته نقطه حاصل‌ضرب غیرثابت است. این هسته نسبت به چرخش مختصات در مبدأ تغییر نمی‌کند، اما با انتقال دچار تغییر می‌شود. اگر $\sigma_0^2 = 0$ این هسته را هسته خطی همگن هم می‌نامند، در غیر این صورت ناهمگن است و توسط معادله (۲-۸) تعریف می‌شود[۱۶].

$$K(x_i, x_j) = \sigma_0^2 + x_i . x_j \quad (۸-۲)$$

فهرست مراجع

[۱] .(۲۰۲۲) *https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpr_noisy.html*. Available: https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpr_noisy.html

[۲] .(۰۱/۲۰۲۲)Available: https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpr_noisy.html

[۳] Available: https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpr_noisy.html

[۴] .(۲۰۲۲) 'Available: https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_compare_gpr_krr.html

[۵] .(۲۰۲۲)Available: https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpc.html

[۶] .(۲۰/۰۱/۲۰۲۲)Available: https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessClassifier.html#sklearn.gaussian_process.GaussianProcessClassifier

[۷] .(۰۵/۰۱/۲۰۲۲)Available: https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessClassifier.html#sklearn.gaussian_process.GaussianProcessClassifier

[۸] .(۲۶/۰۱/۲۰۲۲)Available: https//:scikit-learn.org/stable/modules/multiclass.html#multiclass

[۹] .(۰۲/۰۱/۲۰۲۲)Available: https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpc.html

[۱۰] .(۲۲/۰۱/۲۰۲۲)Available: https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpc_xor.html

[۱۱] .(۲۳/۰۱/۲۰۲۲)Available: https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpc_iris.html

[۱۲] .(۱۴/۰۱/۲۰۲۲)Available: https://scikit-learn.org/stable/modules/gaussian_process.html#duv2014

[۱۳] .(۱۸/۰۱/۲۰۲۲)Available: https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.WhiteKernel.html#sklearn.gaussian_process.kernels.WhiteKernel

[۱۴] .(۲۴/۰۱/۲۰۲۲)Available: https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.ConstantKernel.html#sklearn.gaussian_process.kernels.ConstantKernel

[۱۵] .(۲۹/۰۱/۲۰۲۲)Available: https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpr_prior_posterior.html

[۱۶] .(۲۸/۰۱/۲۰۲۲)Available: https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.DotProduct.html#sklearn.gaussian_process.kernels.DotProduct