
APRENDIZAJE DE MÁQUINAS

Esta versión: 7 de abril de 2024

Última versión: github.com/GAMES-UChile/Curso-Aprendizaje-de-Maquinas

Felipe Tobar
Centro de Modelamiento Matemático
Universidad de Chile

ftobar@dim.uchile.cl
www.dim.uchile.cl/~ftobar

Prefacio

Este apunte es una versión extendida y detallada de las notas de clase utilizadas en el curso MDS7104: APRENDIZAJE DE MÁQUINAS (ex MA5203 y MA5204) dictado anualmente en el *Master of Data Science* de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile entre 2016 y 2024. El objetivo principal de este apunte es presentar material autocontenido y original de las temáticas vistas en el curso tanto para apoyar su realización como para estudio personal de quien lo requiera. Debido a que los contenidos del curso van variando año a año, el apunte está en constante modificación, por esta razón hay secciones de este documento que pueden estar incompletas en cuanto a formato, figuras o contenidos. Sin embargo, creo que el hacer disponible este apunte en desarrollo puede ser un aporte para los alumnos del curso MDS7104 como a la comunidad en general.

El desarrollo de este apunte solo ha sido posible gracias a la contribución de varios integrantes del cuerpo académico de los cursos MA5203: APRENDIZAJE DE MÁQUINAS PROBABILÍSTICO (2016-2018), MA5309: APRENDIZAJE DE MÁQUINAS AVANZADO (2016, 2018, 2020 y 2022), MA5204: APRENDIZAJE DE MÁQUINAS (2019-2021) y MDS7104: APRENDIZAJE DE MÁQUINAS (2022-2024). Me gustaría reconocer la indispensable contribución de ayudantes y participantes de estos cursos, tanto en el desarrollo del curso mismo, ideas de tareas y clases auxiliares, producción de figuras, ejemplos, y mucho más. En orden de *aparición*, gracias: Gonzalo Ríos, Camilo Carvajal, Cristóbal Silva, Alejandro Cuevas, Alejandro Veragua, Cristóbal Valenzuela, Mauricio Campos, Lerko Araya, Nicolás Aramayo, Mauricio Araneda, Mauricio Romero, Luis Muñoz, Jou-Hui Ho, Diego Garrido, José Díaz, Francisco Vásquez, Fernando Fetis, Nelson Moreno, Arie Wortsman, Víctor Faraggi, David Molina, Tomás Valencia, Alonso Vargas, Fernando Fétis, Augusto Tagle, Juan Cuevas. Sin la participación de todos ustedes, el éxito de los cursos mencionados y la composición de este apunte no habría sido posible.

Felipe Tobar
7 de abril de 2024
Santiago, Chile

Índice

| | |
|---|-----------|
| 1. Introducción | 7 |
| 1.1. Orígenes: inteligencia artificial | 7 |
| 1.2. Breve historia del aprendizaje de máquinas | 8 |
| 1.3. Taxonomía del aprendizaje de máquinas | 9 |
| 1.4. Relación con otras disciplinas | 10 |
| 1.5. Estado del aprendizaje de máquinas y desafíos | 11 |
| 2. Regresión | 13 |
| 2.1. Regresión lineal | 13 |
| 2.1.1. Mínimos cuadrados | 14 |
| 2.1.2. Regularización: ajuste versus generalización | 18 |
| 2.2. Máxima verosimilitud | 24 |
| 2.2.1. Elección del modelo de acuerdo a la máxima verosimilitud | 27 |
| 2.2.2. Maxima verosimilitud del modelo lineal gaussiano | 29 |
| 2.3. Regresión vía inferencia bayesiana | 30 |
| 2.3.1. ¿Qué es ser bayesiano? | 31 |
| 2.3.2. Elección de prior: conjugación | 33 |
| 2.3.3. Maximo a posteriori | 39 |
| 2.4. Predicciones | 42 |
| 2.5. Regresión no lineal | 44 |
| 2.5.1. Modelo lineal en los parámetros | 46 |
| 2.5.2. Ejemplo de transformaciones | 47 |
| 3. Clasificación | 50 |
| 3.1. kecinos más cercanos (KNN) | 50 |
| 3.2. Clasificación lineal | 51 |
| 3.2.1. Ajuste mediante mínimos cuadrados | 52 |
| 3.2.2. El perceptrón | 55 |
| 3.3. Clasificación probabilística: modelo generativo | 58 |
| 3.3.1. Regresión logística | 59 |
| 3.3.2. Regresión logística v/s modelo generativo | 61 |
| 4. Selección y evaluación de modelos | 63 |
| 4.1. Descomposición sesgo-varianza | 63 |
| 4.2. Validación cruzada | 65 |
| 4.2.1. Validación cruzada exhaustiva | 65 |
| 4.2.2. Validación cruzada no exhaustiva | 65 |
| 4.3. Selección de modelo | 66 |
| 4.3.1. Criterio de información de Akaike (AIC) | 66 |
| 4.3.2. Criterio de información bayesiano (BIC) | 67 |
| 4.4. Evaluación de modelos | 67 |
| 4.4.1. Error cuadrático medio | 67 |
| 4.4.2. log-densidad predictiva o log-verosimilitud | 68 |
| 4.5. Promedio de modelos | 68 |

| | |
|--|------------|
| 5. Support-vector machines | 70 |
| 5.1. Idea general | 70 |
| 5.2. Formulación del problema | 71 |
| 5.3. Margen suave | 75 |
| 5.4. Método de kernel | 77 |
| 5.4.1. Kernel ridge regression | 80 |
| 5.4.2. Kernel SVM | 82 |
| 6. Modelos de función de base adaptativa | 85 |
| 6.1. Introducción | 85 |
| 6.2. Árboles | 85 |
| 6.2.1. Motivación con caso regresión | 85 |
| 6.2.2. Algoritmo <i>CART</i> | 85 |
| 6.2.3. Criterios de corte para clasificación | 87 |
| 6.2.4. Evitar sobreajuste: detención temprana y poda | 88 |
| 6.2.5. Interpretabilidad | 90 |
| 6.3. Bagging | 91 |
| 6.3.1. Método Bootstrapping | 91 |
| 6.3.2. Bagging: agregación de modelos | 92 |
| 6.3.3. Bosques aleatorios y variaciones | 93 |
| 6.3.4. Árboles extremadamente aleatorios | 94 |
| 6.4. Boosting | 95 |
| 6.4.1. Motivación: algoritmos fuertes versus débiles | 95 |
| 6.4.2. Algoritmo <i>AdaBoost</i> | 95 |
| 6.4.3. Modelamiento aditivo por etapas | 97 |
| 6.4.4. <i>GradientBoosting</i> : boosting como descenso de gradiente funcional | 98 |
| 6.4.5. Aspectos prácticos | 99 |
| 6.4.6. Interpretabilidad de modelos basados en árboles | 100 |
| 6.4.7. Más pérdidas y variaciones | 100 |
| 7. Aprendizaje no supervisado | 102 |
| 7.1. Reducción de dimensionalidad | 102 |
| 7.1.1. Análisis de componentes principales (PCA) | 102 |
| 7.1.2. Kernel PCA | 104 |
| 7.1.3. Probabilistic PCA | 104 |
| 7.1.4. Discriminante lineal de Fisher | 106 |
| 7.1.5. Compressed Sensing | 108 |
| 7.1.6. PCA vs. Compressed Sensing | 109 |
| 7.1.7. Otros métodos de reducción de dimensionalidad | 109 |
| 7.2. Clustering | 110 |
| 7.2.1. Hierarchical clustering (HCA) | 110 |
| 7.2.2. <i>kmeans</i> | 111 |
| 7.2.3. Modelo de mezcla de gaussianas (GMM) | 112 |
| 7.2.4. Density-based spatial clustering of applications with noise (DBSCAN) | 115 |
| 7.2.5. Aprendizaje Semi-Supervisado | 117 |
| 7.2.6. Otros Algoritmos de Clustering | 117 |

| | |
|---|------------|
| 8. Redes Neuronales | 118 |
| 8.1. Introducción y arquitectura | 118 |
| 8.1.1. Conceptos básicos | 118 |
| 8.1.2. El perceptrón y funciones de activación | 119 |
| 8.1.3. Arquitectura de una red neuronal | 120 |
| 8.1.4. Función de costos y unidades de output | 121 |
| 8.2. Entrenamiento de una red neuronal | 121 |
| 8.2.1. Forward Propagation | 121 |
| 8.2.2. Backward Propagation - Preliminares | 122 |
| 8.2.3. Backward Propagation - Capas ocultas | 123 |
| 8.2.4. Backward Propagation - Capa de output | 124 |
| 8.3. Regularización | 124 |
| 8.3.1. Regularización L^2 | 125 |
| 8.3.2. Dropout | 125 |
| 8.3.3. Otros métodos de regularización | 126 |
| 8.4. Algoritmos de optimización | 127 |
| 8.4.1. Minibatch | 127 |
| 8.4.2. Algoritmos con momentum | 127 |
| 8.4.3. Algoritmos con learning rate adaptativos | 128 |
| 8.5. Obstáculos en el entrenamiento de redes neuronales | 128 |
| 8.5.1. Valores Iniciales | 128 |
| 8.5.2. Overfitting | 129 |
| 8.5.3. Cantidad de neuronas en las capas escondidas | 129 |
| 8.6. Deep Learning y otros tipos de redes neuronales | 129 |
| 8.6.1. Redes neuronales convolucionales | 129 |
| 8.6.2. Redes neuronales recurrentes | 132 |
| 8.6.3. Autoencoders | 134 |
| 8.6.4. Redes generativas adversariales | 135 |
| 9. Procesos gaussianos | 136 |
| 9.1. Muestreo de un prior \mathcal{GP} | 137 |
| 9.2. Incorporando información | 138 |
| 9.2.1. Evaluación sin ruido | 138 |
| 9.2.2. Evaluación con ruido | 139 |
| 9.3. Entrenamiento y optimización de un \mathcal{GP} | 140 |
| 9.3.1. ¿Qué es entrenar un \mathcal{GP} ? | 140 |
| 9.3.2. ¿Cómo se entrena un \mathcal{GP} ? | 142 |
| 9.3.3. Complejidad computacional | 143 |
| 9.4. Funciones de covarianza (kernels) | 143 |
| 9.4.1. Rational Quadratic (RQ) | 143 |
| 9.4.2. Kernel periódico | 144 |
| 9.4.3. Operaciones con kernels | 144 |
| 9.4.4. Representación espectral | 145 |
| 9.5. Extensiones para un \mathcal{GP} | 145 |
| 9.5.1. \mathcal{GP} de clasificación | 145 |
| 9.5.2. Selección automática de relevancia (ARD) (<i>Selección automática de features</i>) | 145 |
| 9.5.3. Multi output \mathcal{GP} | 146 |
| 9.6. Diferentes interpretaciones de un \mathcal{GP} | 146 |
| 9.6.1. De regresión lineal a \mathcal{GP} | 146 |
| 9.6.2. Nota sobre RKHS | 147 |

| | |
|--|------------|
| 10. Anexos | 148 |
| 10.1. ¿Qué hizo efectivamente Bayes y por qué? | 148 |
| 10.2. Álgebra lineal | 149 |
| 10.2.1. Cálculo matricial | 149 |
| 10.2.2. Rango e inversa de Moore-Penrose | 151 |
| 10.2.3. Fórmula de Woodbury | 151 |
| 10.3. Optimización | 152 |
| 10.3.1. Teorema de Karush-Khun-Tucker | 152 |
| 10.3.2. Método del gradiente clásico | 152 |
| 10.3.3. Dualidad lagrangiana | 153 |
| Referencias | 155 |

1. Introducción

El aprendizaje de máquinas (AM) es una disciplina que reúne elementos de ciencias de la computación, optimización, estadística, probabilidades y ciencias cognitivas para construir el motor de aprendizaje dentro de la Inteligencia Artificial. Definido por Arthur Samuel en 1950, el AM es la disciplina que da a las máquinas la habilidad de aprender sin ser explícitamente programadas. Si bien existen enfoques al AM inspirados en sistemas biológicos, esta no es la única forma de construir métodos de aprendizaje: una analogía se puede identificar en los primeros intentos por construir máquinas voladoras, en donde se pretendía replicar el movimiento de las alas de un pájaro, sin embargo, estos intentos no fueron exitosos y no fue sino hasta la invención del primer avión, el cual no mueve sus alas, que el hombre logró construir la primera máquina voladora. Esto sugiere que el paradigma biológico no es exclusivo al momento de construir máquinas inteligentes que utilicen observaciones de su entorno para extraer información útil y generar predicciones.

