"LIBRERÍA EN C++ PARA PROCESOS GAUSSIANOS DE MÚLTIPLE SALIDA"

CARLOS ÁLVARO GONZÁLEZ ECHEVERRY

ALEJANDRO SUÁREZ GARCÍA

MANUEL FELIPE PINEDA LOAIZA

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERÍAS

INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

PEREIRA

FEBRERO DE 2016

“LIBRERÍA EN C++ PARA PROCESOS GAUSSIANOS DE MÚLTIPLE SALIDA”

CARLOS ÁLVARO GONZÁLEZ ECHEVERRY

ALEJANDRO SUÁREZ GARCÍA

MANUEL FELIPE PINEDA LOAIZA

INFORME DE PROYECTO DE GRADO DE PREGRADO

UNIVERSIDAD TECNOLÓGICA DE PEREIRA

FACULTAD DE INGENIERÍAS

INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

PEREIRA

FEBRERO DE 2016

Agradecimientos

Al candidato a PhD. Sebastián Gómez González, sin él no hubiésemos comenzado ni terminado este proyecto.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

1.1 Marco referencial

1.1.1 Marco Histórico

Los procesos Gaussianos, son como su nombre lo indica, una generalización de la distribución Gaussiana (normal), y una de sus principales características es su capacidad de describir las propiedades de funciones, mas aún si se necesitaran las propiedades de una función, teniendo en cuenta solo un número finito de puntos, la inferencia Gaussiana responderá con la misma respuesta si se ignoran los (infinitos) otros puntos, como si se hubieran tenido todos en cuenta. Este tipo de ideas ya se han considerado anteriormente, de hecho muchos problemas de Machine Learning se han llegado a modelar como un tipo de proceso Gaussiano "restringido", debido precisamente a sus ventajas a nivel computacional, sin embargo estos procesos son, tal vez no tan conocidos como deberían, Rasmussen ha sido uno de los grandes autores en esta área, habiendo publicado gran cantidad de artículos y un libro al respecto[18], y ha sido él quien ha desencadenado de nuevo interés en el área, y uno de los interesados ha sido Neil Lawrence, quien ha trabajado con Procesos Gaussianos, para el denominado Modelo de Variable Latente (Modelo estadístico en el que se relacionan Variables Manifiestas o "visibles" con variables latentes o "invisibles") para procesos gaussianos[9].

Trabajos anteriores

Según los artículos consultados, alrededor de 2003, el profesor Neil Lawrence (Universidad de Sheffield) inició su trabajo en Procesos Gaussianos[14], y en 2004 empieza a indagar en el modelo de variable latente[9], sin embargo no es hasta 2009 que empieza a publicar literatura específicamente sobre procesos Gaussianos de múltiple salida, en compañía del Doctor Mauricio Álvarez (Universidad de Manchester, Profesor en la Universidad tecnológica de Pereira)[19]. Todo esto eventualmente lleva al inicio de la implementación de una librería de procesos Gaussianos de Múltiple salida por parte de Lawrence y Álvarez, eventualmente esta implementación fue usada por Sebastián Gómez, quien la usaría en su proyecto de Maestría "Classification of

time series with Multiple Output Gaussian Processes and MCE" específicamente implementada en MatLAB.

Estado del arte

Actualmente se cuenta con algunas implementaciones de procesos gaussianos para aprendizaje automático, entre ellas un conjunto de herramientas escrito en MatLAB [18] y una implementación para procesos gaussianos de variable latente publicada en [13], pero no se cuenta con ninguna implementación para procesos gaussianos de múltiple salida.

Una de las referencias más importantes para este proyecto es GPy[6], ya que es un marco de trabajo de desarrollo para procesos gaussianos escrito en python el cual cuenta con una comunidad activa de desarrolladores.

1.1.2 Marco teórico

Proceso estocástico

En teoría de probabilidad un proceso estocástico (a veces llamado proceso aleatorio) es una colección de variables aleatorias, representando la evolución de algún sistema de valores aleatorios a través del tiempo. Esta es la contraparte probabilística de un proceso determinista (o sistema determinista). En vez de describir un proceso que sólo puede evolucionar de una manera (por ejemplo en el caso de las soluciones de una ecuación diferencial ordinaria), en un proceso estocástico hay cierto grado de no-determinismo: incluso si la condición inicial es conocida, hay cierta cantidad (o infinita cantidad) de direcciones en las que el proceso puede evolucionar.

Procesos Gaussianos

Un proceso Gaussiano es un proceso estocástico cuyos valores observados consisten de valores aleatorios asociados con cada punto en un rango de tiempo (o espacio) de manera que cada variable aleatoria tenga una distribución normal. Mas aún cada colección finita de las variables aleatorias tiene una distribución normal multivariada. El concepto de proceso Gaussiano es nombrado así debido a Carl Friedrich Gauss, porque estaba basada en la noción de distribución normal, que es usualmente llamada Distribución Gaussiana. De hecho una manera de pensar en un proceso Gaussiano es como una generalización de la distribución normal multivariada de infinitas dimensiones.[17]

Los procesos Gaussianos son importantes en el modelamiento estadístico debido a sus propiedades heredadas de la distribución normal. Por ejemplo si un proceso es modelado como un Proceso Gaussiano, las distribuciones de varias cantidades derivadas pueden ser obtenidas de manera explícita. Algunas de estas cantidades son: el valor medio del proceso en un rango de tiempo, el error de estimación del promedio usando valores de muestra en un conjunto pequeño de elementos.

Los Procesos Gaussianos son un conjunto de modelos generativos muy usados en Series de Tiempo. El Proceso Gaussiano de múltiples salidas es necesario para modelos generativos donde la entrada consiste de más de una serie de tiempo, correlacionadas entre sí. Este modelo generativo puede ser usado para problemas de clasificación, sin embargo la meta de optimización en el modelo generativo no es una función que mida el desempeño de la clasificación, sino una función de verosimilitud (likelihood) que es optimizada para cada clase por separado. El modelo generativo puede entonces ser usado para clasificar usando la regla de Bayes, se puede mostrar que este clasificador es óptimo, si están siendo de hecho generados con el modelo generativo, pero con datos reales, el modelo real es usualmente desconocido y los modelos que usamos son modelos que esperamos sean lo suficientemente flexibles para proveer un buen desempeño en la clasificación.

En un modelo discriminativo la función objetivo está directamente relacionado con el desempeño de la clasificación, y los resultados son usualmente mejores que aquellos de un modelo generativo para clasificación de series de tiempo como el reconocimiento de voz. Juang et al[7] propuso un método para entrenar de manera discriminativa un modelo de Markov con "error mínimo de clasificación" (Minimum Clasification Error, MCE). Probando su método en una tarea de reconocimiento de voz prueba un 50 % de mejora comparado con la aproximación usando Maximum Likelihood para el mismo modelo. Ben Yishai y Burshtein [5] propusieron un entrenamiento discriminativo para un Modelo Oculto de Markov (Hidden Markov Model, HMM) usando "máxima información mutua", mostrando mejoras entre el 15 % y 28 % con respecto a la aproximación usando ML, también para tareas de clasificación de voz.

Nótese que las referencias previas usan HMM como un modelo generativo para series de tiempo, pero otra aproximación popular para modelar series de tiempo es el uso de procesos Gaussianos.

1.1.3 Marco conceptual

- **Librería:** Es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca. En este caso se busca implementar cierta funcionalidad de los proceso Gaussianos de múltiple salida en el lenguaje C++.
- **API:** Es el conjunto de funciones y procedimientos que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción.
- **Verosimilitud:** Una función de verosimilitud es una función de los parámetros de un modelo estadístico, definida así: la verosimilitud de un conjunto de valores de parámetros dados algunos resultados observados es igual a la probabilidad de esos resultados dados los valores de los parámetros. Las funciones de verosimilitud juegan un papel clave en la inferencia estadística.
- **Valor Observado:** En estadística, un valor observado de una variable aleatoria es el valor que es de hecho observado (lo que realmente pasó). La variable aleatoria como tal debería verse como el proceso de como se realiza la observación. Cantidades estadísticas calculadas a través de valores observados, sin el uso de modelado se llaman empíricas (distribución empírica o probabilidad empírica)
- **Distribución Normal o Gaussiana:** Es una distribución de probabilidad (una función que muestra la probabilidad de que cualquier observación real esté entre dos límites o números reales) que ocurre muy a menudo. Las distribuciones normales son de extrema importancia en estadística y son usadas en todo tipo de áreas en las que las distribuciones no son conocidas.
- **Complejidad computacional:** Formaliza en términos matemáticos la cantidad de recursos que un computador necesita para resolver un problema en particular. Dichos recursos pueden ser, por ejemplo, el tiempo de ejecución o la cantidad de memoria utilizada por el programa.

Capítulo 2

Implementación de la librería

2.1 Administración del proyecto

Debido a que desde su inicio gplib nace como un proyecto de software libre, se decide utilizar el modelo de construcción colaborativa descrito en [1]. Este modelo brinda la posibilidad de que nuevas personas contribuyan al proyecto de manera efectiva. El modelo c4 brinda entre otras, las siguientes características las cuales son importantes para el proyecto.