AM es una disciplina joven que ha experimentado un vertiginoso crecimiento en las últimas décadas, esto ha sido posible en gran parte gracias a los recientes avances computacionales y la cantidad de datos que permite entrenar modelos complejos. El uso masivo de técnicas AM se ha visto reflejado en distintas áreas que incluyen visión computacional, clasificación de secuencias de ADN, marketing, detección de fraude, diagnósticos médicos, análisis financiero y traducción de texto por nombrar algunas. Adicionalmente, si bien el objetivo de AM es desarrollar algoritmos de aprendizaje prescindiendo en gran medida de la intervención humana, otra razón del éxito del AM es su facilidad para acoplarse con otras disciplinas aplicadas, en particular al área que hoy conocemos como *Data Science*.

1.1. Orígenes: inteligencia artificial

El hecho de que nuestras habilidades cognitivas nos diferencien fuertemente del resto de las especies del reino animal es la principal prazaón del dominio del *Homo sapiens* sobre el planeta: la inteligencia humana superior a la del resto de los animales nos permite adaptarnos a diferentes situaciones ambientales y sociológicas de forma rápida y sin la necesidad de un cambio evolutivo. Por ejemplo, para migrar desde África al norte de Europa y a Oceanía, el *Homo sapiens* no necesitó adaptarse biológicamente a climas distintos, sino que manipuló herramientas y materiales para producir vestimenta adecuada y embarcaciones. Otra característica única del *Homo sapiens* es su habilidad de creer en un imaginario colectivo que permite construir organizaciones con un gran número de individuos, lo cual es exclusivo a nuestra especie y consecuencia de nuestra inteligencia (Harari, 2015).

Nuestro interés en entender la inteligencia puede ser identificado desde los comienzos de la Filosofía y Sicología, disciplinas que se han dedicado al estudio de la forma en que entendemos, recordamos, razonamos y aprendemos. La inteligencia artificial (IA) es una disciplina mucho más reciente que las anteriores y va un paso más allá de la mera comprensión de la inteligencia, pues apunta a replicarla e incluso mejorarla (Bostrom, 2014). Hay varias definiciones de IA dependiendo de si (i) adoptamos un punto de vista cognitivo o conductual, o bien si (ii) identificamos las acciones inteligentes como humanas u objetivamente racionales. Una de estas posiciones es la de Alan Turing, el que mediante el juego de la imitación (Turing, 1950), sentencia que una máquina es inteligente si es capaz de desarrollar tareas cognitivas a un nivel “humano” suficiente para engañar a un interrogador (también humano). En este contexto para que una máquina sea inteligente, o equivalentemente, apruebe el test de Turing, es necesario que posea (Russell y Norvig, 2009):

- **representación del conocimiento** para guardar información recibida antes y durante la interrogación,
- **procesamiento de lenguaje natural** para comunicarse con seres humanos, en particular, el interrogador,

- **razonamiento automático** para usar la información guardada y formular respuestas y conclusiones, y
- **aprendizaje de máquinas** para adaptarse a nuevas circunstancias y descubrir patrones.

El test de Turing sigue siendo un tópico de investigación en Filosofía hasta el día de hoy, sin embargo, los avances actuales de la inteligencia artificial no están necesariamente enfocados en diseñar máquinas para aprobar dicho test. Si bien los inicios de la IA están en la Filosofía, actualmente los avances en IA se enfocan en ambientes controlados, donde apuntan a desarrollar metodologías para extraer información de bases de datos en situaciones que el operador humano tiene limitaciones de velocidad, capacidad o eficiencia. Estos esfuerzos de la IA no necesariamente replican el actuar del humano, en particular, el componente de aprendizaje de máquinas en la lista anterior ha jugado un papel fundamental en esta nueva etapa, en donde su aporte en distintas áreas de aplicación es cada vez más evidente.

1.2. Breve historia del aprendizaje de máquinas

En base a las definiciones de la inteligencia de máquinas asentadas por Turing en 1950, las primeras redes neuronales artificiales comenzaron aemerger en los trabajos seminales de (Minsky, 1952) que programó el primer simulador de una red neuronal, (Farley y Clark, 1954) que implementaron un algoritmo de prueba y error para el aprendizaje, y (Rosenblatt, 1958) que propuso el Perceptrón. En los años siguientes la investigación en redes neuronales se vio afectada por las limitaciones de dichas estructuras expuestas en (Minsky y Papert, 1969) dando origen a lo que es conocido como el primer invierno de la inteligencia artificial, en donde el Profesor Sir James Lighthill expuso frente al parlamento inglés que la inteligencia artificial se fijaba objetivos no realistas y solo servía para escenarios básicos (Lighthill, 1973). Esta desconfianza en los alcances de la IA ralentizó su desarrollo, sin embargo, los *conexionistas*¹ seguirían investigando sobre formas de diseñar y entrenar redes neuronales. Específicamente, los resultados de (Werbos, 1974) que culminarían en el algoritmo de *backpropagation* propuesto por (Rumelhart, Hinton, y Williams, 1986) y los avances de (Hopfield, 1982) en redes neuronales recurrentes permitirían terminar con el primer invierno de la inteligencia artificial.

Durante el receso del conexionismo proliferaron los sistemas basados en reglas, en particular, los sistemas experto compuestos por una serie de reglas condicionales “si-entonces” (if-then), las cuales replican el comportamiento de un humano experto, estos métodos se convirtieron en la primera herramienta exitosa de la IA en aplicaciones reales. Sin embargo, los sistemas experto no aprenden por sí solos, en el sentido de que las reglas “si-entonces” deben ser explícitamente programadas por un humano. Este enfoque tiene un costo considerable dependiendo de la complejidad del problema en cuestión, por lo que hacia el comienzo de la década de los 90s los sistemas experto colapsaron debido a que la cantidad de información disponible aumentaba y dicho enfoque no es “escalable”. Un sistema basado en reglas que aún se utiliza son los llamado árboles de decisión (Breiman, Friedman, Olshen, y Stone, 1984), los cuales difieren de los sistemas experto en que las reglas no son definidas por un humano sino que descubiertas en base a la elección de variables que mejor segmentan los datos de forma supervisada.

Las redes neuronales vieron un resurgimiento en la década de los 80s con el método de *backpropagation*, el cual permitía entrenar redes neuronales de más de dos capas usando la regla de la cadena. Esto permitió finalmente validar la premisa conexionista en tareas complejas, específicamente en reconocimiento de caracteres usando redes convolucionales como el Neocognitron (Fukushima, 1980) y LeNet-5 (LeCun y cols., 1989), y reconocimiento de voz usando redes neuronales recurrentes (Hopfield, 1982). Posterior a esta validación experimental, vino una segunda caída del conexionismo hacia fines de los 80s, esta se debió a la inexistencia de una clara teoría que explicara el desempeño de las redes neuronales, su tendencia a sobreajustar y su elevado costo computacional. A principios de los 90s, y basado en la teoría

¹Es decir, los partidarios del *conexionismo*, enfoque en el que los fenómenos mentales pueden ser explicados mediante la conexión de elementos más simples, e.g., *neuronas*. En el contexto de IA o AM, nos referimos coloquialmente a los *conexionistas* como los partidarios de las redes neuronales.

del aprendizaje estadístico (Vapnik, V., and Chervonenkis, A. , 1971), surgieron los métodos basados en *kernels* (o núcleos), específicamente las máquinas de soporte vectorial (MSV) (Boser, Guyon, y Vapnik, 1992). Esta nueva clase de algoritmos estaba fundamentada en una base teórica que combinaba elementos de estadística y análisis funcional, para caracterizar los conceptos de sobreajuste, optimalidad de soluciones y funciones de costo en el contexto del aprendizaje de máquinas. Además de sus fundamentos teóricos, las MSV mostraron ser una alternativa competitiva a las redes neuronales en cuanto a su desempeño y costo computacional en distintas aplicaciones.

También en la década de los 90s, surgieron nuevos enfoques de AM en donde el manejo de incertidumbre era abordado usando teoría de probabilidades, este es probablemente el punto de mayor similitud entre AM y Estadística (Ghahramani, 2015). Este enfoque procede definiendo una clase de modelos probabilísticos, es decir, se asume que los datos observados han sido generados por un modelo incierto y aleatorio, luego, el aprendizaje consiste en identificar dicho modelo usando Estadística bayesiana. Este es un enfoque elegante y teóricamente sustentado que permitió reinterpretar enfoques anteriores, sin embargo, muchas veces la formulación probabilística no puede ser resuelto de forma exacta y es necesario considerar aproximaciones basadas en Monte Carlo (Neal, 1993) o métodos variacionales (Jordan, Ghahramani, Jaakkola, y Saul, 1999). El enfoque bayesiano permite definir todos los elementos del problema de aprendizaje (modelos, parámetros y predicciones) mediante distribuciones de probabilidad con la finalidad de caracterizar la incertidumbre en el modelo y definir intervalos de confianza en las predicciones, esto incluso permite hacer inferencia sobre modelos con infinitos parámetros (Hjort, Holmes, Müller, y Walker, 2010). En estos casos, el espacio de parámetros pueden ser todas las posibles soluciones de un problema de aprendizaje, por ejemplo, el conjunto de las funciones continuas en regresión (Rasmussen, C. E., and Williams, C. K., 2006), o el conjunto de todas las distribuciones de probabilidad en el caso de estimación de densidades (Ferguson, 1973).

Un nuevo resurgimiento de las redes neuronales (RN) se vio en los primeros años de la década del 2000, donde el área se renombró como *deep learning* (Bengio, 2009). Progresivamente, el foco de la comunidad migró desde temáticas probabilistas o basadas en kernels para volver a RN pero ahora con un mayor número de capas. El éxito del enfoque conexionista finalmente logró objetivos propuestos hace décadas, principalmente por dos factores: (i) la gran cantidad de datos disponibles, e.g., la base de datos ImageNet (Deng y cols., 2009), y (ii) la gran capacidad computacional y parallelización del entrenamiento mediante el uso de tarjetas gráficas, lo cual permitió finalmente implementar RNs con billones de parámetros con técnicas para evitar el sobreajuste. El hecho que los parámetros pierdan significado para el entendimiento de las relaciones entre los datos aleja al AM de la Estadística, donde el objetivo es netamente predictivo y no la inferencia estadística: hasta el momento no hay otro enfoque que supere a *deep learning* en variadas aplicaciones. Este fenómeno ha sido principalmente confirmado por los avances de Google, DeepMind y Facebook. De acuerdo a Max Welling, si bien la irrupción de *deep learning* aleja al AM de la Estadística, aún hay temáticas que se nutren de ambas áreas como programación probabilista y computación bayesiana aproximada (Welling, 2015). Adicionalmente, Yoshua Bengio sentencia que aún hay muchos aspectos inciertos de *deep learning* en los cuales los estadísticos podrían ayudar, tal como los especialistas de las ciencias de la computación se han dedicado a los aspectos estadísticos del aprendizaje de máquinas en el pasado (Bengio, 2016).

1.3. Taxonomía del aprendizaje de máquinas

Las herramientas mencionadas en el apartado anterior (árboles, reglas, redes, kernels, modelos estadísticos), son transversales a los tipos de aprendizaje. A grueso modo, existen tres tipos de AM: supervisado, no-supervisado y reforzados. El aprendizaje supervisado (AS), considera datos en forma de pares (dato, etiqueta) y el objetivo es estimar una función $f(\cdot)$ tal se tiene la siguiente igualdad

$$\text{etiqueta} = f(\text{dato}) \quad (1.1)$$

o bien lo más cercano posible de acuerdo a una medida de error apropiada. El nombre supervisado viene del hecho que los datos disponibles están “etiquetados”, y por ende es posible supervisar el entrenamiento (o ajuste) del método. Ejemplos de AS son la identificación de *spam* en correos electrónicos (clasificación), como también la estimación del precio de una propiedad en función de su tamaño, ubicación y otras características (regresión). Ambos casos requieren de un conjunto de entrenamiento (datos etiquetados) construido por un humano. La segunda categoría es el aprendizaje no supervisado (AnS), en donde los datos no están etiquetados y el objetivo es encontrar estructura entre ellos, esto puede ser agrupar subconjuntos de datos que tienen algún grado de relación o propiedades en común, como realizar *clustering* de artículos en distintas grupos basado en su frecuencia aparición de palabras en un texto (Salakhutdinov, 2006). La tercera categoría del AM es el aprendizaje reforzado (AR), en la cual un agente (sintético) aprende a tomar decisiones mediante la maximización de un funcional de recompensa, este es probablemente el tipo de aprendizaje más cercano a la forma en que los animales aprendemos, mediante prueba y error. Un ejemplo de AR es en el cual entrenamos un perro para que aprenda algún truco recompensándolo con comida cada vez realiza la tarea correctamente. El reciente resultado de DeepMind donde una máquina aprendió a jugar Go usando una búsqueda de árbol y una red neuronal profunda es uno de los ejemplos más exitosos de este aprendizaje reforzado (Silver y cols., 2016). Es posible identificar categorías o subdivisiones de AM adicionales a las anteriormente descritas, estas incluyen aprendizaje continuo, activo, semi-supervisado, multi-tarea, etc.

1.4. Relación con otras disciplinas

AM y Estadística. En el análisis de datos es posible identificar dos extremos: el aprendizaje inductivo en donde los datos son abundantes, los supuestos a priori sobre su naturaleza son vagos y el objetivo es realizar predicciones con métodos sofisticados que no necesariamente expliquen los datos. En el otro extremo está el aprendizaje deductivo, donde los datos no son masivos, se adoptan supuestos sobre su naturaleza y el objetivo es aprender relaciones significativas entre los datos usando modelos simples que posteriormente permiten realizar predicciones. Si bien en general al analizar datos nos movemos entre estos dos extremos, podemos decir que la estadística es más cercana al segundo extremo, mientras que el AM contiene componentes que son de enfoque deductivo y muy relacionado a estadística (inferencia bayesiana), pero al mismo tiempo considera otros que son de enfoque inductivo (redes neuronales). Una consecuencia de esto es la importancia que tienen los parámetros en distintos métodos de AM: En los métodos deductivos (estadísticos) los parámetros tienen un rol explicativo de la naturaleza del fenómeno en cuestión que es revelado por los datos, mientras que los métodos inductivos se caracterizan por tener una infinidad de parámetros sin una explicación clara, pues el objetivo muchas veces es hacer predicciones directamente. La relación entre estadística y algunos enfoques a AM es tan cercana que Robert Tibshirani se ha referido a AM como “estadística pretenciosa” (*glorified statistics*).

AM y Programación Clásica. La programación clásica construye algoritmos basados en reglas, es decir, una lista de instrucciones que son ejecutadas en una máquina, lo cual requiere que el programador conozca de antemano el algoritmo a programar, e.g., calcular la transformada de Fourier rápida (FFT) de una grabación de audio. En AM, por el contrario, los algoritmos son precisamente lo que buscamos, por lo tanto, el enfoque que adopta AM es diferente al de la programación clásica en el sentido de que se busca programar la máquina para que aprenda la lista apropiada de instrucciones (en vez de programar la máquina para implementar dichas instrucciones). Tomemos el caso del ajedrez, de acuerdo a (Shannon, 1950) el número de combinaciones posibles para el juego de ajedrez es del orden de 10^{40} , esto significa que usando programación clásica un programa ingenuo para jugar ajedrez tendría que tener al menos esa cantidad de instrucciones de la forma

“para la combinación C, ejecutar la acción A”.

Si todos los humanos sobre la faz de la tierra se uniesen para programar dicha rutina y cada uno pudiese

escribir 10 de estas instrucciones por segundo, nos tomaría 4×10^{21} años, esto es casi un billón (10^{12}) de veces la edad de la tierra (4.54×10^9), lo cual hace impracticable adoptar un enfoque clásico de programación. Una alternativa basada en AM es un programa simple en el cual la máquina explora distintos posibles escenarios del tablero e inicialmente toma decisiones aleatorias de qué acción ejecutar para luego registrar si dicha movida llevó a ganar o perder el juego; este enfoque de programación no pretende programar la máquina para “jugar ajedrez” sino para “aprender a jugar ajedrez”. Una implementación exitosa de este concepto usando aprendizaje reforzado y redes neuronales profundas para el juego de Go puede verse en (Silver y cols., 2016).

AM, Knowledge Discovery in Databases (KDD) y minería de datos. KDD (Fayyad, Piatetsky-Shapiro, y Smyth, 1996) es “el proceso no trivial de identificar patrones potencialmente válidos, novedosos, útiles y explicativos en datos”, y consta de 5 etapas: selección, preparación, transformación, minería e interpretación de datos. La etapa de minería de datos consiste en extraer información desde datos disponibles, por ejemplo, agruparlos en subconjuntos afines o identificar situaciones anómalas. Para esto generalmente se consideran herramientas de AM, especialmente de aprendizaje no-supervisado debido a la cantidad de los datos que deben ser analizados en aplicaciones de KDD, los cuales en general no están no etiquetados y existe poco conocimiento *a priori* de su naturaleza.

AM y Data Science. En línea con sus orígenes en la inteligencia artificial, el objetivo del AM es encontrar relaciones entre datos y hacer predicciones prescindiendo de la intervención humana, es decir, con poco o nada de conocimiento *a priori*. Sin embargo, las técnicas de AM pueden ser complementadas con el conocimiento de un problema específico, en donde especialistas en AM colaboran con especialistas de (i) el área en cuestión, (ii) minería de datos, y (iii) visualización. Esto es *Data Science*, una disciplina colaborativa donde especialistas de variadas áreas trabajan de forma conjunta para hacer un análisis detallado de los datos, con la finalidad de resolver un problema en particular. El perfil del *Data Scientist* es muy completo, pues debe tener conocimiento de AM, estadística, programación, minería de datos, interactuar con especialistas y entender el impacto comercial del análisis. De hecho, la posición de *Data Scientist* ha sido considerada la más sexy del siglo XXI según *Harvard Business Review* (Davenport y Patil, 2012).

1.5. Estado del aprendizaje de máquinas y desafíos

El aprendizaje de máquinas es una reciente disciplina que provee de herramientas a una gran cantidad de disciplinas, pero también es un área de investigación en sí misma con una activa comunidad y muchas preguntas abiertas. Desde el punto de vista algorítmico es posible identificar en primer lugar el desafío del entrenamiento en línea, es decir, cómo ajustar un algoritmo/modelo cada vez que se dispone de un nuevo dato para operar continuamente (e.g. análisis de series de tiempo). Este es un concepto fundamental en procesamiento adaptativo de señales y no ha sido tomado en cuenta satisfactoriamente aún por la comunidad de AM. Otro desafío que tiene relación con la implementación de algoritmos es la capacidad de escalar el entrenamiento de AM para bases de datos masivas y no estructuradas (i.e., *Big Data*). Esto último porque en general los métodos de AM son costosos de implementar y entrenar, pues se enfocan en descubrir estructura y no en procesar datos de alta dimensión *per se*. Sin embargo, esta habilidad es cada vez más necesaria en la era de la información donde, los conceptos que actualmente se usan para lidiar con grandes volúmenes de datos son básicos comparados con es estado del arte en AM.

También es posible identificar desafíos en un plano conceptual, como por ejemplo la “transferencia de aprendizaje”, en donde la experiencia adquirida en la realización de una tarea (e.g., reconocimiento de automóviles) sirve como punto de partida para una tarea relacionada (e.g., reconocimiento de camiones). Este concepto es encontrado en el aprendizaje humano también, por ejemplo cuando un físico o matemático migra al mundo bancario sin tener conocimiento *a priori* de finanzas pero aposentador exitosamente en dicha área en poco tiempo. Por otro lado, un desafío ético son los riegos del uso de AM: el avance de AM parece a veces descontrolado y abre interrogantes con respecto de la legislación sobre el actuar de máquinas inteligentes. Por ejemplo, ¿quién es responsable en un accidente en el que está involucrado un

automóvil autónomo? Estos últimos dos desafíos, el conceptual y el ético, revelan que hay una dimensión importante que no hemos explorado y que, a pesar de los avances teóricos y sobretodo aplicados del AM, estamos lejos de entender la inteligencia. Como ha sido expuesto en (Gal, 2015), nuestra relación con el entendimiento de la inteligencia mediante el uso del AM puede ser entendido como el *Homo erectus* hace 400.000 años frotando dos ramas para producir fuego. Ellos usaron el fuego para abrigarse, cocinar y cazar, sin embargo, esto no quiere decir que entendían por qué al frotar dos ramas generaban fuego o, peor aún, qué es el fuego. Estamos en una etapa temprana del entendimiento del aprendizaje, en la que usamos estas herramientas “inteligentes” para nuestro bienestar, sin embargo, estamos lejos de entender la ciencia que hay detrás.

2. Regresión

2.1. Regresión lineal

El problema de regresión busca determinar la relación entre una variable *independiente* (entrada, estímulo o característica; usualmente denotada por x) y una variable *dependiente* (salida, respuesta o etiqueta; usualmente denotada y). Intuitivamente, un modelo de regresión permite entender cómo cambia la variable dependiente cuando la variable independiente es modificada. Esta relación entre ambas variables es representada por una función. Consecuentemente, el problema de regresión es equivalente a encontrar una función definida desde el espacio de la entrada x al de la salida y . De esta forma, en base al espacio de las posibles funciones donde se busque dicha relación (e.g., los polinomios de grado menor o igual a 5), y al criterio de búsqueda que se aplique (e.g., mínimos cuadrados), podemos obtener distintas soluciones para el problema de regresión.

El escenario básico de regresión, y que sirve de base para casos más complejos, es el de regresión lineal. En este caso, el espacio de funciones donde se busca la relación entre las variables dependientes e independientes es el de las funciones lineales afines. Específicamente, para un conjunto de entrenamiento \mathcal{D} que contiene $N \in \mathbb{N}$ observaciones de entrada y salida, respectivamente $\{x_i\}_{i=1}^N$ y $\{y_i\}_{i=1}^N$, de la forma

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N \subset \mathbb{R}^M \times \mathbb{R}, \quad (2.1)$$

la regresión lineal busca encontrar un modelo lineal, es decir, una función f definida por

$$\begin{aligned} f: \mathbb{R}^M &\rightarrow \mathbb{R} \\ x &\mapsto f(x) = a^\top x + b, \quad a \in \mathbb{R}^M, b \in \mathbb{R} \end{aligned} \quad (2.2)$$

que *mejor represente* la forma en que la variable y depende de la variable x , en base a las observaciones contenidas en el conjunto \mathcal{D} de la ec. (2.1). Antes de proceder a definir un criterio de *mejor representación*, el siguiente recuadro justifica la elección de modelos lineales.

¿Por qué consideramos el caso lineal en particular?

Existen distintas razones para estudiar los modelos lineales. En primer lugar, con el criterio de mínimos cuadrados que veremos a continuación, el modelo lineal es el único que admite resolución de forma explícita (o, como diremos alternativamente, *tiene forma cerrada*). Además de calcular dicha solución, la existencia de su forma cerrada nos permite interpretar las propiedades de dicha solución. En segundo lugar, los resultados que obtendremos a continuación requieren linealidad solo en los parámetros —ver ec. (2.2)— y no necesariamente en la variable independiente x . Por esta razón, el estudio del modelo lineal también incluye modelos no lineales del tipo

$$\begin{aligned} f: \mathbb{R}^M &\rightarrow \mathbb{R} \\ x &\mapsto f(x) = \theta^\top \phi(x), \quad \theta \in \mathbb{R}^{M'}, \end{aligned} \quad (2.3)$$

donde $\phi: \mathbb{R}^M \rightarrow \mathbb{R}^{M'}$ es una función no lineal sin parámetros libres. Es decir, estrictamente hablando deberíamos referirnos a los modelos lineales como *lineales en los parámetros* y no necesariamente *lineales en la entrada*.

Finalmente, cuando los parámetros a determinar afectan de forma no lineal la relación entre las variables dependiente e independiente, el análisis presentado a continuación no es válido y, en general, la solución óptima de mínimos cuadrados no tiene forma cerrada.

2.1.1. Mínimos cuadrados

En el contexto recién presentado, aflora naturalmente la siguiente pregunta: *¿qué es una buena función f ?* o, equivalentemente, *¿cómo cuantificar la bondad de un modelo de regresión lineal?* Una práctica ampliamente utilizada es elegir la función f en la ec. (2.2) de acuerdo al criterio de **mínimos cuadrados**. Es decir, elegir la función f que minimiza la suma de los cuadrados de las diferencias entre las observaciones $\{y_i\}_{i=1}^N$ y las predicciones calculadas por la función $\{f(x_i)\}_{i=1}^N$ de acuerdo al siguiente costo (denotado por J):

$$J(\mathcal{D}, f) = \sum_{i=1}^N (y_i - f(x_i))^2, \quad (2.4)$$

Donde hemos sido enfáticos en que el costo depende del conjunto de entrenamiento $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ y la función f , sin embargo, cuando estas cantidades son claras, nos referiremos al costo simplemente como J . Además, denotamos la (o las) funciones que satisfacen el criterio de mínimos cuadrados mediante

$$f^* = \arg \min_{f \text{ es afín}} J. \quad (2.5)$$

Debido a la forma afín de f , resolver este problema de optimización es equivalente a encontrar los parámetros a y b en la ec. (2.2), es decir:

$$a^*, b^* = \arg \min_{a, b} \sum_{i=1}^N (y_i - a^\top x_i - b)^2. \quad (2.6)$$

Observemos que el costo en la ec. (2.6) es cuadrático en a y b , por lo que el problema de optimización tiene un único mínimo que puede ser encontrado explícitamente. Para esto, como la función f en la ec. (2.6) no es lineal sino que *afín*, hacemos el siguiente cambio de variable:

$$\tilde{x}_i = \begin{pmatrix} x_i \\ 1 \end{pmatrix} \in \mathbb{R}^{M+1}, \quad \theta = \begin{pmatrix} a \\ b \end{pmatrix} \in \mathbb{R}^{M+1} \implies J(\mathcal{D}, \theta) = \sum_{i=1}^N (y_i - \theta^\top \tilde{x}_i)^2. \quad (2.7)$$

Una forma común y natural de representar los datos es mediante matrices, transformando la minimización en un problema de optimización matricial. De este modo, se definen las matrices:

$$\tilde{X} = \begin{pmatrix} \tilde{x}_1^\top \\ \vdots \\ \tilde{x}_N^\top \end{pmatrix} \in \mathbb{R}^{N \times (M+1)}, \quad Y = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \in \mathbb{R}^N \implies J = \|Y - \tilde{X}\theta\|_2^2, \quad (2.8)$$

donde \tilde{X} se denomina matriz de diseño o matriz de regresión. De esta forma, dado que el funcional J es convexo, puede ser minimizado utilizando condiciones de primer orden:

$$\begin{aligned} \nabla_\theta J &= 2(Y - \tilde{X}\theta)^\top(-\tilde{X}) = 0 \\ \iff Y^\top \tilde{X} - \theta^\top \tilde{X}^\top \tilde{X} &= 0 \\ \iff \theta^\top &= Y^\top \tilde{X} (\tilde{X}^\top \tilde{X})^{-1} \\ \iff \theta &= (\tilde{X}^\top \tilde{X})^{-1} \tilde{X}^\top Y. \end{aligned} \quad (2.9)$$

Con los parámetros del modelo regresión lineal encontrados con el criterio de mínimos cuadrados, es posible implementar la solución y compararla visualmente con los datos. La Figura 1 muestra la regresión lineal correspondiente a chirridos de grillos por segundo en función de la temperatura (Pierce, 1949).