- Utilizar git como sistema de gestión de versiones distribuido.
- Mantener el proyecto público en github.com [2]
- Utilizar una plataforma para administrar las tareas.
- Utilizar una guía de estilo para todo el proyecto.
- Mantener un registro de todos los cambios que se realizaron en el proyecto (en el administrador de tareas).
- Garantizar que todo el código sea revisado por al menos 2 personas.
- Mantener una documentación oportuna de la API pública ofrecida por la librería.

2.1.1 Pruebas

Para las pruebas unitarias de cada una de las clases se decidió usar el módulo de pruebas de la librería BOOST para C++ [1]. En general, con la implementación de cada una de las clases como tal, se diseñaron un conjunto de pruebas que aseguraran su correcta funcionalidad, esto es particularmente importante en lo referente a lo cálculos de los gradientes, y asegura que estén correctamente implementados a través de la comparación entre los gradientes numéricos y los analíticos.

[1] Mas información en `www.boost.org/doc/libs/1_57_0/libs/test/doc/html/index.html`

Las pruebas se pueden aplicar fácilmente previo a la instalación de la librería usando el comando `make test`.

2.1.2 Integración continua

Así mismo se usó el sistema de integración continua Travis [2] (Gratuito para proyectos Open Source), este sistema lanza una instancia del proyecto y corre el conjunto de pruebas cada vez que se realiza un nuevo "commit" en `github.com/pin3da/gplib` en cualquiera de sus ramas, esto garantiza que la última versión de la rama master, es decir la rama principal del proyecto, funciona correctamente. Otra de las grandes ventajas de usar el sistema de integración continua es la capacidad de llevar registro de qué versiones del código han funcionado correctamente y cuales han fallado, y mas aún, el hecho de poder utilizar varios compiladores para la realización de las mismas pruebas.

2.1.3 Documentación

Un esqueleto de la documentación necesaria para instalar la librería puede visualizarse fácilmente en la página principal del proyecto en github [3], este esqueleto contiene las dependencias del proyecto, así como los comandos necesarios para correr las pruebas e instalar la librería, está escrito usando Markdown que es renderizado por github.

Para la realización de la documentación completa del proyecto se utilizó Doxygen, una herramienta para generación de documentación de C++, con capacidad de generar manuales de referencia, o documentación online (HTML) a partir de anotaciones (comentarios) en el código fuente, y no solo esto, Doxygen tiene también gran cantidad de utilidades, como generar diagramas de clase y colaboración, utilizando Graphviz[4]. La documentación generada por doxygen se puede crear localmente (si se tienen instaladas las herramientas) o se puede encontrar en `pin3da.github.io/gplib/`.

2.2 Dependencias

2.2.1 Armadillo

Para todas las operaciones algebraicas que utilizan matrices se utilizó la librería Armadillo, que no solo tiene implementados gran cantidad de métodos que facilitan considerablemente el manejo de las matrices, sino que a su vez utiliza otras librerías como OpenBLAS, LAPACK y ARPACK para mejorar considerablemente su desempeño.

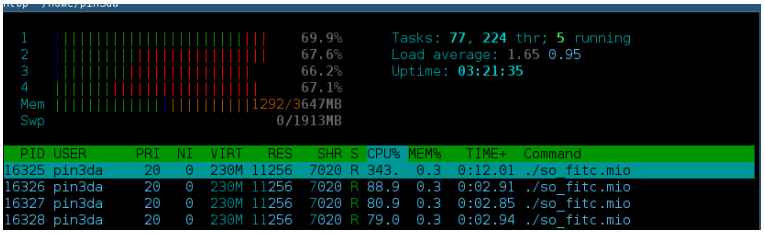
[2] `https://travis-ci.org/pin3da/gplib`

[3] `https://github.com/pin3da/gplib`

[4] Más información sobre Doxygen `http://www.stack.nl/~dimitri/doxygen/index.html`, Graphviz `http://www.graphviz.org`

Otra de las mas grandes ventajas de utilizar Armadillo es su capacidad de utilizar OpenBLAS para paralelizar en procesador las operaciones que lo permiten, mejorando considerablemente el desempeño de la librería, esto se ve particularmente evidenciado en problemas con gran cantidad de datos.

Figura 2.1: Uso de los 4 hilos del procesador al ejecutar una regresión de única salida usando FITC



2.2.2 NLopt

Para los procesos de optimización (entrenamiento) de la regresión utilizamos la librería de optimización NLopt[5], desarrollada en Massachussets Institute of Technology (MIT), esta librería es software libre y provee gran cantidad de algoritmos para optimización no lineal, y es capaz de lidiar con gran cantidad de parámetros de optimización, así como con gran cantidad de restricciones. Para nuestro caso particular se utilizó el MMA (Method of Moving Asymptotes), un algoritmo de optimización local basado en Gradiente.

2.3 Instalación

El proceso de instalación detallado se puede encontrar en la página principal del proyecto en github, pero básicamente consiste en seguir los pasos básicos de instalación estándar de la gran mayoría de librerías que utilizan la herramienta **make**. Simplemente se deben correr los siguientes comandos una vez instaladas las dependencias:

```
make
make install
```

Pruebas previo a la instalación:

```
make check
```

Para probar la instalación:

```
make installcheck
```

[5] `http://ab-initio.mit.edu/wiki/index.php/NLopt`

Capítulo 3

Regresión

3.1 FULL GP

A continuación se da una breve explicación de como funciona el proceso completo de regresión de múltiple salida: En general, un proceso gaussiano, es básicamente una distribución Gaussiana sobre funciones:

$$GP = \mathcal{N}(\mu, \Sigma) \tag{3.1}$$

Donde μ es un vector de medias y Σ una matriz de covarianza. Cuando se hace una regresión usando procesos Gaussianos, se busca obtener una distribución multivariada capaz de aproximar una función basada en un set de entrenamiento. Para obtener esta distribución, usualmente denominada distribución predictiva basta simplemente condicionar la distribución a los datos observados. En nuestro caso, si las entradas X están compuestas por X_1, el conjunto de datos nuevos o faltantes, y X_2 el conjunto de datos observados, y de manera acorde, se tiene que:

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \tag{3.2}$$

Donde μ es el vector completo de medias, μ_1 hace referencia al conjunto de medias de los datos no observados y μ_2 a las medias de los datos observados. Y además de ello:

$$\Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \tag{3.3}$$

Donde Σ hace referencia a la matriz de covarianza completa, Σ_{11} hace referencia a la covarianza de los datos no observados, Σ_{12} hace referencia a la covarianza de los datos no observados con respecto a los observados, etc. La nueva distribución predictiva condicionada por $X_2 = y$, donde y representa simplemente el conjunto de resultados para X_2 (su correspondiente valor de las variables dependientes), está dada por[10]:

$$N(\bar{\mu}, \bar{\Sigma}) \tag{3.4}$$

Donde

$$\bar{\mu} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(y - \mu_2) \tag{3.5}$$

y

$$\bar{\Sigma} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} \tag{3.6}$$

Sin embargo, se debe hacer notar que entonces, para obtener esta nueva distribución predictiva debemos haber calculado μ y Σ con anterioridad, usualmente se puede inicializar el vector de medias μ con un conjunto de ceros, sin embargo la covarianza Σ debe ser calculada a través de una función kernel. Para el caso de múltiples salidas, usamos un kernel basado en el Linear Model of Corregionalization (LMC), y está dado por:

$$K = \sum_{q=1}^{Q} \beta_q \otimes k_q(X, Y) \tag{3.7}$$

donde X y Y son las matrices de entrada para la evaluación del kernel, k es un conjunto de funciones kernel (de una sola salida) y β un conjunto de matrices que indican la correlación entre los distintos componentes de X y Y para cada una de las Q funciones latentes. Para ahondar mas en el concepto de funciones latentes puede consultarse [4], pero por ahora sírvase decir que es un conjunto de funciones implícitas que relacionan X y Y. La cantidad de funciones latentes debe ser exclusivamente menor a la cantidad de salidas.

Con lo anterior se puede definir a Σ así:

$$\Sigma = K(X, X) \tag{3.8}$$

Sin embargo, ahora tenemos que el kernel de múltiples salidas necesario para calcular Σ requiere una gran cantidad de hiper-parámetros, dados por el conjunto de matrices β y los parámetros de cada uno de los kernels internos (en nuestro caso un conjunto de kernels squared exponential), se hace necesario entonces "entrenar" los valores de estos parámetros de acuerdo a los datos de entrada conocidos (conjunto de entrenamiento) para obtener el mejor resultado posible, para este propósito es necesario utilizar un optimizador, en nuestro caso se utiliza un optimizador basado en gradientes.

La función objetivo a optimizar para la distribución de probabilidad es denominada Log Likelihood, esta función representa la probabilidad de los parámetros dado un conjunto específico de datos y sus resultados (en nuestro caso, los datos observados o el conjunto de entrenamiento), por consecuencia optimizar esta función equivale a encontrar los parámetros que maximicen esta probabilidad dado el conjunto de entrenamiento. El Log Likelihood ($ln(L)$) está dado por:

$$ln(L) = -\frac{1}{2}ln(|\Sigma|) - \frac{1}{2}y^T\Sigma^{-1}y - \frac{n}{2}ln(2\pi) \tag{3.9}$$

Y teniendo en cuenta el ruido (σ_{noise}):

$$ln(L) = -\frac{1}{2}ln(|\Sigma + \sigma^2_{noise}I|) - \frac{1}{2}y^T(\Sigma + \sigma^2_{noise}I)^{-1}y - \frac{n}{2}ln(2\pi) \tag{3.10}$$

Donde $y = \mu$ es el vector de medias, Σ la matriz de covarianza y n la cantidad de observaciones, para el caso del entrenamiento la matriz de covarianza y la media solo contendrán los datos referentes al conjunto de entrenamiento (marginal), no se tienen en cuenta los puntos no observados, es decir que durante el entrenamiento $y = \mu = \mu_2$, y $\Sigma = K(X_2, X_2)$. A continuación se muestran las derivadas de esta función con respecto a los hiperparámetros.