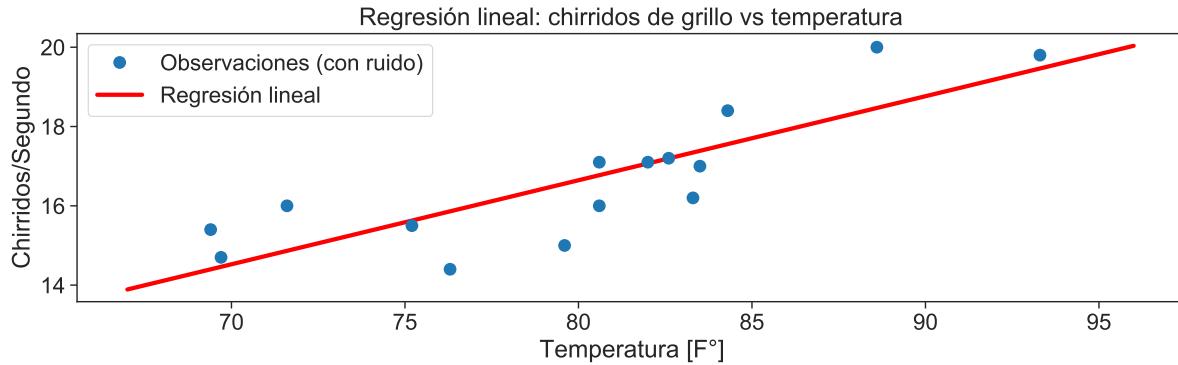


Fig. 1. Ejemplo de regresión lineal mediante mínimos cuadrados sobre la base de datos de chirridos versus temperatura.

La expresión $(\tilde{X}^\top \tilde{X})^{-1} \tilde{X}^\top$ en la ec. (2.9) corresponde a la pseudoinversa de Moore-Penrose de \tilde{X} (Ben-Israel y Greville, 2006, p. 7) y por lo tanto, es necesario que $r(\tilde{X}) = M + 1$ para que esté bien definida (ver anexo). Además, dado que $\tilde{X} \in \mathbb{R}^{N \times (M+1)}$, entonces $r(\tilde{X}) \leq \min\{N, M + 1\}$ por lo que necesariamente se debe cumplir que $N \geq M + 1$, es decir, que el número de muestras sea mayor que el número de dimensiones. Intuitivamente esto se debe a que se necesitan $M + 1$ puntos para fijar un hiperplano en \mathbb{R}^{M+1} .

Es claro que para el caso de variables continuas es muy poco usual que dos observaciones sean perfectamente colineales (i.e., linealmente dependientes), sin embargo, en el caso de variables categóricas donde las observaciones son asignadas a un número finito de símbolos, es probable que dos o más valores para la variable dependiente sean exactamente iguales.

En la práctica, generalmente tendremos más observaciones que parámetros y estas serán linealmente independientes. Sin embargo, es posible que las observaciones sean tal que la inversión de la matriz $\tilde{X}^\top \tilde{X}$ sea numéricamente inestable. Esto ocurre fundamentalmente en dos casos ilustrados en el siguiente recuadro.

Matriz cuasi-singular o incorrectamente escalada

Al tratar de invertir una matriz de forma computacional, probablemente hemos obtenido un mensaje de la forma `matrix is singular, close to singular or badly scaled`. Veremos dos ejemplos para entender de dónde viene esta advertencia.

Caso 1: Consideremos la matriz

$$A = \begin{bmatrix} 10^{50} & 1 \\ 10^{50} & 2 \end{bmatrix},$$

dicha matriz es claramente invertible y su inversa puede ser calculada mediante

$$A^{-1} = \frac{1}{10^{50} \cdot 2 - 10^{50} \cdot 1} \begin{bmatrix} 2 & -1 \\ -10^{50} & 10^{50} \end{bmatrix} = \begin{bmatrix} 2 \cdot 10^{-50} & -10^{-50} \\ -1 & 1 \end{bmatrix},$$

donde las entradas difieren en 50 órdenes de magnitud. Sin embargo, la representación usual que consideramos cuando programamos es la de punto flotante de precisión simple, la cual considera a $2^{-127} \approx 10^{-38}$ como el menor valor positivo. Consecuentemente, los valores más pequeños que este límite serán aproximados por el elemento más cercano, es decir, cero. Utilizando la inversa

aproximada, denotada \tilde{A}^{-1} , resulta en errores como el siguiente:

$$A\tilde{A}^{-1} = \begin{bmatrix} 10^{50} & 1 \\ 10^{50} & 2 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ -2 & 2 \end{bmatrix} \neq \mathbb{I}.$$

Caso 2: Consideremos

$$A = \begin{bmatrix} a & a \\ b & b + \epsilon \end{bmatrix},$$

la cual también es invertible para $a, \epsilon > 0$, pues su determinante está dado por

$$\det A = a(b + \epsilon) - ab = a\epsilon > 0.$$

Sin embargo, si $\epsilon \ll 1$, entonces el cálculo de la inversa puede sufrir inestabilidades numéricas como en el caso anterior. Sin embargo, observe para un $\eta > 0$ suficientemente grande, la matriz $A + \eta I$ puede tener un determinante arbitrariamente grande (ver sección 2.1.2).

Es relevante reflexionar por qué consideramos mínimos cuadrados como la métrica de error relacionada al problema de regresión. Existen varias razones de por qué lo hacemos, tanto técnicas como conceptuales, como también diversas desventajas de este criterio que es importante identificar. Desde el punto de vista técnico, el costo estrictamente convexo de un modelo lineal (en los parámetros) define un problema de optimización que también es estrictamente convexo y por ende tiene una solución única. Además, en el caso particular del costo cuadrático, este óptimo puede ser determinado de forma explícita (lo cual es fuertemente deseado), pues está dado únicamente por la inversión de una matriz y no mediante, e.g., una búsqueda iterativa.

Desde un punto de vista, el exponente 2 busca penalizar mayormente grandes diferencias (mayores que 1) entre la predicción y el valor real, mientras que le quita importancia a errores de predicción pequeños (menores que 1). Si bien es posible utilizar otros exponentes bajo el mismo razonamiento de penalización, se utiliza mayormente la minimización de cuadrados ya que tiene propiedades matemáticas muy favorables en cuanto al análisis técnico por lo permite obtener conclusiones teóricas más fuertes.

Desde un punto de vista conceptual, otra justificación para usar la medida del error cuadrático es que este representa la varianza muestral. Es decir, si considerásemos que x_i e y_i son observaciones iid de variables aleatorias X e Y respectivamente, entonces el error cuadrático medio (asociado a la función f) definido por

$$ECM = \frac{1}{N} \sum_{i=1}^N (y_i - f(x_i))^2, \quad (2.10)$$

corresponde a la varianza muestral de la variable aleatoria $Y - f(X)$ (asumiendo que $\mathbb{E}(Y - f(X)) = 0$). De igual forma, la varianza de la suma de múltiples variables aleatorias (pensemos en errores acumulados, los cuales son independientes) corresponde a la suma de las varianzas de dichas VAs. Esto ocurre precisamente cuando usamos el exponente igual a 2, y no si usáramos 1.95 o 2.05.

Podemos además justificar el uso del error cuadrático con una motivación geométrica. Recordemos que el problema de regresión lineal requiere encontrar una solución aproximada de un sistema lineal sobredeterminado definido por

$$\tilde{X}\theta = Y, \quad (2.11)$$

donde la cantidad de incógnitas ($M+1$) es ampliamente superada por el número de ecuaciones (N). Como esta solución, desde el punto de vista de un sistema lineal, no necesariamente existe (y en la práctica no existe), uno puede proceder a encontrar la solución para θ que reporta *la menor discrepancia* entre ambos lados de la ecuación (2.11). En este sentido, podemos identificar el espacio formado por todos

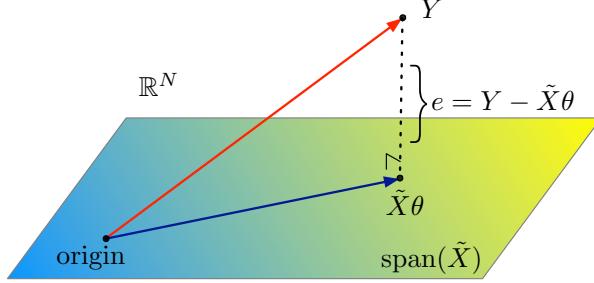


Fig. 2. Interpretación geométrica de la regresión lineal y mínimos cuadrados

los posibles valores que toma la combinación lineal $\tilde{X}\theta$ para distintos valores de $\theta \in \mathbb{R}^{M+1}$, es decir, el span de todas las columnas de \tilde{X} , formando un subespacio vectorial de \mathbb{R}^N . Luego, podemos identificar el elemento de dicho espacio que está más cerca de $Y \in \mathbb{R}^N$ como la proyección del propio Y en $\text{span}(\tilde{X})$. Esto está ilustrado en la Figura 2, donde la condición para identificar dicha proyección es precisamente que el vector error $e = Y - \tilde{X}\theta$ sea ortogonal al espacio $\text{span}(\tilde{X})$ generado por los datos de entrada. Además, dado que las columnas de \tilde{X} son una base de $\text{span}(\tilde{X})$:

$$u \in \text{span}(\tilde{X}) \iff \exists \theta_u \in \mathbb{R}^{M+1} : u = \tilde{X}\theta_u, \quad (2.12)$$

por lo tanto:

$$\begin{aligned} e \perp \text{span}(\tilde{X}) &\iff e \perp u, \forall u \in \text{span}(\tilde{X}) \iff (Y - \tilde{X}\theta)^T \tilde{X}\theta_u = 0, \forall \theta_u \in \mathbb{R}^{M+1} \\ &\iff (Y - \tilde{X}\theta)^T \tilde{X} = 0 \iff \theta = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T Y. \end{aligned} \quad (2.13)$$

Es decir, la distancia se minimiza para el θ , correspondiente a la solución por mínimos cuadrados.

Finalmente, notemos que el criterio de mínimos cuadrados (MC) también tiene desventajas. Implícitamente, MC está intrínsecamente relacionado con un supuesto de gaussianidad de los datos; esto será evidente cuando estudiemos el criterio de máxima verosimilitud. Consecuentemente, el uso de MC produce estimaciones razonables cuando la relación entre x e y es simétrica y sin *grandes desviaciones*. Por el contrario, cuando existen datos que se alejan mucho de la tendencia buscada, las estimaciones encontradas mediante MC pueden desviarse considerablemente de la solución buscada, esto se debe precisamente a la contribución cuadrática del error, donde, coloquialmente, una muestra *muy alejada* pesa tanto o más que varias muestras *ligeramente alejadas*. La Figura 3 ilustra este fenómeno para el mismo ejemplo de los chirridos en la Fig. 1, donde se ha introducido un *outlier*, es decir, una observación que está inusualmente alejada de los datos y se ha recalculado el resultado de la regresión lineal mediante el criterio de MC. Se puede ver cómo se deteriora la estimación solo con la introducción de un nuevo dato.

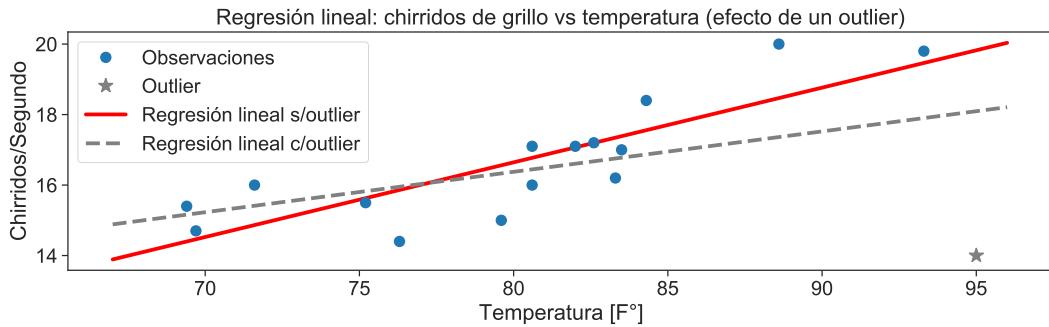


Fig. 3. Se ha agregado un dato erróneo (*outlier* en gris) y se ha recalculado la regresión lineal, note cómo la inclusión de dicho punto deteriora el resultado de la regresión.

La lección que queda de este ejemplo es que debemos considerar una métrica *ad hoc* al problema que estamos considerando, por ejemplo, si es muy probable que existan outliers, no debemos penalizar cuadráticamente los errores. De igual forma, al elegir una métrica de error debemos verificar cuán relevante es que el error de regresión sea nulo vs muy pequeño, o bien, grande vs extremadamente grande. La Figura 4 presenta cuatro métricas de error (como función del propio error), donde podemos interpretar sus propiedades.

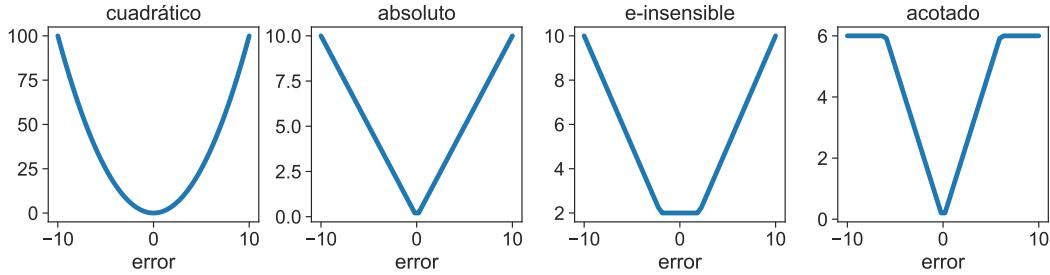


Fig. 4. Distintas funciones de costo en función del error de estimación, de izquierda a derecha: cuadrático, absoluto (crecimiento lineal en función del error), ϵ -insensible (es irrelevante si el error está entre 0 o ϵ) y acotado (es irrelevante si el error es mayor que cierto umbral).

2.1.2. Regularización: ajuste versus generalización

Perseguir ciegamente la solución de mínimos cuadrados puede resultar, como discutimos en la sección anterior, en situaciones donde la inversa de Moore-Penrose sea *cercana* a singular, especialmente en los casos que las observaciones son parecidas o redundantes. En este sentido, debemos considerar un criterio que no simplemente busque un ajuste a los datos, sino que también promueva ciertas propiedades de la solución, por ejemplo, suavidad, bajas magnitudes de los parámetros o incluso pocos parámetros. Nos referiremos a estas soluciones como *regulares*, y el objetivo de este apartado será *regularizar* la solución de mínimos cuadrados.

Las penalizaciones a considerar en el problema de regresión pueden ser codificadas directamente en la función de costo. Por ejemplo, esta puede incluir un término que promueva el ajuste de los datos y otro término que sancione soluciones que se alejan de lo deseado. Un criterio estándar de penalización es el basado en la norma de los parámetros, es decir,

$$J_\rho = \left\| Y - \tilde{X}\theta \right\|_2^2 + \rho \|\theta\|_p^p, \quad p \geq 0, \quad (2.14)$$

donde $\|\cdot\|_p$ denota la norma ℓ_p , es decir, $\|\theta\|_p = \left(\sum_{j=1}^{M+1} |\theta_j|^p \right)^{\frac{1}{p}}$ y el parámetro $\rho \geq 0$ tiene el rol de balancear la importancia entre ajuste (primer término) y regularidad de la solución (segundo término). Distintos valores de p inducen distintas propiedades sobre las soluciones, siendo las más usadas las correspondientes a $p = 1$, conocido como **LASSO**² (Tibshirani, 1996), y $p = 2$ conocido como **regularización de Tikhonov**³ (Tikhonov y Arsenin, 1977) o bien **Ridge Regression**.

Una ventaja de la regularización de Tikhonov es que su solución, al igual que el caso de mínimos cuadrados no regularizados, puede ser encontrada en forma exacta, en efecto, dado que el funcional J_ρ

²Least Absolute Shrinkage and Selection Operator.

³La regularización de Tikhonov puede ser generalizada mediante penalizaciones matriciales, de este modo se tiene un costo $J_\Gamma(\theta) = \left\| Y - \tilde{X}\theta \right\|_2^2 + \|\Gamma\theta\|_2^2 \implies \theta = (\tilde{X}^\top \tilde{X} + \Gamma^\top \Gamma)^{-1} \tilde{X}^\top Y$. Se recupera la regularización $p = 2$ tomando $\Gamma = \alpha I$.

nuevamente es convexo, se puede minimizar utilizando la condición de primer orden:

$$\begin{aligned}
\nabla_{\theta} J_{\rho} &= 0 \\
\iff -2(Y - \tilde{X}\theta)^{\top} \tilde{X} + 2\rho\theta^{\top} &= 0 \\
\iff -Y^{\top} \tilde{X} + \theta^{\top} \tilde{X}^{\top} \tilde{X} + \rho\theta^{\top} &= 0 \\
\iff \theta^{\top} &= Y^{\top} \tilde{X}(\tilde{X}^{\top} \tilde{X} + \rho\mathbb{I})^{-1} \\
\iff \theta &= (\tilde{X}^{\top} \tilde{X} + \rho\mathbb{I})^{-1} \tilde{X}^{\top} Y.
\end{aligned} \tag{2.15}$$

De la última expresión, es posible ver que el requerimiento de no colinealidad de las observaciones y $N \geq M + 1$ ya no son necesarios para que la solución esté bien definida ya que es posible *regularizar* la solución forzando que la matriz $\tilde{X}^{\top} \tilde{X} + \rho\mathbb{I}$ sea arbitrariamente lejana de las matrices singulares (o tenga un determinante arbitrariamente grande) aumentando el valor de ρ , en efecto:

Proposición 2.0.1. *Sea $A \in \mathcal{M}_{nn}(\mathbb{R})$, entonces $\det(A + \rho\mathbb{I}) \rightarrow \infty$ cuando $\rho \rightarrow \infty$.*

*Demuestra*ción. El polinomio $q(\rho) = \det(A + \rho\mathbb{I})$ puede ser visto como el polinomio característico $p(\lambda)$ de la matriz A con el cambio de variable $\lambda \mapsto -\rho$. Dado que el polinomio característico es de grado n (i.e., no es constante), entonces $|q(\rho)| = |\det(A + \rho\mathbb{I})| \rightarrow \infty$ cuando $\rho \rightarrow \infty$. ■

Es relevante entender por qué una disminución en la norma de los parámetros, puede ayudar a ajustar *mejores* modelos. A primera impresión, un podría pensar que el criterio de mínimos cuadrados regularizados (MCR) en ningún caso puede reportar mejores modelos que su contraparte MC, pues MCR es una variante restringida del problema original y consecuentemente solo puede *en el mejor de los casos* alcanzar la solución óptima de MC. Esto es cierto si por *mejor modelo* solo consideramos el error cuadrático medio sobre \mathcal{D} , sin embargo, solo enfocarse en esta métrica no siempre es la mejor opción. Para ilustrar este concepto, tomemos las siguientes consideraciones: asumamos que efectivamente los datos cumplen la relación

$$y_i = \underbrace{\theta^{\top} \tilde{x}_i}_{f_i} + \epsilon_i, \tag{2.16}$$

donde ϵ_i son observaciones iid de una variable aleatoria de media nula y varianza σ^2 , θ es un parámetro fijo pero desconocido y $f_i = \theta^{\top} \tilde{x}_i$ se refiere a la *parte determinista* del modelo (para la observación \tilde{x}_i). Además, consideremos una estimación del parámetro θ construida en base a un conjunto de entrenamiento $\mathcal{D} = \{(\tilde{x}_i, y_i)\}_{i=1}^N$, denotada $\hat{\theta} = \hat{\theta}_{\mathcal{D}}$. Con estas consideraciones, se puede probar que para un nuevo par $(\tilde{x}_{\star}, y_{\star})$, el *error cuadrático esperado* asociado a la **predicción** $\hat{f}_{\star} = \hat{\theta}^{\top} \tilde{x}_{\star}$ se puede descomponer de acuerdo al trade-off entre el sesgo y la varianza (James, Witten, Hastie, y Tibshirani, 2014). Dicha descomposición viene dada por:

$$\mathbb{E}(y_{\star} - \hat{f}_{\star})^2 = \text{Sesgo}(\hat{f}_{\star})^2 + \text{Varianza}(\hat{f}_{\star}) + \sigma^2, \tag{2.17}$$

donde el valor esperado es tomado con respecto a la ley de ϵ , la única fuente de incertidumbre en este escenario. Los 3 sumandos de la descomposición corresponden a:

- Sesgo(\hat{f}_{\star}) := $\mathbb{E}(\hat{f}_{\star}) - f_{\star}$. Representa una medida de exactitud: ¿cuán buena (en valor esperado) es la estimación con respecto al valor real?
- Varianza(\hat{f}_{\star}) = $\mathbb{E}(\mathbb{E}(\hat{f}_{\star}) - \hat{f}_{\star})^2$. Representa una medida de precisión: ¿cuán disperso es el estimador?
- σ^2 es la varianza de *ruido* ϵ y es la parte irreducible del costo, en el sentido que no puede ser controlada por la elección de $\hat{\theta}$.

Podemos evaluar el sesgo y la varianza para el estimador de mínimos cuadrados, denotado $\hat{\theta}_{MC}$, eligiendo $\hat{f}_{\star} = \hat{\theta}_{MC}^{\top} \tilde{x}_{\star}$. Se tiene el siguiente resultado:

Teorema 2.1. *Bajo la hipótesis de ruido aditivo sobre un modelo lineal, el estimador de mínimos cuadrados cumple que:*

$$\text{Sesgo}(\hat{f}_*) = 0 \quad (2.18)$$

$$\text{Varianza}(\hat{f}_*) = \sigma^2 \tilde{x}_*^\top (\tilde{X}^\top \tilde{X})^{-1} \tilde{x}_*. \quad (2.19)$$

Es decir, el modelo de regresión lineal ajustado mediante MC reporta un estimador insesgado (sesgo nulo) pero con una varianza que depende de los datos en el conjunto de entrenamiento \mathcal{D} , la varianza del ruido σ^2 y la propia entrada \tilde{x}_* . Si bien no es posible determinar cuánto es esta varianza sin tomar supuestos estadísticos sobre los \tilde{x}_i , recordemos que la matriz $\tilde{X}^\top \tilde{X}$ puede ser cercana a singular cuando los datos son pocos, redundantes o colineales, lo cual resultará en alta varianza para la predicción \hat{f}_* . De hecho, si asumíramos que $\tilde{x}_i \sim \mathcal{N}(0, 1)$ iid, tendríamos que $\text{Varianza}(\hat{f}_*) = \sigma^2 M/N$, es decir, la varianza es inversamente proporcional a la razón entre la dimensión de las entradas y la cantidad de muestras.

Evaluaciones dentro de muestra y fuera de muestra

Notemos que la expresión en la ecuación (2.17) es una medida de error *fueras de muestra*, pues evalúa un estimador $\hat{\theta}$, construido en base a un conjunto \mathcal{D} , en un nuevo dato (\tilde{x}_*, y_*) que no está originalmente contenido en \mathcal{D} . No debemos confundir esta expresión con el error cuadrático medio, en la ecuación (2.4), el cual representa un error *dentro de muestra*. La evaluación de ambos tipos de costos es clave para diseñar modelos y estimadores que puedan *generalizar* a datos no vistos anteriormente.

Al penalizar la norma l_2 del parámetro θ , la regresión de ridge sacrifica la propiedad insesgada del estimador de mínimos cuadrados, pero en retorno construye un estimador que tiene menos varianza que el de MC. Esto puede entenderse como un balance entre confiar únicamente en los datos (los cuales pueden ser pocos o muy ruidoso y, consecuentemente, insuficientes para determinar un modelo apropiado) e introducir un *sesgo* al modelo (por ejemplo, parámetros pequeños) con la finalidad de robustecer el modelo encontrado en función de los datos disponibles. En efecto, se tiene lo siguiente:

Teorema 2.2. *Bajo la hipótesis de ruido aditivo sobre un modelo lineal, el estimador de mínimos cuadrados regularizados cumple que:*

$$\text{Sesgo}(\hat{f}_*) = \tilde{x}_*^\top \left((\mathbb{I} + \rho(\tilde{X}^\top \tilde{X})^{-1})^{-1} - \mathbb{I} \right) \theta \quad (2.20)$$

$$\text{Varianza}(\hat{f}_*) = \sigma^2 \tilde{x}_*^\top (\tilde{X}^\top \tilde{X} + \rho \mathbb{I})^{-1} \tilde{X}^\top \tilde{X} (\tilde{X}^\top \tilde{X} + \rho \mathbb{I})^{-1} \tilde{x}_*. \quad (2.21)$$

Además, la varianza del estimador de MCR es menor a la varianza del estimador de MC.