Sea θ_i un parámetro arbitrario, y $R = \Sigma + \sigma^2_{noise}I$ la derivada del log marginal viene dada por:

$$\frac{\partial}{\partial\theta_i}ln(L) = -\frac{1}{2}tr(R^{-1}\frac{\partial R}{\partial\theta_i}) + \frac{1}{2}y^TR^{-1}\frac{\partial R}{\partial\theta_i}R^{-1}y \tag{3.11}$$

3.2 Aproximación FITC

Debido a la alta complejidad del proceso de regresión completo ($O(n^3)$), se implementó una aproximación denominada Fully Independent Training Conditional (FITC), esta aproximación está basada en el concepto de sparse pseudo-inputs o sparse inducing points (puntos ocultos), y de una manera muy abstracta consiste en "resumir" la información provista por el conjunto de entrenamiento completo en una cantidad menor de puntos[16].

A pesar que el objetivo y el procedimiento siguen siendo muy similares, el uso de los puntos ocultos requiere varios ajustes en las ecuaciones. En primer lugar, la distribución predictiva de FITC está dada también por una distribución normal $N(\mu, \Sigma)$, donde:

$$\mu = Q_{*,f}(Q_{f,f} + \Lambda)^{-1}y \tag{3.12}$$

y

$$\Sigma = K_{*,*} - Q_{*,f}(Q_{f,f} + \Lambda)^{-1}Q_{f,*} \tag{3.13}$$

En esta notación, y indica los resultados de los puntos conocidos, los asteriscos ($*$) denotan el conjunto de puntos no conocidos, y las f el conjunto de puntos conocidos. Así $K_{*,*}$ significa la evaluación del kernel sobre el conjunto de datos nuevos o no observados, $K_{*,f}$ el kernel evaluado sobre el conjunto de puntos no observados y el conjunto de puntos observados, etc. y a su vez

$Q_{x,y}$ (donde x y y pueden representar el conjunto de datos no observados, o el conjunto de datos observados) está dado por:

$$Q_{x,y} = K_{x,u}K_{u,u}^{-1}K_{u,y} \tag{3.14}$$

Donde u representa el conjunto de puntos ocultos. Adicionalmente Λ está dado por:

$$\Lambda = diag[K_{f,f} - Q_{f,f}] + \sigma^2 I \tag{3.15}$$

Donde σ es el ruido e I la matriz identidad del tamaño apropiado.

Finalmente la función Log Likelihood para FITC sería:

$$ln(L) = -\frac{1}{2}ln(|Q_{f,f} + \Lambda|) - \frac{1}{2}y^T(Q_{f,f} + \Lambda)^{-1}y - \frac{n}{2}ln(2\pi) \tag{3.16}$$

A continuación se presentan las derivadas del Log Likelihood de FITC.

Sea θ_i un parámetro arbitrario de la función objetivo:

$$Q_{ff} = K_{fu}{K_{uu}}^{-1}K_{uf} \tag{3.17}$$

$$\frac{\partial Q_{ff}}{\partial \theta_i} = K_{fu}\frac{\partial {K_{uu}}^{-1}K_{uf}}{\partial \theta_i} + \frac{\partial K_{fu}}{\partial \theta_i}{K_{uu}}^{-1}K_{uf} \tag{3.18}$$

$$\frac{\partial Q_{ff}}{\partial \theta_i} = K_{fu}\Big[{K_{uu}}^{-1}\frac{\partial K_{uf}}{\partial \theta_i} - ({K_{uu}}^{-1}\frac{\partial K_{uu}}{\partial \theta_i}{K_{uu}}^{-1})K_{uf}\Big] + \frac{\partial K_{fu}}{\partial \theta_i}{K_{uu}}^{-1}K_{uf} \tag{3.19}$$

$$\frac{\partial Q_{ff}}{\partial \theta_i} = K_{fu}{K_{uu}}^{-1}\big[\frac{\partial K_{uf}}{\partial \theta_i} - (\frac{\partial K_{uu}}{\partial \theta_i}{K_{uu}}^{-1})K_{uf}\big] + \frac{\partial K_{fu}}{\partial \theta_i}{K_{uu}}^{-1}K_{uf} \tag{3.20}$$

$$R = Q_{ff} + diag[K_{ff} - Q_{ff}] + \sigma^2_{noise}I \tag{3.21}$$

$$\frac{\partial R}{\partial \theta_i} = \frac{\partial Q_{ff}}{\partial \theta_i} + diag[\frac{\partial K_{ff}}{\partial \theta_i}] - diag[\frac{\partial Q_{ff}}{\partial \theta_i}] + \frac{\partial \sigma^2_{noise}I}{\partial \theta_i} \tag{3.22}$$

$$log(y|X) = -\frac{1}{2}log|R| - \frac{1}{2}\mathbf{y}^\top R^{-1}\mathbf{y} - \frac{n}{2}log(2\pi) \tag{3.23}$$

$$\frac{\partial log(y|X)}{\partial \theta_i} = -\frac{1}{2}tr\Big(R^{-1}\frac{\partial R}{\partial \theta_i}\Big) + \frac{1}{2}\mathbf{y}^\top R^{-1}\frac{\partial R}{\partial \theta_i}R^{-1}\mathbf{y} \tag{3.24}$$

Capítulo 4

Pruebas de funcionalidad

Con el fin de comprobar el correcto funcionamiento de la librería, se realizaron pruebas de funcionalidad, pero debido a la naturaleza aleatoria de los procesos gaussianos no es posible realizar estas pruebas utilizando métodos exactos o el error cuadrático medio, ya que son una generalización de la distribución normal sobre la función. El log marginal likelihood es una medida de la confiabilidad de los datos, y se calcula como $log(L(y|X,\theta))$ donde y es la salida esperada, X es el conjunto de entrada y θ son los hiperparámetros[8] por lo tanto se puede decir que a mayor valor del log marginal likelihood más acertada es la regresión.

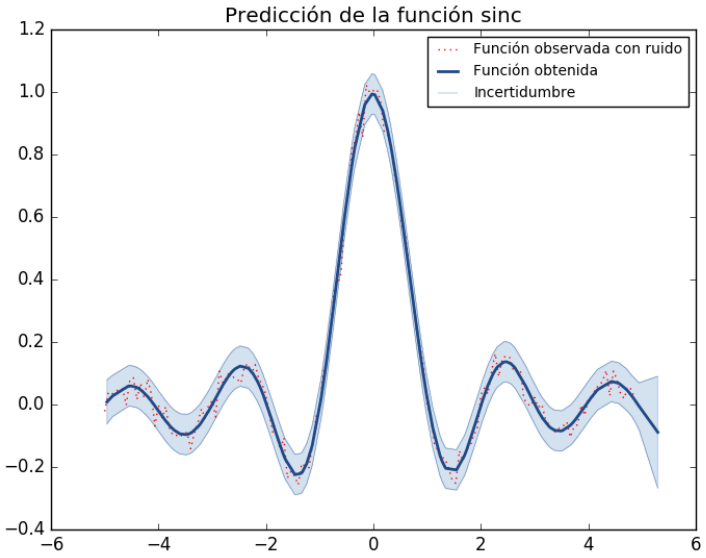
Para cada tipo de proceso gaussiano se realizaron pruebas de funcionalidad, haciendo regresión sobre la función *sinc* normalizada, definida como:

$$sinc_N(x) = \frac{seno(\pi x)}{\pi x} \tag{4.1}$$

4.1 GP de única salida

Para el GP de una salida se utilizó la función: $sinc(x) + \sigma$ donde σ es una variable aleatoria normalmente distribuida con media 0 y varianza 0,001, la cual es el ruido; para esta prueba se utilizaron 100 puntos y para optimizar máximo 1000 iteraciones con una tolerancia de $10e^{-4}$.

Figura 4.1: Predicción GP de única salida



4.2 GP de múltiple salida

4.2.1 Full GP

Para el GP de múltiples salidas, se realizaron dos pruebas idénticas, una utilizando el método FULL GP y la otra utilizando la aproximación FITC, para los cuales se utilizaron 3 funciones observadas, con 2 funciones latentes, siendo las funciones:

- $sinc(x) + \sigma$
- $sinc(x + \frac{\pi}{4}) + \sigma$
- $1{,}3 * sinc(x) + \sigma$

Donde σ es una variable aleatoria normalmente distribuida con media 0 y varianza 0,0001, para esta prueba se utilizaron 150 puntos y para optimizar máximo 1000 iteraciones con una tolerancia de $10e^{-4}$.