Esta noción de (sobre-)ajustar a los datos de entrenamiento versus generalizar a nuevos puede ser ilustrada con el siguiente ejemplo: consideremos $N = 1000$ datos relacionados linealmente (pares de entrada y salida) donde la dimensión de entrada es $M = 100$, generados por el siguiente script:

```

1 ##generacion de datos relacionados linealmente.
2 n_samples, n_features = 1000, 100
3 rng = np.random.RandomState(0)
4 X = rng.randn(n_samples, n_features)
5 theta = rng.randn(n_features, 1)
6 y = X@theta + 10rng.randn(n_samples, 1)

```

En vez de utilizar todas las muestras de entrenamiento, utilicemos solo $N' = 15$ muestras para entrenar el modelo usando los criterios de MC, y regresión de ridge (RR) con $\rho \in \{40, 80\}$. Repitiendo este proceso 400 veces, podemos analizar cómo se comportan los distintos estimadores mediante el error dentro de

muestra (con respecto a los datos de entrenamiento) y el error fuera de muestra (con respecto a los $N - N'$ datos no usados para entrenar). La Figura 5 muestra dichos histogramas, desde donde podemos ver que a mayor ρ (recordemos que MC es equivalente a RR con $\rho = 0$), los parámetros encontrados tienen menor magnitud. Adicionalmente, notemos que el modelo no regularizado (MC) se comporta mejor en evaluación dentro de muestra, sin embargo, sus papeles se invierten cuando se trata de evaluación fuera de muestra: el incluir un sesgo en el ajuste de modelos (RR) puede ayudar a generalizar y no sobreajustar cuando se tienen pocos datos.

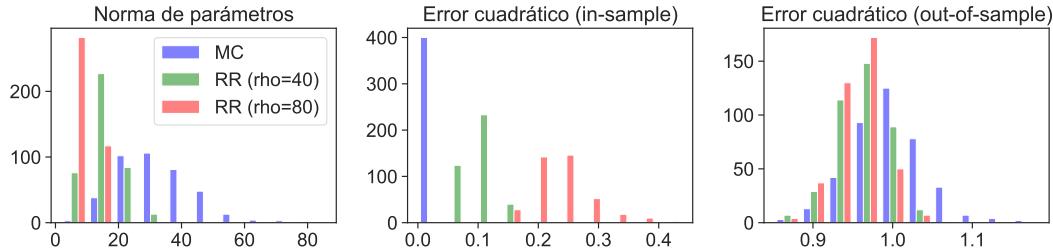


Fig. 5. Mínimos cuadrados versus regresión de Ridge ($\rho \in \{40, 80\}$): determinación de parámetros usando solo 15 muestras para un parámetro de dimensión $M = 100$. De derecha a izquierda podemos ver magnitud de los parámetros encontrados, error dentro de muestra y error fuera de muestra. Experimento repetido 400 veces.

Si bien el problema de mínimos cuadrados regularizados incluye en su función objetivo una penalización sobre la norma de θ , el problema también puede ser visto como el problema de mínimos cuadrados original con una restricción adicional que fija la norma del parámetro. En efecto, dicho problema viene dado por:

$$\min_{\theta} \left\| Y - \tilde{X}\theta \right\|_2^2 \quad (2.22)$$

$$\text{s.a. } \|\theta\|_p^p = \tau, \quad (2.23)$$

donde $\tau \geq 0$ es una constante fijada. Este problema puede ser resuelto mediante multiplicadores de Lagrange:

$$L(\theta, \lambda) = \left\| Y - \tilde{X}\theta \right\|_2^2 + \lambda (\|\theta\|_p^p - \tau), \quad (2.24)$$

donde λ es el multiplicador de Lagrange asociado al problema. Usando la condición de primer orden en la ecuación (2.24) se tiene que:

$$\frac{\partial L}{\partial \theta} = 0 \Rightarrow \frac{\partial}{\partial \theta} \left(\left\| Y - \tilde{X}\theta \right\|_2^2 + \lambda \|\theta\|_p^p \right) = 0 \quad (2.25)$$

$$\frac{\partial L}{\partial \lambda} = 0 \Rightarrow \|\theta\|_p^p = \tau, \quad (2.26)$$

lo cual recupera la forma del problema de minimización de mínimos cuadrados regularizados. Enfatizamos esta relación en el siguiente recuadro.

Mínimos cuadrados regularizados: optimización con restricciones

Observemos que el problema de mínimos cuadrados regularizados, determinado por el costo en la ecuación (2.14), es equivalente al dual del problema de optimización con restricciones dado en las ecuaciones (2.22)-(2.23) para un λ dado tal que $\lambda = \rho$. Consecuentemente, como el valor óptimo de λ depende del nivel de la restricción τ , podemos aseverar que **para cualquier $\rho \geq 0$, existe un $\tau \geq 0$ tal que la minimización de (2.14) es equivalente a la de (2.24)**. Consecuentemente,

podemos interpretar el problema de MCR como el de MC sujeto a una restricción sobre la norma del parámetro.

Con la interpretación del problema de regularización como un problema de optimización con restricciones sobre la norma del parámetro θ , podemos entender distintos regularizadores (distintos $p \geq 0$) mediante sus curvas de nivel. La Figura 6 ilustra las curvas correspondientes al costo cuadrático (izquierda) y al término de regularización en la ecuación (2.14) para $p \in \{0.5, 1, 2\}$.

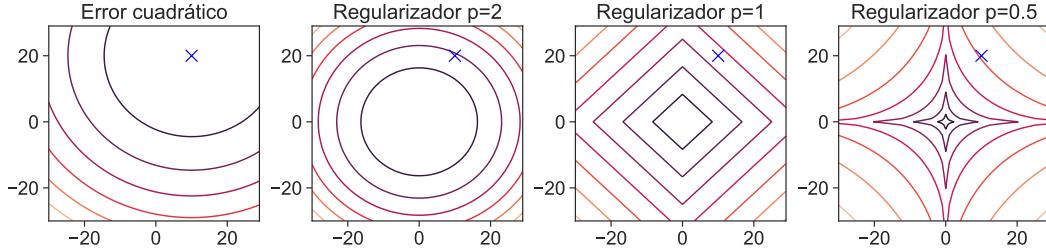


Fig. 6. Curvas de nivel del costo cuadrático para un problema hipotético con solución $\theta = [10, 20]$ (izquierda) y términos de regularización para órdenes $p \in \{0.5, 1, 2\}$. Observe cómo las curvas de nivel atraen el mínimo hacia el origen de distinta forma: $p = 2$ lleva la solución directamente al origen, mientras que $p \in \{0.5, 1\}$ lleva la solución a los bordes, es decir, privilegiando soluciones ralas. La solución $\theta = [10, 20]$ se ha denotado con una cruz azul, recuerde que solo el costo cuadrático depende de este valor, no los términos de regularización.

La formulación en base a restricciones es clave para entender la propiedad de *selección de características* de los métodos de MCR. Al estimar el parámetro θ , estamos verificando cuán importante es cada coordenada de la entrada o en la jerga de reconocimiento de patrones, cada *característica*. Indirectamente estamos también implícitamente descubriendo cuáles son las características que importan y cuales no, a esto nos referimos como selección de características. Distintas normas para el término de regularización, como las ilustradas en la Figura 6, inducen distintas propiedades para la solución del problema de MCR. En particular, RR atrae *homogéneamente* el parámetro hacia el origen, lo cual resulta en estimaciones de menor varianza como vimos en el apartado anterior. LASSO ($p = 1$) y el caso $p \leq 1$ en general presenta una propiedad adicional, donde la forma de la curva de nivel permite que usualmente la solución del problema se concentre en las puntas del *diamante* (ver Fig. 6, $p = 1$), llevando algunas coordenadas del parámetro θ directamente a cero. Por esto decimos que LASSO (y $p \leq 1$ en general) tiene la propiedad de selección de variables y entrega modelos ralos con respecto a MC tradicional.

Para ilustrar la propiedad de selección de características, consideremos el *Breast Cancer Wisconsin Data Set*⁴, el cual tiene $N = 569$ muestras y un dimensión de entrada de $M = 30$. Los posibles valores para la variable de salida y son solo dos, pues este es un problema de clasificación (*cáncer vs no cáncer*), sin embargo, nosotros ajustaremos un modelo de regresión lineal usando MC, RR y LASSO para luego evaluar los estimadores encontrados. Usaremos $2/3$ de los datos para entrenar y el $1/3$ restante para calcular puntajes fuera de muestra. La Figura 7 presenta los parámetros encontrados para cada uno de los métodos, donde podemos ver la propiedad de selección de variables de LASSO. Adicionalmente, la Tabla 1 muestra los puntajes de cada método, tanto dentro como fuera de muestras: en la línea de la discusión anterior, los modelos regularizados presentan mejor generalización y usan menos parámetros (o más parámetros iguales a cero).

⁴[https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+\(original\)](https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(original))

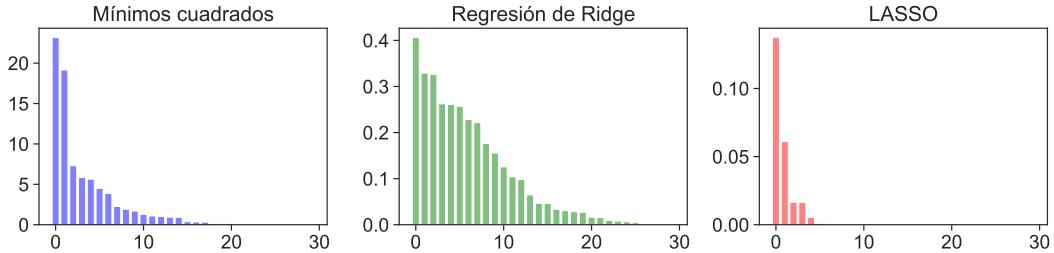


Fig. 7. Parámetros de la regresión lineal del *Breast Cancer Dataset* usando MC, RR y LASSO. Observe cómo RR y LASSO disminuye críticamente la magnitud de los parámetros y, además, LASSO lleva parámetros directamente a cero, resultando en un modelo más simple (i.e, con menos parámetros).

| | in-sample | out-of-sample |
|-------|---------------|---------------|
| MC | 0.7896 | 0.6911 |
| RR | 0.6905 | 0.6903 |
| LASSO | 0.7452 | 0.7242 |

Tabla 1. Puntajes de modelos de regresión implementados en *Breast Cancer Wisconsin Data Set*, más alto es mejor. Observe la superioridad de los modelos regularizados para generalizar.

Por otro lado, a diferencia de ridge-regression, LASSO no tiene una solución con forma cerrada por lo que se debe optimizar con algoritmos de programación cuadrática. Otra de las desventajas que tiene este regularizador ocurre al trabajar con entradas de alta dimensión (por ejemplo, imágenes o videos) y un conjunto de entrenamiento pequeño (es decir, $N < M$), donde la selección de características escoge a lo más N componentes no nulas y el resto las descarta, ignorando posibles correlaciones entre las variables debido a un conjunto de entrenamiento pequeño. Para evitar esto, se pueden utilizar la regularización LASSO y ridge al mismo tiempo, resultando en lo que se denomina *elastic net regularization*, cuyo funcional de costo es

$$J_\rho = \left\| Y - \tilde{X}\theta \right\|_2^2 + \lambda_1 \|\theta\|_2^2 + \lambda_2 \|\theta\|_1. \quad (2.27)$$

Consideraciones generales

Para concluir esta sección, es importante tener en cuenta lo siguiente:

- **¿es justo comparar MC y MCR en términos del ECM?** Ciertamente no, el criterio de MC siempre reportará un menor ECM, pues ha sido entrenado para minimizar dicho costo. Las ventajas de MCR están en su desempeño fuera de muestra, selección de variables o, en general, en su habilidad de incorporar sesgo de *diseñador* en la soluciones que no afloren naturalmente de los datos.

- **¿cómo elegir ρ ?** De forma general, este *hiperparámetro* determina el balance entre regularización (cuán sesgado) y ajuste (cuán bien replica los datos de entrenamiento), consecuentemente lo debemos elegir según nuestra intención. En la práctica, se puede evaluar el desempeño fuera de muestra para distintos valores de ρ con la finalidad de elegir un valor apropiado. Una manera de realizar la evaluación de desempeño para diferentes ρ es la *validación cruzada* (ver selección

de modelos).