Figura 4.2: Predicción FULL GP 1

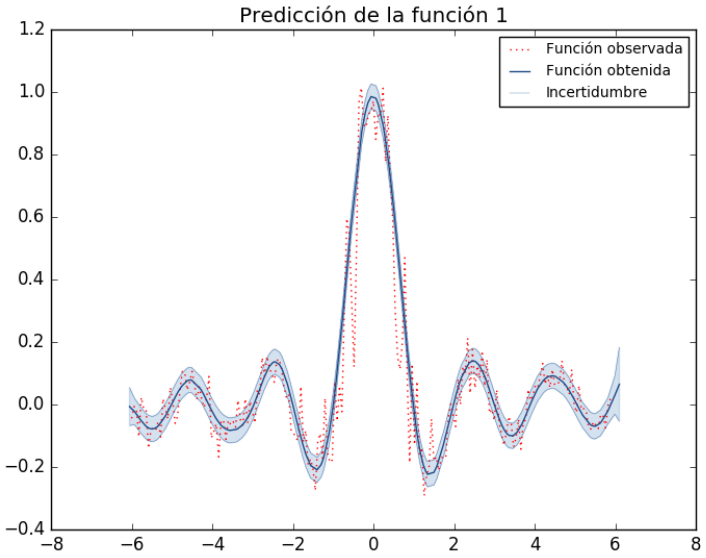


Figura 4.3: Predicción FULL GP 2

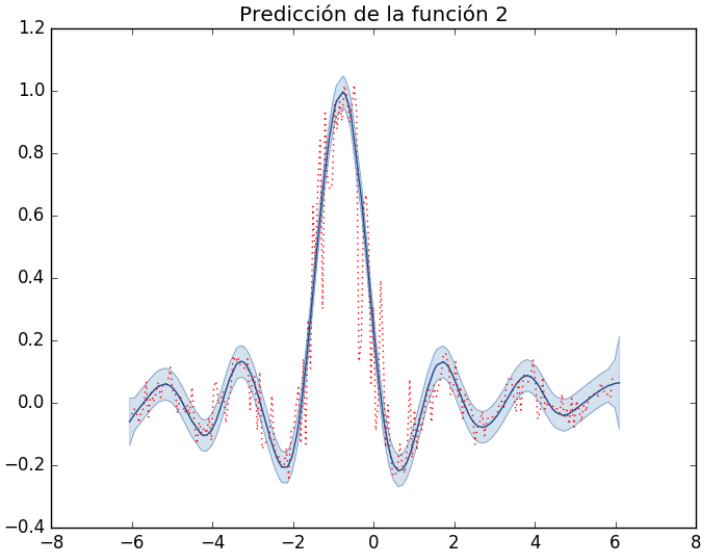
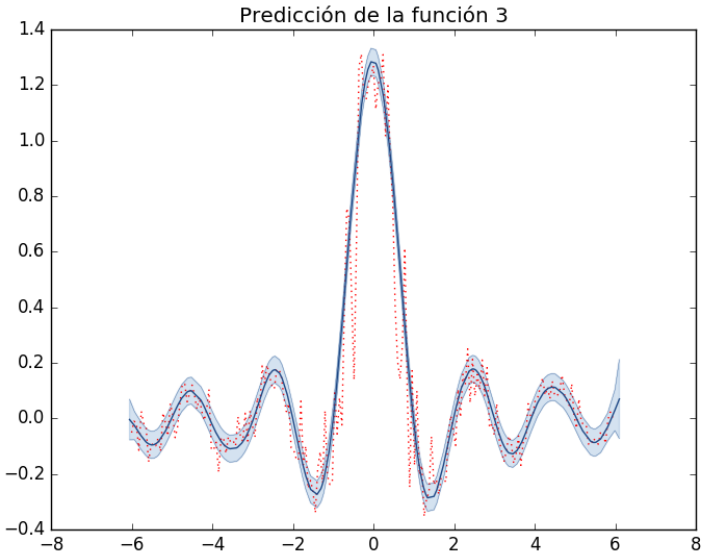


Figura 4.4: Predicción FULL GP 3



4.2.2 FITC

Para esta prueba, se utilizaron las mismas funciones y ruido que en las del FULL GP, con la diferencia de que se utilizaron 30 puntos ocultos.

Figura 4.5: Predicción FITC 1

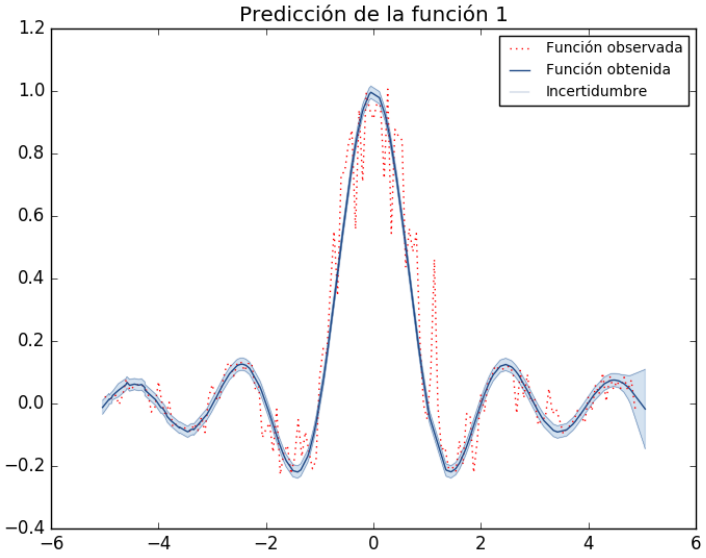


Figura 4.6: Predicción FITC 2

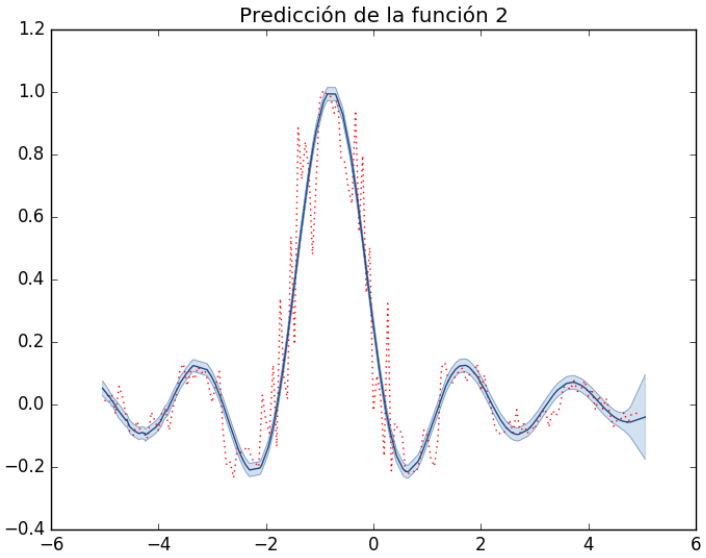
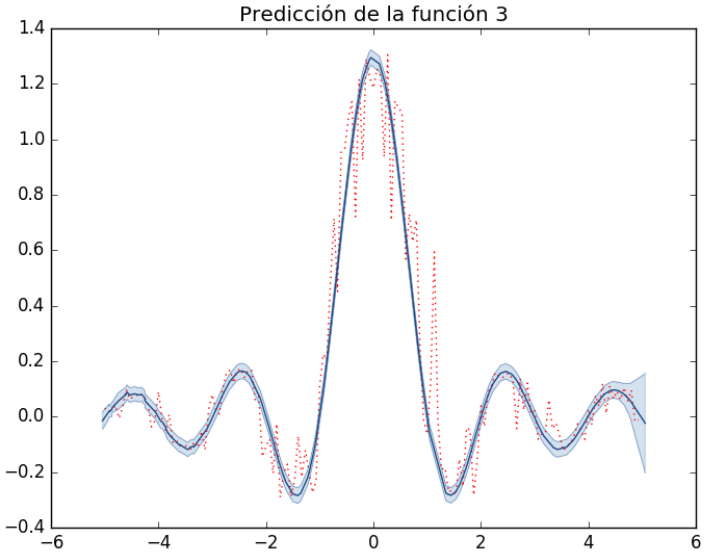


Figura 4.7: Predicción FITC 3



Puede ocurrir que durante el proceso de optimización se agrupen puntos ocultos en una misma región, y por consecuencia en otras regiones aumente la incertidumbre, por este motivo, como se puede apreciar en la parte final de las funciones en las gráficas de FITC, esta crece debido a la poca información en esta región.

Capítulo 5

Pruebas de desempeño

En estas pruebas se compara el desempeño de la librería usando los dos métodos descritos anteriormente.

- El primer método es el FULL GP, el cual tiene una complejidad de $O(N^3)$, donde N representa el número de puntos.
- El segundo método es la aproximación FITC, el cual tiene una complejidad teórica de $O(N * K^2)$, donde N es el número de puntos y K es el número de puntos ocultos.

Para cada una de las pruebas, se toma un máximo de 1000 iteraciones para el entrenamiento y se lleva registro de 3 mediciones diferentes en cada una de estas iteraciones.

- **Función de costo:** La función a optimizar es el log marginal likelihood, de ahora en adelante descrita como log_marginal.
- **Cálculos previos al gradiente:** A fin de optimizar el desempeño de la librería se precalculan algunos términos los cuales son comunes a todas las derivadas del vector de gradientes. En adelante referenciado como before_grad.
- **Cálculos del gradiente:** Tiempo total en calcular todas las derivadas del gradiente. Se garantiza que todas las iteraciones de una misma prueba calculan la misma cantidad de derivadas.

Después de tomados los tiempos en cada iteración, se promedian y se clasifican como se mostrará a continuación.

5.1 GP de única salida

En esta prueba se fija el número de puntos ocultos a 30 y se utilizan diferentes valores de N.

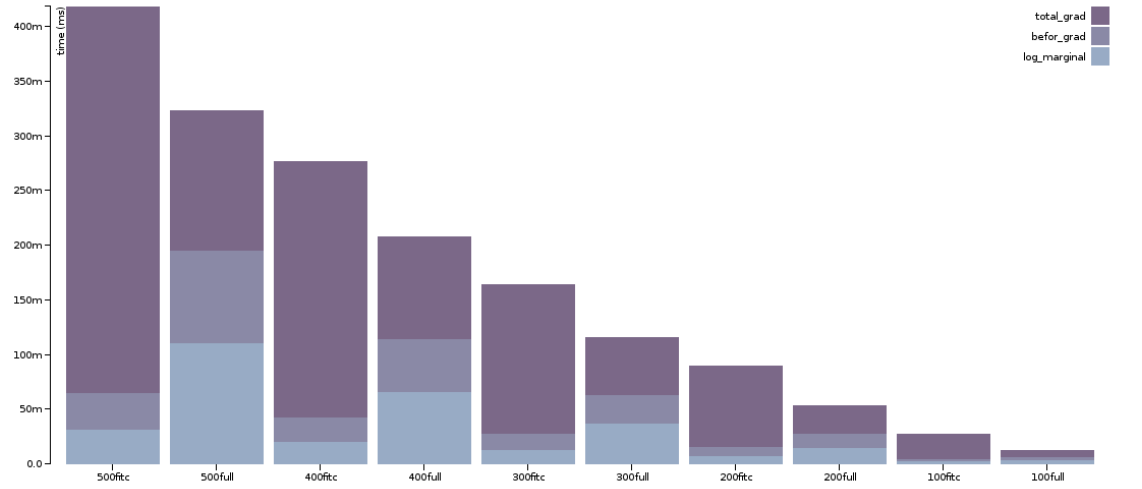
5.1.1 Caso 1: $N <= 500$

Como se nota en la figura 5.1, la implementación de FULL GP es más rápida que la aproximación FITC, esto es debido al costo de cálculos extras usados en la implementación de FITC.

Tabla 5.1: Tiempo de ejecución GP de única salida con $N <= 500$

Número de puntos (N)	Tiempo log_marginal (s)	Tiempo cálculos antes del gradiente (s)	Tiempo total gradiente (s)
100 fitc	0.0023613	0.00214263	0.0232438
200 fitc	0.00758566	0.00820048	0.0745419
300 fitc	0.0130925	0.0146854	0.136782
400 fitc	0.0207176	0.0216352	0.234722
500 fitc	0.0318134	0.0336362	0.35332
100 full	0.00340657	0.00313866	0.00657217
200 full	0.0152476	0.012867	0.0262469
300 full	0.0372075	0.0263854	0.0531106
400 full	0.0664606	0.0478779	0.094341
500 full	0.110781	0.0850772	0.12756

Figura 5.1: Tiempo de ejecución GP de única salida con $N <= 500$



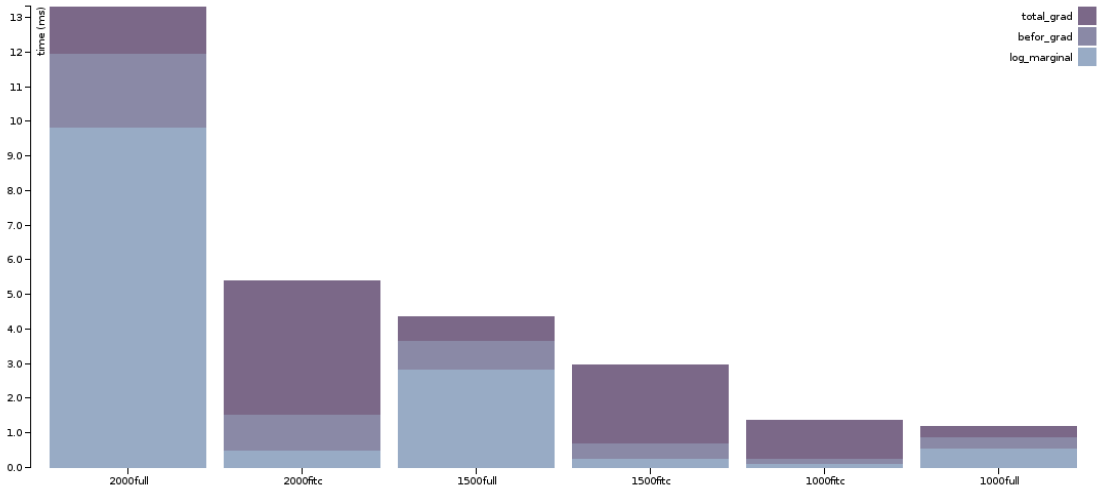
5.1.2 Caso 2: $N >= 1000$

En este caso, como se ve en la figura 5.2 la implementación de FITC es más rápida que la implementación del FULL GP. Basado en la complejidad teórica de ambas implementaciones se puede garantizar que esto seguirá siendo así para tallas más grandes de N.

Tabla 5.2: Tiempo de ejecución GP de única salida con $N >= 1000$

Número de puntos (N)	Tiempo log_marginal (s)	Tiempo cálculos antes del gradiente (s)	Tiempo total gradiente (s)
1000 fitc	0.113109	0.142562	1.1343
1500 fitc	0.272764	0.424732	2.29391
2000 fitc	0.502827	1.02285	3.88411
1000 full	0.57393	0.319724	0.326905
1500 full	2.83802	0.820963	0.712416
2000 full	9.83389	2.14142	1.35117

Figura 5.2: Tiempo de ejecución GP de única salida con $N >= 1000$



5.2 GP de múltiple salida

Para este caso se trabajó haciendo una regresión de 3 salidas, al igual que el caso anterior se utilizó $K = 30$ y diferentes valores de N.

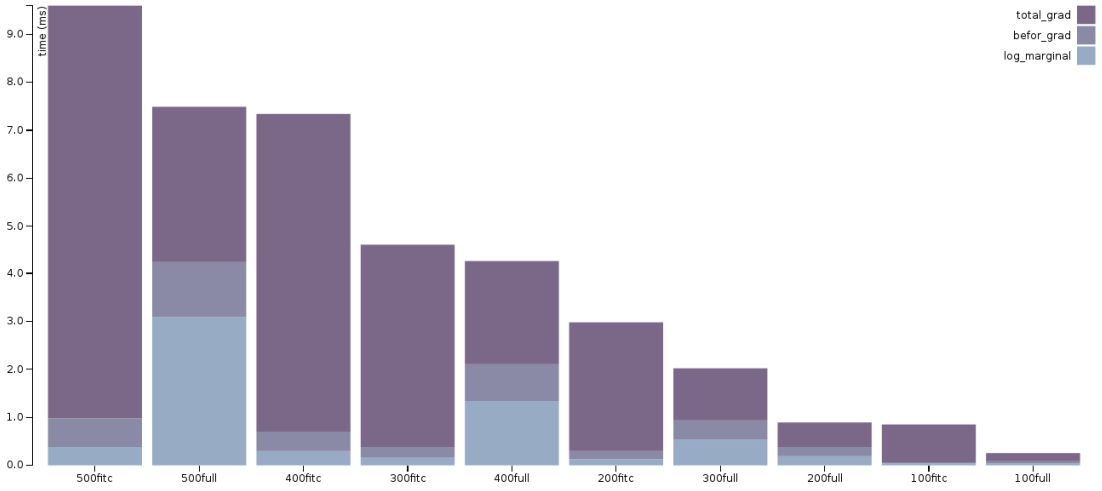
5.2.1 Caso 1: $N <= 500$

Como se puede ver en la figura 5.3 al igual que el caso de una única salida, en múltiples salidas la implementación de FULL GP es más rápida que la aproximación FITC para valores de $N <= 500$.