- Vimos que la norma ℓ_p con $0 < p \leq 1$ tiene la propiedad de selección de características, pero, **¿qué pasa con la “norma” ℓ_0 ?** La cantidad $\ell_0(\theta)$ denota la cantidad de elementos no nulos de θ y, si bien no es una norma en el sentido formal de la palabra, puede de todas formas ser usada en la definición del costo en la ecuación (2.14), con la finalidad de directamente penalizar la cantidad de características usadas por el modelo. Desafortunadamente, encontrar la solución usando la “norma” ℓ_0 es muy difícil, sin embargo, bajo ciertas condiciones, la consideración de la norma ℓ_1 puede llevar a la misma solución.

2.2. Máxima verosimilitud

En el apartado anterior vimos que para la regresión lineal, el criterio de mínimos cuadrados, y su variante regularizada, ofrecen una alternativa simple, elegante, interpretable y con solución en forma cerrada. Sin embargo, también vimos que dicho criterio sufre de desventajas en cuanto a su capacidad de ajustar modelos en casos generales, pues el criterio de MC es particularmente apropiado para variables continuas, con perturbaciones aditivas y simétricamente dispersas con respecto a una tendencia dada. Existen distintos casos donde el criterio de MC no es apropiado, por ejemplo aplicaciones financieras con perturbaciones multiplicativas, mediciones de intensidad como frecuencia de aparición de palabras o sismos en donde las perturbaciones son con alta probabilidad solo positivas, y problemas de clasificación o asignación (*clustering*) en donde las métricas de error toman la forma como “correcto”/“incorrecto”, con lo que una medida de error que reporte ajustes “más incorrectos” no tiene sentido.

Una alternativa natural es considerar una métrica de desempeño distinta y diseñada específicamente en función de cada aplicación con la finalidad de capturar asimetrías, no-estacionariedad, asignación correcta e invarianzas (en el problema de *clustering* por ejemplo) entre otras propiedades. Sin embargo, esta no solo es una tarea tediosa y poco elegante — en el sentido que va en contra los objetivos de inteligencia artificial expuestos en el capítulo inicial — sino que puede ser muchas veces imprácticable, pues precisamente no conocemos cuáles son las propiedades de los datos antes de ajustar modelos. Además, usar distintas métricas dificulta la interpretación y comparación de los enfoques considerados. Consecuentemente, nos proponemos considerar un criterio global de ajuste de modelos, el que en cada caso particular *colapse* a una forma explícita que sí es *ad hoc* al problema/modelo en cuestión y permite comparar distintos enfoques de manera unificada.

El enfoque general para ajuste de modelos que consideraremos en esta sección, y continuaremos utilizando durante el resto del curso, será el *criterio de máxima verosimilitud*, dado un conjunto de datos de entrenamiento \mathcal{D} . Este es un criterio general para una amplia gama de modelos que, tal como se mencionó en el párrafo anterior, toma una forma específica en cada problema, aunque su solución no siempre es calculable de forma explícita. Este enfoque es radicalmente distinto al de MC y a cualquier otro criterio de “ajuste”: con el criterio MC se asume que ningún modelo es el modelo correcto y por lo tanto se busca un modelo *aproximado* a los datos tal que la discrepancia entre el modelo candidato y los datos sea mínima. Por el contrario, en el criterio de *máxima verosimilitud* nuestro objetivo es encontrar el modelo que — con mayor probabilidad — ha generado *exactamente* los datos observados. Debido a la naturaleza aleatoria de los datos, para implementar este concepto es necesario considerar modelos probabilísticos, de forma de poder calcular la probabilidad de que los datos \mathcal{D} hayan sido generados por un modelo m , luego, elegiremos el modelo que maximice dicha probabilidad.

El criterio de máxima verosimilitud (MV) es aplicable a modelos probabilísticos para la generación de datos, a los cuales nos referiremos como *modelos generativos*. Para el caso del problema de regresión, el modelo generativo es cualquiera que modele la variable de salida como una variable aleatoria y a través

de una distribución condicional (a la entrada x y el parámetro θ) de la forma

$$y|x, \theta \sim p(y|x, \theta), \quad (2.28)$$

donde enfatizamos que y es la única variable aleatoria y tanto el parámetro θ como la entrada x son cantidades fijas (la primera desconocida y la segunda conocida u observable).

Notación sobre variables aleatorias

Si bien la convención estándar en teoría de probabilidades es denotar las variables aleatorias con letras mayúsculas, en este apunte se seguirá la usanza de la comunidad de Aprendizaje de Máquinas donde denotamos tanto la variable aleatoria como su valor indistintamente con la letra minúscula, e.g., y . Seguiremos esta notación a menos que sea estrictamente necesario para evitar confusión. Además, en todos los casos asumiremos que las distribuciones de probabilidad consideradas tienen densidad con respecto a alguna medida — usualmente *Lebesgue* (caso continuo) o *cuenta-puntos* (caso discreto) — en ambos casos denotadas indistintamente por p . Finalmente, usualmente escribiremos

$$y|x \sim p(y|x), \quad (2.29)$$

sin enunciar explícitamente la dependencia del parámetro θ , esto con el fin de no sobrecargar la notación.

En particular, en el caso de la regresión lineal podemos considerar el siguiente modelo generativo:

$$y = a^\top x + b + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2), \quad (2.30)$$

el cual consta de una parte determinística (afín en x) y una parte aleatoria caracterizada por la variable aleatoria ϵ , la cual hemos elegido gaussiana con media cero y varianza σ_ϵ^2 (por determinar). El modelo probabilístico en la ec. (2.30) puede expresarse mediante la siguiente densidad condicional

$$y|x \sim p(y|x, \theta) = \mathcal{N}(y; a^\top x + b, \sigma_\epsilon^2), \quad (2.31)$$

donde hemos denotados el vector de todos los parámetros del modelo mediante $\theta = (a, b, \sigma_\epsilon^2)^\top$.

Si bien, lo siguiente no es necesario en el caso general, usualmente asumiremos que las realizaciones del modelo anterior, i.e., los datos $\{y_i\}_{i=1}^N$ generados a partir de la entrada $\{x_i\}_{i=1}^N$, son **condicionalmente independientes** dado el modelo. Esto significa que *si conociésemos el modelo*, o equivalentemente, si conociésemos θ , y dos entradas x_i, x_j independientes, entonces las salidas correspondientes y_i, y_j son independientes. Es importante clarificar que los valores generados por el modelo $\{y_i\}_{i=1}^N$ **no son independientes**. En efecto, si fueren independientes no podríamos hacer predicciones: la predicción de una observación nueva y_* en base a una secuencia de observaciones $\{y_i\}_{i=1}^N$ estaría dada por⁵

$$[\text{esto es falso}] \quad p(y_*|\{y_i\}_{i=1}^N) \stackrel{\text{(prob. cond.)}}{=} \frac{p(y_*, \{y_i\}_{i=1}^N)}{p(\{y_i\}_{i=1}^N)} \stackrel{\text{(indep.)}}{=} \frac{p(y_*)}{p(\{y_i\}_{i=1}^N)} = p(y_*), \quad (2.32)$$

es decir, las observaciones pasadas no aportarían para la predicción. Por el contrario, como nuestro supuesto es de **independencia condicional** la expresión correcta es la siguiente:

$$[\text{esto es verdadero}] \quad p(y_*|\{y_i\}_{i=1}^N, \theta) = p(y_*, \theta), \quad (2.33)$$

⁵Hemos ignorado la dependencia de las variables independientes $\{x_i\}_{i=1}^N, x_*$.

lo cual quiere decir que las observaciones pasadas no son útiles para predecir el futuro **solo si conozco el modelo**. Esto es evidente, pues si conozco el modelo, no necesito datos para saber de y_* .

El supuesto de independencia condicional está garantizado al imponer que las realizaciones de $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$ sean *independientes e idénticamente distribuidas* (iid). Esto es fundamental para poder aprender el modelo desde múltiples observaciones, pues intuitivamente todas las observaciones aportan evidencia no redundante sobre el parámetro en común θ . Por el contrario, si las observaciones fuesen condicionalmente dependientes, entonces la información que reportan para estimar θ sería redundante. Igualmente, si no todas las observaciones siguiesen la misma distribución, entonces cada una tendría “su propio θ ” y solo tendríamos “un dato” para estimar cada parámetro. Más adelante en el curso veremos casos donde los datos no son iid pero asumimos cierta regularidad en los modelos que permiten determinar sus parámetros.

La dependencia de las observaciones cuando el modelo es desconocido permite introducir la siguiente definición:

Definición 2.1 (verosimilitud). Consideremos un modelo generativo definido mediante la densidad de probabilidad $y \sim p(y|\theta)$, donde $\theta \in \Theta$ es el parámetro (desconocido) del modelo. Sean además $\{y_i\}_{i=1}^N$ observaciones generadas por dicho modelo. Se define $L : \Theta \rightarrow [0, 1]$ como la probabilidad de los datos observados condicional al parámetro θ , es decir,

$$\theta \mapsto L(\theta) := p(\{y_i\}_{i=1}^N | \theta) \quad (2.34)$$

Dicho valor de L se denomina *verosimilitud* del modelo $p(y|\theta)$ o, equivalentemente, del parámetro θ . En algunos casos, consideraremos las notaciones $L_y(\theta)$ y $L(\theta|\mathcal{D})$ para enfatizar que la verosimilitud es tomada con respecto a las observaciones \mathbf{y} del conjunto \mathcal{D} .

La definición anterior es elocuente: la función de verosimilitud precisamente cuantifica cuán verosímil es un modelo (o equivalentemente, un parámetro) de haber generado las observaciones $\{y_i\}_{i=1}^N$. En este sentido, ante dos valores candidatos para el parámetro, por ejemplo θ_1 y θ_2 , estos pueden ser evaluados mediante la comparación de $L(\theta_1)$ y $L(\theta_2)$. En efecto, si la razón $L(\theta_1)/L(\theta_2)$ es, por ejemplo, 3, entonces diremos que *el valor de θ sea θ_1 es 3 veces más verosímil a que sea θ_2* . En este sentido, la función de verosimilitud $L(\theta)$ representa una medida relativa de la *bondad* de cada valor que el parámetro pueda tomar en función de los datos observados.

Es importante enfatizar que la función $L(\theta)$ **no es una densidad de probabilidad**. En efecto, podemos considerar la siguiente función en dos variables: θ e $\mathbf{y} = \{y_i\}_{i=1}^N$

$$\tilde{L}(\theta, \mathbf{y}) = p(\mathbf{y} | \theta), \quad (2.35)$$

la cual toma distintos significados si fijamos una de las variables: si fijamos el valor del parámetro θ , entonces, $\tilde{L}(\theta, \cdot) = p(\cdot | \theta)$ es una densidad de probabilidad, en efecto:

$$\int_{\mathbb{R}^N} \tilde{L}(\cdot, \mathbf{y}) d\mathbf{y} = \int_{\mathbb{R}^N} p(\mathbf{y} | \theta) d\mathbf{y} = 1, \quad (2.36)$$

lo que quiere decir que para “cualquier θ ”, $p(\mathbf{y} | \theta)$ es un modelo válido. Por el contrario, si fijamos \mathbf{y} , entonces obtenemos la función de verosimilitud: $\tilde{L}(\cdot, \mathbf{y}) = p(\mathbf{y} | \cdot) = L_y(\theta)$, la cual no necesariamente integra uno con respecto a θ .

Ejemplo: Verosimilitud para el modelo gaussiano (muestras independientes)

Consideremos un modelo gaussiano definido por

$$y \sim p(y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(y-\mu)^2}{2\sigma^2}\right), \quad (2.37)$$

y las observaciones $\mathbf{y} = \{y_i\}_{i=1}^N$ iid. La verosimilitud de $\theta = (\mu, \sigma^2)^\top$ está dada por

$$\begin{aligned} L(\theta) &= p(\mathbf{y}|\mu, \sigma^2) \stackrel{\text{(iid)}}{=} \prod_{i=1}^N p(y_i|\mu, \sigma^2) = \prod_{i=1}^N \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(y_i-\mu)^2}{2\sigma^2}\right) \\ &= \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(\frac{-\sum_{i=1}^N (y_i-\mu)^2}{2\sigma^2}\right) \\ &= \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(\frac{-\left(\sum_{i=1}^N y_i^2 - 2\mu \sum_{i=1}^N y_i + N\mu^2\right)}{2\sigma^2}\right) \\ &= \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(\frac{-N\left(\sum_{i=1}^N y_i^2/N - (\sum_{i=1}^N y_i/N)^2 + (\sum_{i=1}^N y_i/N)^2 - 2\mu \sum_{i=1}^N y_i/N + \mu^2\right)}{2\sigma^2}\right) \\ &= \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(\frac{-(\sum_{i=1}^N y_i^2/N - (\sum_{i=1}^N y_i/N)^2)}{2\sigma^2/N}\right) \exp\left(\frac{-(\mu - \sum_{i=1}^N y_i/N)^2}{2\sigma^2/N}\right) \\ &= \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(\frac{-(\bar{s} - \bar{y}^2)}{2\sigma^2/N}\right) \exp\left(\frac{-(\mu - \bar{y})^2}{2\sigma^2/N}\right), \end{aligned}$$

donde $\bar{y} = \frac{1}{N} \sum_{i=1}^N y_i$ es el promedio de las observaciones y $\bar{s} = \sum_{i=1}^N y_i^2/N$ es el promedio de los cuadrados de las observaciones. Observemos que como funciones de μ y σ^2 , la expresión anterior es respectivamente proporcional a las densidades Normal (para μ) y Gamma-inversa (para σ^2). Sin embargo, recordemos que esta expresión no necesariamente integra uno y por ende es solo coincidentemente proporcional a una pdf conocida. La Fig. 8 muestra la densidad normal ($\mu = 0, \sigma = 1$) y la verosimilitud para la media y la varianza en base a 20 y 200 muestras.