Tabla 5.3: Tiempo de ejecución GP de múltiple salida con $N <= 500$

Número de puntos (N)	Tiempo log_marginal (s)	Tiempo cálculos antes del gradiente (s)	Tiempo total gradiente (s)
100 fitc	0.0355823	0.0335358	0.776129
200 fitc	0.137224	0.154128	2.69992
300 fitc	0.173883	0.208453	4.22504
400 fitc	0.306353	0.397176	6.64015
500 fitc	0.385638	0.584745	8.62614
100 full	0.0350008	0.0427648	0.173566
200 full	0.193956	0.181947	0.512944
300 full	0.534696	0.400459	1.08983
400 full	1.34758	0.763713	2.15903
500 full	3.10189	1.13293	3.25302

Figura 5.3: Tiempo de ejecución GP de múltiple salida con $N <= 500$



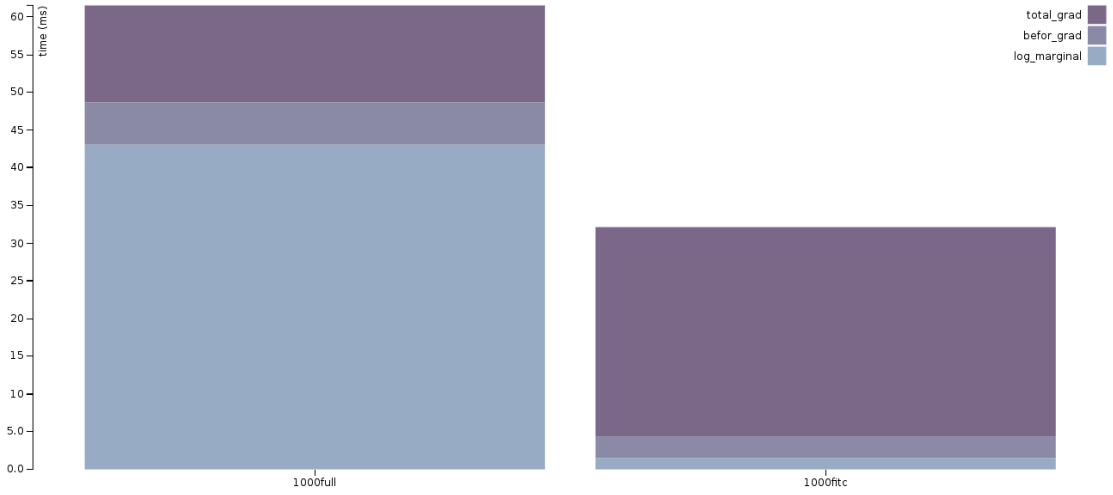
5.2.2 Caso 2: $N >= 1000$

De igual manera en la figura 5.8 se evidencia claramente que la aproximación FITC es más rápida que la implementación FULL GP para valores de $N >= 1000$. También se garantiza que esto se mantendrá para valores aún más grandes.

Tabla 5.4: Tiempo de ejecución GP de múltiple salida con $N >= 1000$

Número de puntos (N)	Tiempo log_marginal (s)	Tiempo cálculos antes del gradiente (s)	Tiempo total gradiente (s)
1000fitc	1.43872	2.89731	27.7408
1000full	43.0265	5.62635	12.8744

Figura 5.4: Tiempo de ejecución GP de múltiple salida con $N >= 1000$



5.3 Desempeño aproximación FITC sin entrenar los puntos ocultos

Las pruebas anteriores describen un escenario general en el cual siempre se deben entrenar los puntos ocultos, sin embargo es una comparación injusta entre la aproximación FITC y el FULL GP. En muchos casos basta con fijar los puntos ocultos de manera equidistante en el dominio de las entradas para obtener una buena predicción, en otros casos el usuario conoce de antemano cuales son los puntos ocultos adecuados y por lo tanto no es necesario optimizarlos.

Los siguientes gráficos muestran la comparación de desempeño del FULL GP y la aproximación FITC sin optimizar los puntos ocultos.

Figura 5.5: FITC SO sin optimizar puntos ocultos $N <= 500$

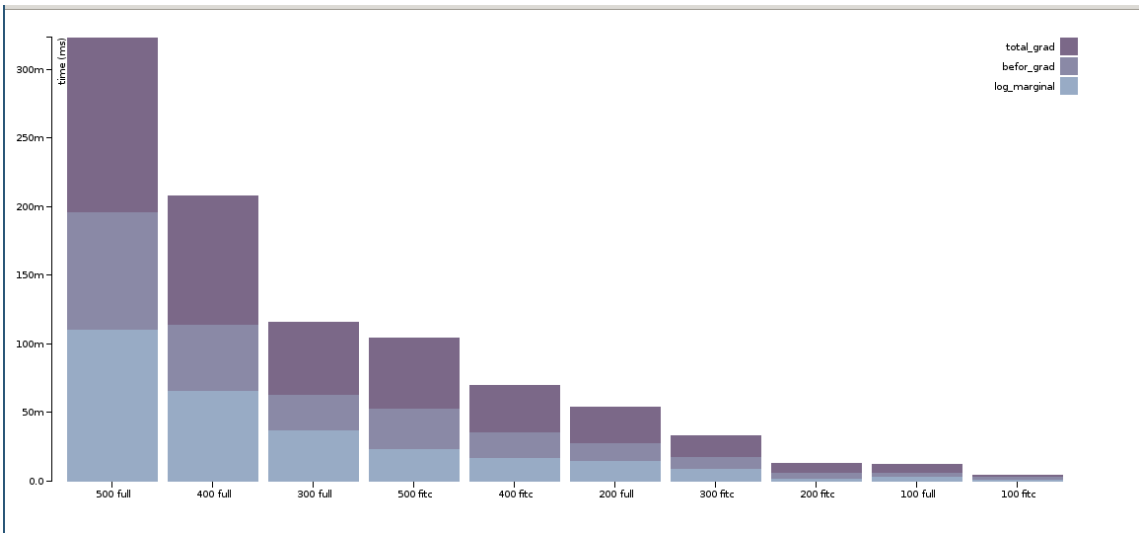


Figura 5.6: FITC SO sin optimizar puntos ocultos $N >= 1000$

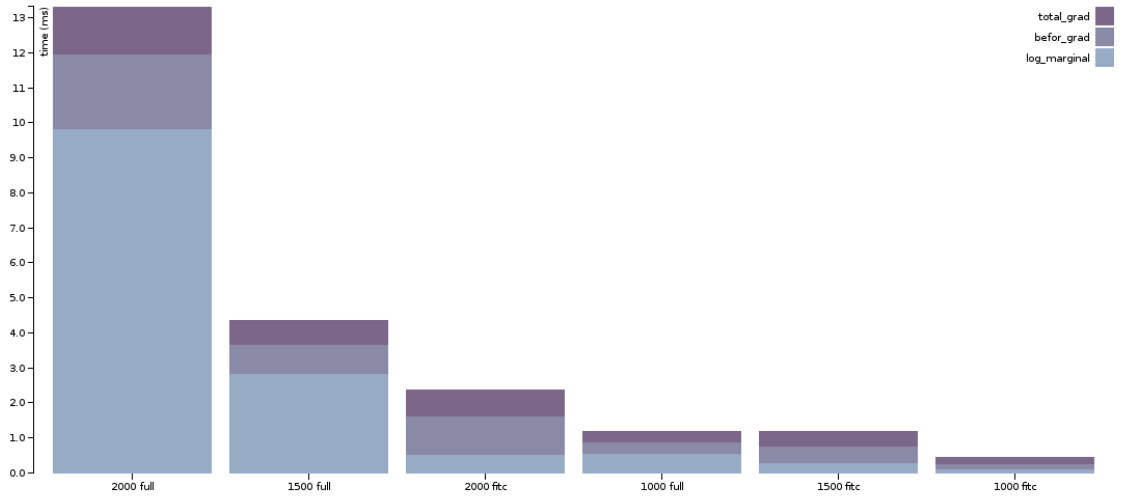


Figura 5.7: FITC MO sin optimizar puntos ocultos $N <= 500$

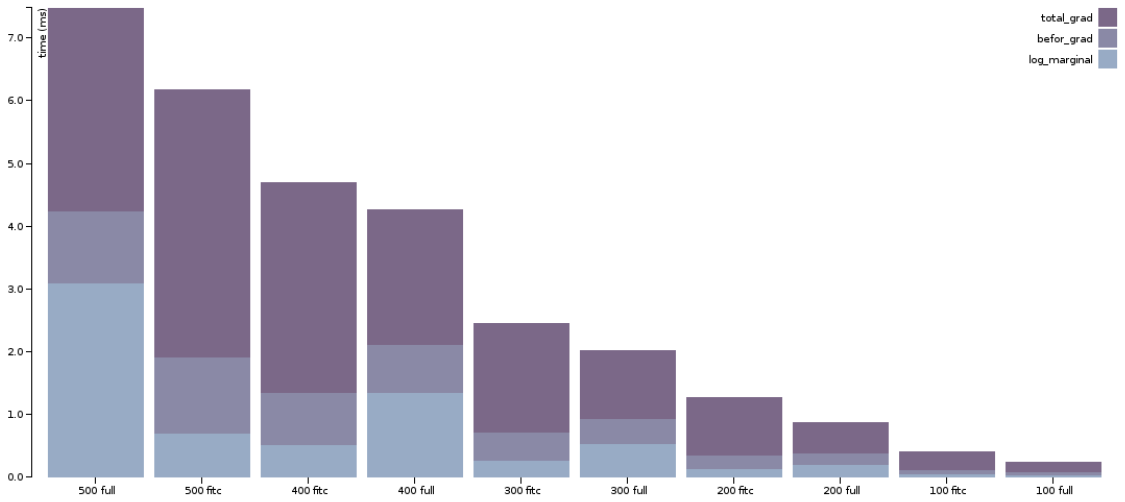
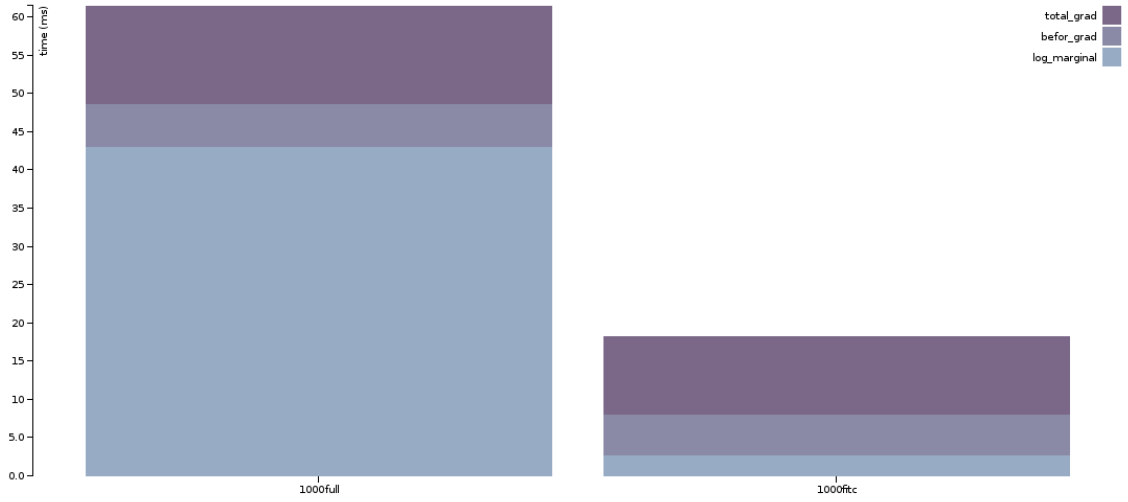


Figura 5.8: FITC MO sin optimizar puntos ocultos $N >= 1000$



Capítulo 6

Conclusiones

- Se implementó el proceso de regresión utilizando procesos Gaussianos, usando tanto el método tradicional, como la aproximación usando Fully Independent Training Conditional (FITC).
- Se implementaron todas las rutinas a modo de librería de fácil instalación, de manera que puede ser llamada desde cualquier código en C++ o se pueden crear "bindings", para ser utilizada desde otros lenguajes de programación y así ser fácilmente adicionada a otros proyectos.
- Para el desarrollo de la librería se utilizó el patrón de diseño Pointer to implementation (PIMPL), este permite realizar cambios internos a la Librería sin que el usuario tenga que recompilar los módulos que hacen uso de ella.
- Se proveen métodos sencillos para crear un kernel de múltiple salida y un objeto de regresión, de manera que el usuario pueda utilizar fácilmente la librería, pero además se proveen métodos más complejos para quienes necesiten mayor flexibilidad a la hora de modelar el proceso gaussiano.
- Para la construcción de la documentación de los métodos públicos de la librería se utilizó Doxygen, la cual es una herramienta para generar documentación de C++ utilizando comentarios en el código, con esta se creó la documentación en linea y los diagramas de clase. La API completa se puede consultar en `http://pin3da.github.io/gplib/`
- Para garantizar el correcto funcionamiento de la librería se realizaron pruebas para cada una de las clases utilizando la librería BOOST para C++, en las cuales se comprobaron los algoritmos más críticos para el funcionamiento de la librería, como lo son los cálculos de los gradientes, comparando la implementación de los analíticos contra los numéricos o la evaluación de los kernels para comprobar que las matrices obtenidas son positivas semidefinidas entre otras.
- Con el propósito de evaluar el desempeño de la librería, se realizó un conjunto de pruebas, evaluando los dos métodos de regresión, FULL GP y FITC, para las cuales se tomaron 3 mediciones diferentes, siendo función de costo, cálculos previos al gradiente y cálculos

del gradiente como es descrito en el capítulo 5, en las cuales se demostró que la implementación del FITC es asintóticamente más rápida que la implementación FULL GP.

- El proceso de entrenamiento sin tomar en cuenta los puntos ocultos y el proceso de predicción en la aproximación FITC son significativamente más rápidos que el FULL GP, tal como lo sugería su complejidad teórica.
- Gracias al uso de la librería Armadillo, GPlib corre utilizando múltiples hilos del procesador, lo cual es de gran conveniencia para cumplir las metas de desempeño.
- Debido a la forma en que se implemento la librería, es posible agregar de manera fácil nuevos kernels, tanto los de única salida como los de múltiples.
- La librería fue implementada de una manera muy limpia y tiene una guía de estilo para codificar, lo que permite que se pueda modificar de manera sencilla para que el usuario la adapte a sus necesidades.

6.1 Trabajos futuros

- **Mejorar el desempeño en el cálculo de la derivada Q_{ff}:** El cálculo de la ecuación 3.18, durante el proceso de entrenamiento de la aproximación FITC consume una importante cantidad de tiempo, debido a que tiene una complejidad de $O(N^2U)$, donde N es la cantidad de puntos observados y U la cantidad de puntos ocultos, por lo tanto, buscar una forma alternativa de calcularla o mejorar la actual con alguna propiedad matemática mejoraría de manera importante el desempeño de esta.
- **Implementar otras aproximaciones:** Además de la aproximación FITC vista en la sección 3.2 existen algunas otras muy atractivas en cuanto a complejidad computacional, como las vistas en [16], las cuales serían un aporte muy interesante para la librería.
- **Implementar clasificación:** Según lo descrito en la sección 1.1.2, los procesos gaussianos son muy utilizados para modelar clasificadores donde la entrada consiste en mas de una serie de tiempo, correlacionadas entre si, por lo tanto, esta característica en la librería sería muy atractiva para los usuarios.
- **Aceleración de algoritmos con computación heterogénea:** Ya que algunos algoritmos utilizados por la librería tienen una gran carga computacional, además de un gran volumen de datos a procesar y estos pueden ser realizados de manera paralela por su independencia, como, por ejemplo las multiplicaciones de matrices se da la posibilidad de paralelizarlas utilizando GPUs, clústers u otras técnicas de aceleración para mejorar significativamente el desempeño del algoritmo.

Apéndice A

Derivadas del LMC kernel

Debido a que B debe ser una matriz positiva semidefinida, es necesario redefinirla para poder optimizar libremente:

$$B^q = A^{qT} A^q$$

Ahora se optimizará con respecto a los parámetros de A

Para derivar el LMC kernel con respecto al parámetro a, b de la matriz A^q:

$$\forall_{i,j} \frac{\partial K_{i,j}}{\partial A_{a,b}} = \begin{cases} A^q_{a,b} K^q(X_i, Y_j) & \text{if } i = a \text{ and } j = b \\ A^q_{i,b} K^q(X_i, Y_j) & \text{if } j = a \\ A^q_{j,b} K^q(X_i, Y_j) & \text{if } i = a \end{cases}$$

Para derivar el LMC kernel con respecto al parámetro θ_p de los kernels internos que pertenecen a la matriz A^q:

$$\forall_{i,j} \frac{\partial K_{i,j}}{\partial \theta_p} = A^q \frac{\partial K^q(X_i, Y_j)}{\partial \theta_p}$$

Apéndice B

Derivadas con respecto a las entradas (puntos ocultos) para FITC

B1. Derivadas del kernel LMC

El kernel LMC puede ser descrito por la siguiente ecuación $\sum_{q=1}^{Q} \beta_q \otimes k_q(X_q, Y_q)$ donde β es un arreglo que contienen matrices de parámetros, k son cada uno de los kernels internos, y X y Y son arreglos que contienen matrices de entradas. Por consecuencia cada $\beta_q \otimes k_q(X_q, Y_q)$ es igual a:

$$\begin{bmatrix} \beta_{q(1,1)} k_q(X_q, Y_q) & \dots & \beta_{q(1,m)} k_q(X_q, Y_q) \\ \vdots & \ddots & \vdots \\ \beta_{q(n,1)} k_q(X_q, Y_q) & \dots & \beta_{q(n,m)} k_q(X_q, Y_q) \end{bmatrix}$$

La derivada de la matriz anterior con respecto a un parámetro arbitrario θ sería entonces:

$$\begin{bmatrix} \frac{\partial}{\partial \theta} \beta_{q(1,1)} k_q(X_q, Y_q) & \dots & \frac{\partial}{\partial \theta} \beta_{q(1,m)} k_q(X_q, Y_q) \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial \theta} \beta_{q(n,1)} k_q(X_q, Y_q) & \dots & \frac{\partial}{\partial \theta} \beta_{q(n,m)} k_q(X_q, Y_q) \end{bmatrix}$$

Pero como sabemos que β contiene solo parámetros (no contiene los puntos ocultos) y estamos derivando con respecto a las entradas (debido a que el kernel tomará los puntos ocultos como entradas), se puede ver como una "constante", el resultado de aplicar este concepto es:

$$\begin{bmatrix} \beta_{q(1,1)} \frac{\partial}{\partial \theta} k_q(X_q, Y_q) & \dots & \beta_{q(1,m)} \frac{\partial}{\partial \theta} k_q(X_q, Y_q) \\ \vdots & \ddots & \vdots \\ \beta_{q(n,1)} \frac{\partial}{\partial \theta} k_q(X_q, Y_q) & \dots & \beta_{q(n,m)} \frac{\partial}{\partial \theta} k_q(X_q, Y_q) \end{bmatrix}$$

B2. Derivada de los kernels internos (Squared Exponential)

El kernel interno utilizado (squared exponential) $k(X,Y)$ puede ser representado por la siguiente matriz:

$$\begin{bmatrix} \sigma^2 e^{\frac{(X_{row(1)}-Y_{row(1)})(X_{row(1)}-Y_{row(1)})^t}{2l}} & \dots & \sigma^2 e^{\frac{(X_{row(1)}-Y_{row(m)})(X_{row(1)}-Y_{row(m)})^t}{2l}} \\ \vdots & \ddots & \vdots \\ \sigma^2 e^{\frac{(X_{row(n)}-Y_{row(1)})(X_{row(n)}-Y_{row(1)})^t}{2l}} & \dots & \sigma^2 e^{\frac{(X_{row(n)}-Y_{row(m)})(X_{row(n)}-Y_{row(m)})^t}{2l}} \end{bmatrix}$$

La derivada del kernel interno respecto a un parámetro arbitrario θ sería entonces:

$$\begin{bmatrix} \frac{\partial}{\partial\theta}\sigma^2 e^{\frac{(X_{row(1)}-Y_{row(1)})(X_{row(1)}-Y_{row(1)})^t}{2l}} & \dots & \frac{\partial}{\partial\theta}\sigma^2 e^{\frac{(X_{row(1)}-Y_{row(m)})(X_{row(1)}-Y_{row(m)})^t}{2l}} \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial\theta}\sigma^2 e^{\frac{(X_{row(n)}-Y_{row(1)})(X_{row(n)}-Y_{row(1)})^t}{2l}} & \dots & \frac{\partial}{\partial\theta}\sigma^2 e^{\frac{(X_{row(n)}-Y_{row(m)})(X_{row(n)}-Y_{row(m)})^t}{2l}} \end{bmatrix}$$

Donde, de nuevo, considerando que σ no tiene relación con las entrados podemos reescribir de la siguiente manera:

$$\begin{bmatrix} \sigma^2 \frac{\partial}{\partial\theta} e^{\frac{(X_{row(1)}-Y_{row(1)})(X_{row(1)}-Y_{row(1)})^t}{2l}} & \dots & \sigma^2 \frac{\partial}{\partial\theta} e^{\frac{(X_{row(1)}-Y_{row(m)})(X_{row(1)}-Y_{row(m)})^t}{2l}} \\ \vdots & \ddots & \vdots \\ \sigma^2 \frac{\partial}{\partial\theta} e^{\frac{(X_{row(n)}-Y_{row(1)})(X_{row(n)}-Y_{row(1)})^t}{2l}} & \dots & \sigma^2 \frac{\partial}{\partial\theta} e^{\frac{(X_{row(n)}-Y_{row(m)})(X_{row(n)}-Y_{row(m)})^t}{2l}} \end{bmatrix}$$

Y

$$\frac{\partial}{\partial\theta} e^{\frac{(X_{row(i)}-Y_{row(j)})(X_{row(i)}-Y_{row(j)})^t}{2l}} = e^{\frac{(X_{row(i)}-Y_{row(j)})(X_{row(i)}-Y_{row(j)})^t}{2l}} \frac{\partial}{\partial\theta} \frac{(X_{row(i)}-Y_{row(j)})(X_{row(i)}-Y_{row(j)})^t}{2l}$$

De nuevo, $2l$ es una constante:

$$\frac{e^{\frac{(X_{row(i)}-Y_{row(j)})(X_{row(i)}-Y_{row(j)})^t}{2l}}}{2l} \frac{\partial}{\partial\theta}(X_{row(i)}-Y_{row(j)})(X_{row(i)}-Y_{row(j)})^t$$

Y Finalmente,

$$\frac{\partial}{\partial\theta}(X_{row(i)}-Y_{row(j)})(X_{row(i)}-Y_{row(j)})^t =$$

$$(\frac{\partial}{\partial\theta}(X_{row(i)}-Y_{row(j)}))(X_{row(i)}-Y_{row(j)})^t+(X_{row(i)}-Y_{row(j)})(\frac{\partial}{\partial\theta}(X_{row(i)}-Y_{row(j)})^t) =$$

$$(\frac{\partial}{\partial\theta}X_{row(i)}-\frac{\partial}{\partial\theta}Y_{row(j)})(X^t_{row(i)}-Y^t_{row(j)})+(X_{row(i)}-Y_{row(j)})(\frac{\partial}{\partial\theta}X^t_{row(i)}-\frac{\partial}{\partial\theta}Y^t_{row(j)}) =$$

$$\begin{aligned}(\frac{\partial}{\partial\theta}X_{row(i)})X^t_{row(i)}-(\frac{\partial}{\partial\theta}X_{row(i)})Y^t_{row(j)}-(\frac{\partial}{\partial\theta}Y_{row(j)})X^t_{row(i)}+(\frac{\partial}{\partial\theta}Y_{row(j)})Y^t_{row(j)}+\\ (X_{row(i)})\frac{\partial}{\partial\theta}X^t_{row(i)}-(X_{row(i)})\frac{\partial}{\partial\theta}Y^t_{row(j)}-(Y_{row(j)})\frac{\partial}{\partial\theta}X^t_{row(i)}+(Y_{row(j)})\frac{\partial}{\partial\theta}Y^t_{row(j)}\end{aligned}$$

Pero, como se está tratando con vectores fila, y, en general $AB^t = BA^t$ (donde A y B son, obviamente vectores fila), podemos reagrupar términos así:

$$2(\frac{\partial}{\partial\theta}X_{row(i)})X^t_{row(i)}-2(\frac{\partial}{\partial\theta}X_{row(i)})Y^t_{row(j)}-2(\frac{\partial}{\partial\theta}Y_{row(j)})X^t_{row(i)}+2(\frac{\partial}{\partial\theta}Y_{row(j)})Y^t_{row(j)}$$

Lo cual es equivalente a:

$$2((\frac{\partial}{\partial\theta}X_{row(i)})X^t_{row(i)}-(\frac{\partial}{\partial\theta}X_{row(i)})Y^t_{row(j)}-(\frac{\partial}{\partial\theta}Y_{row(j)})X^t_{row(i)}+(\frac{\partial}{\partial\theta}Y_{row(j)})Y^t_{row(j)})$$

Y reemplazando en la ecuación original:

$$\begin{aligned}\frac{e^{\frac{(X_{row(i)}-Y_{row(j)})(X_{row(i)}-Y_{row(j)})^t}{2l}}}{2l}2((\frac{\partial}{\partial\theta}X_{row(i)})X^t_{row(i)}-(\frac{\partial}{\partial\theta}X_{row(i)})Y^t_{row(j)}-\\ (\frac{\partial}{\partial\theta}Y_{row(j)})X^t_{row(i)}+(\frac{\partial}{\partial\theta}Y_{row(j)})Y^t_{row(j)})\end{aligned}$$

Finalmente obtenemos:

$$\begin{aligned}\frac{e^{\frac{(X_{row(i)}-Y_{row(j)})(X_{row(i)}-Y_{row(j)})^t}{2l}}}{l}((\frac{\partial}{\partial\theta}X_{row(i)})X^t_{row(i)}-(\frac{\partial}{\partial\theta}X_{row(i)})Y^t_{row(j)}-\\ (\frac{\partial}{\partial\theta}Y_{row(j)})X^t_{row(i)}+(\frac{\partial}{\partial\theta}Y_{row(j)})Y^t_{row(j)})\end{aligned}$$

Bibliografía

[1] C4.1 - collective code construction contract. http://rfc.zeromq.org/spec:22. Accedido: 2016-01-20.

[2] GPlib - multi output gaussian process library. https://github.com/pin3da/gplib. Accedido: 2016-01-20.

[3] Mauricio Alvarez and Neil D Lawrence. Sparse convolved gaussian processes for multi-output regression. In *Advances in neural information processing systems*, pages 57–64, 2009.

[4] Mauricio A Alvarez and Neil D Lawrence. Computationally efficient convolved multiple output gaussian processes. *The Journal of Machine Learning Research*, 12:1459–1500, 2011.

[5] Ben-Yishai Assaf and Burshtein David. A discriminative training algorithm for hidden markov models. *Speech and Audio Processing, IEEE Transactions*.

[6] The GPy authors. GPy: A gaussian process framework in python. http://github.com/SheffieldML/GPy, 2012–2014.

[7] Wu Hou Biing-Hwang Juang and Chin-Hui Lee. Minimum classification error rate methods for speech recognition. *Speech and Audio Processing, IEEE Transactions*.

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[9] Lawrence N. D. Probabilistic non-linear principal component analysis with gaussian process latent variable models. 2004.

[10] Morris L. Eaton. *Multivariate Statistics : A Vector Space Approach*. Institute of Mathematical Statistics, Ohio, USA, 2007.

[11] Sebastián Gómez. Classifcation of times series with multiple output gaussian process and mce. 2014.

[12] Carnegie Mellon Graphics Lab. MOCAP: Carnegie mellon university motion capture database. http://mocap.cs.cmu.edu/.

[13] Dr Neil Lawrence. Gaussian process models for visualisation of high dimensional data. 2004.

[14] Seeger M. Lawrence N. D. and Herbrich R. Fast sparse gaussian process methods: the informative vector machine. 2003.

[15] Patrick Lucey, Jeffrey F Cohn, Takeo Kanade, Jason Saragih, Zara Ambadar, and Iain Matthews. The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, pages 94–101. IEEE, 2010.

[16] Joaquin Quinonero-Candela, Carl Edward Rasmussen, and Christopher KI Williams. Approximation methods for gaussian process regression. *Large-scale kernel machines*, pages 203–223, 2007.

[17] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[18] Carl Edward Rasmussen and Hannes Nickisch. Gaussian processes for machine learning (gpml) toolbox. *The Journal of Machine Learning Research*, 11:3011–3015, 2010.

[19] Luengo D. Álvarez M., Lawrence N. D. and Titsias M. K. Variational inducing kernels for sparse convolved multiple output gaussian processes. 2009.