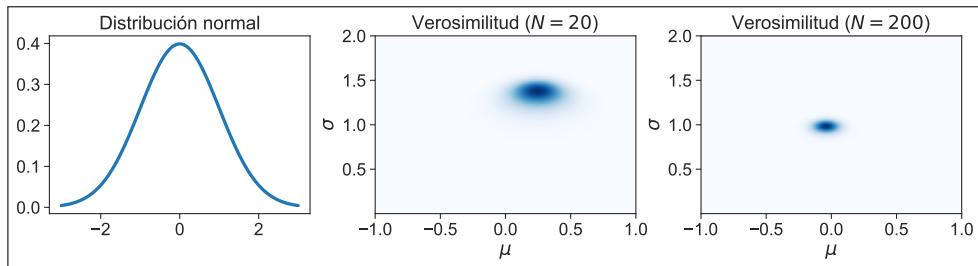


Fig. 8. Densidad normal (izquierda, $\mu = 0$ y $\sigma = 1$) y verosimilitud para la media y la varianza en base a 20 (centro) y 200 (derecha) observaciones.

2.2.1. Elección del modelo de acuerdo a la máxima verosimilitud

La verosimilitud es fundamental cuando realizamos *inferencia*, es decir, cuando nuestro objetivo es descubrir o identificar los modelos y parámetros en base a las observaciones que dicho modelo ha generado; esto muchas veces se refiere coloquialmente como *probabilidad inversa*. La importancia de la función de verosimilitud está documentado en el **principio de la verosimilitud**, el cual sentencia que toda la información relevante que la observación $\mathbf{y} = \{y_i\}_{i=1}^N$ puede aportar a la estimación del parámetro θ ,

está contenido en la función de verosimilitud $L(\theta)$. Una consecuencia directa de este principio es que si diseñamos dos experimentos para realizar inferencia sobre un parámetro desconocido θ y ambos resultan en la misma función de verosimilitud (salvo una constante de proporcionalidad), entonces, ambos experimentos, y los datos adquiridos en ellos, reportan la misma información sobre θ . Lo de igualdad salvo una constante de proporcionalidad es porque recordemos que la verosimilitud es una medida *relativa* de la bondad de (cada valor del) parámetro a inferir.

La pregunta natural entonces es ¿cómo usar la función $L_y(\theta)$ para determinar “el buen θ ”? Por supuesto el título de esta sección hace las veces de *spoiler* para esta pregunta: simplemente elegir el máximo de la función, pues este nos da el(los) valor(es) más *verosímil* para θ relativo a todo el resto de las opciones disponibles. Sin embargo, veamos que el uso del argumento que maximiza $L_y(\theta)$ tiene un significado mucho más acabado. Consideraremos la siguiente forma de encontrar el parámetro θ : recordemos que el modelo real es $p(y|\theta)$ y definamos una discrepancia entre modelos, denotada $D(p_1, p_2)$, luego, encontraremos el $\hat{\theta}$ tal que $p(y|\hat{\theta})$ es lo más *cercano* posible al modelo real con respecto a la discrepancia $D(\cdot, \cdot)$, es decir, el que minimiza la expresión

$$D(p(y|\theta), p(y|\hat{\theta})). \quad (2.38)$$

Este criterio es interesante, pues notemos que no hemos incorporado ningún supuesto sobre la parametrización de los modelos (cómo el modelo depende de θ) ni de la geometría del espacio Θ ; estamos comparando directamente los modelos y no los valores específicos de los parámetros. Desafortunadamente, notemos que formular y resolver este problema no es posible en el caso general, pues la expresión de arriba depende del parámetro real θ , el cual no conocemos, con lo que no podríamos resolver dicho problema de optimización.

Sin embargo, veamos que podemos considerar una métrica que ofrece una alternativa para optimizar la discrepancia entre el modelo real y el aproximado, independientemente de que no conozcamos el valor de θ . Dicha métrica, la cual es motivada desde la teoría de la información, es un estándar para comparar distribuciones de probabilidad generales y es conocida como la divergencia de Kullback-Leibler (ver anexos). Bajo esta métrica, la discrepancia entre el modelo real $p = p(y|\theta)$ y el aproximado $q = p(y|\hat{\theta})$ viene dada por:

$$\text{KL}(p(y|\theta), p(y|\hat{\theta})) = \int_y \log \left(\frac{p(y|\theta)}{p(y|\hat{\theta})} \right) p(y|\theta) dy. \quad (2.39)$$

En general, no es claro que podamos calcular dicha integral, sin embargo, observemos que esta es una esperanza con respecto a la densidad $p(y|\theta)$, por lo que podemos considerar su aproximación de Monte Carlo usando las N observaciones en D , las cuales están precisamente generadas por la medida de la integral en la ec. (2.39) de acuerdo a

$$\text{KL}(p(y|\theta), p(y|\hat{\theta})) \approx \text{KL}_N(p(y|\theta), p(y|\hat{\theta})) = \frac{1}{N} \sum_{i=1}^N \log \left(\frac{p(y_i|\theta)}{p(y_i|\hat{\theta})} \right). \quad (2.40)$$

Denotemos ahora $\hat{\theta}_N$ el minimizante de la expresión anterior, el cual podemos calcular mediante (ignoramos la constante N^{-1})

$$\begin{aligned}\hat{\theta}_N &= \arg \min_{\hat{\theta}} \sum_{i=1}^N \log \left(\frac{p(y_i|\theta)}{p(y_i|\hat{\theta})} \right) \\ &= \arg \min_{\hat{\theta}} \sum_{i=1}^N \log p(y_i|\theta) - \sum_{i=1}^N \log p(y_i|\hat{\theta}) \\ &= \arg \max_{\hat{\theta}} \sum_{i=1}^N \log p(y_i|\hat{\theta}) \\ &= \arg \max_{\hat{\theta}} \prod_{i=1}^N p(y_i|\hat{\theta}),\end{aligned}\tag{2.41}$$

donde hemos eliminado los términos que no dependen de $\hat{\theta}$ y se ha usado el hecho de que el logaritmo es estrictamente creciente. Observemos que si nuestras muestras son condicionalmente independientes, entonces la expresión anterior implica que $\hat{\theta}_N$ es también el maximizante de la función de verosimilitud:

$$\hat{\theta}_N = \arg \max_{\hat{\theta}} \prod_{i=1}^N p(y_i|\hat{\theta}) = \arg \max_{\hat{\theta}} p(\mathbf{y}|\hat{\theta}) = \arg \max_{\hat{\theta}} L_{\mathbf{y}}(\theta),\tag{2.42}$$

al que nos referiremos como *estimador de máxima verosimilitud*. Finalmente, queda la pregunta de cómo se relaciona el estimador de máxima verosimilitud (MV) $\hat{\theta}_N$ con el estimador óptimo en el sentido KL, $\hat{\theta}$. Para esto, notemos que la aproximación de Monte Carlo de la KL en la ec. (2.39) converge puntualmente a la KL, i.e., para cada $\theta \in \Theta$, $\text{KL}_N(p(y|\theta), p(y|\hat{\theta})) \rightarrow \text{KL}(p(y|\theta), p(y|\hat{\theta}))$ por la ley de los grandes números cuando $N \rightarrow \infty$. Consecuentemente, podemos asumir que los minimizantes de la secuencia de aproximaciones de Monte Carlo también convergen al minimizante de la KL. Esta es la razón por la cual consideramos el estimador de máxima verosimilitud: en el límite ($N \rightarrow \infty$), el estimador de MV es el que reporta la mínima divergencia (KL) entre el modelo real y el aproximado. Esta condición nos da un sentido de *consistencia* del estimador de MV, donde por consistencia entendemos que mientras más datos observamos nuestra aproximación del modelo converge al mejor modelo posible (en la métrica KL).

2.2.2. Maxima verosimilitud del modelo lineal gaussiano

Retomemos el problema de regresión lineal: la verosimilitud del modelo lineal gaussiano definido en la ec. (2.31) (con parámetro $\theta = (a, b, \sigma_\epsilon^2)^\top$) está dada por (recordemos que los datos son condicionalmente independientes)

$$L_{\mathbf{y}}(\theta) = \prod_{i=1}^N \mathcal{N}(y_i; a^\top x_i + b, \sigma_\epsilon^2) = \frac{1}{(2\pi\sigma_\epsilon^2)^{N/2}} \exp \left(\frac{-\sum_{i=1}^N (y_i - a^\top x_i - b)^2}{2\sigma_\epsilon^2} \right).\tag{2.43}$$

Usualmente, se usa el logaritmo de la verosimilitud, referido como *log-verosimilitud*, $l(\theta) = \log L(\theta)$, por su facilidad de interpretación y optimización. Recordemos que esta transformación es válida, pues la función logaritmo es estrictamente monótona y consecuentemente el(s) mínimo(s) no cambia(n). De este modo, la log-verosimilitud del modelo lineal y gaussiano está dada por

$$l(\theta) = \underbrace{-N \log \sqrt{2\pi\sigma_\epsilon^2}}_{\text{dispersión}} + \underbrace{\frac{-1}{2\sigma_\epsilon^2} \sum_{i=1}^N (y_i - a^\top x_i - b)^2}_{\text{ajuste}},\tag{2.44}$$

donde podemos de inmediato reconocer que la maximización de $l(\theta)$ implica el balance entre dos términos. El de la izquierda es una medida de dispersión o complejidad, pues para aumentar este término necesitamos que la varianza sea pequeña o el modelo tenga errores poco dispersos. El término de la derecha, por otro lado, es una medida de ajuste, para aumentar este término necesitamos que el modelo represente bien, muestra a muestra, nuestros datos.

En particular, el estimador de máxima verosimilitud para los parámetros de la parte lineal (i.e., ignorando σ_ϵ^2) está dado por:

$$[a^{\text{MV}}, b^{\text{MV}}] = \arg \min_{a,b} \sum_{i=1}^N (y_i - a^\top x_i - b)^2. \quad (2.45)$$

Para nuestra sorpresa, observemos que es posible identificar esta última expresión con la del costo de mínimos cuadrados, es decir, el estimador de máxima verosimilitud es el minimizante del mismo costo que el estimador de mínimos cuadrados. Consecuentemente, ambos estimadores son iguales y por lo tanto:

$$[\hat{a}, \hat{b}] = [a^{\text{MV}}, b^{\text{MV}}] = [a^{\text{MC}}, b^{\text{MC}}] = (\tilde{X}^\top \tilde{X})^{-1} \tilde{X}^\top Y. \quad (2.46)$$

Además, recordemos que luego de determinar el estimador con criterio de MC, es posible calcular la varianza de los errores (error cuadrático medio) de nuestro modelo mediante

$$\text{Varianza} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{a}^\top x_i - \hat{b})^2, \quad (2.47)$$

Por otra parte, en el contexto de máxima verosimilitud, recordemos que la varianza es un parámetro del modelo y no una cantidad asociada al modelo que calculamos de forma independiente. Este parámetro puede ser calculado maximizando la log-verosimilitud, tal como se hizo para la media en la ecuación (2.45), de acuerdo a

$$\sigma_{\text{MV}}^2 = \arg \max -\frac{N}{2} \log(\sigma_\epsilon^2) + \frac{-1}{2\sigma_\epsilon^2} \sum_{i=1}^N (y_i - a^\top x_i - b)^2. \quad (2.48)$$

Dado que ya se optimizó sobre los parámetros a y b , solo falta aplicar la condición de primer orden sobre σ_ϵ^2 :

$$\frac{\partial l(\theta)}{\partial \sigma_\epsilon^2} = -\frac{N}{2\sigma_\epsilon^2} + \frac{1}{2(\sigma_{\text{MV}}^2)^2} \sum_{i=1}^N (y_i - \hat{a}^\top x_i - \hat{b})^2 = 0 \Rightarrow \sigma_\epsilon^2 = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{a}^\top x_i - \hat{b})^2. \quad (2.49)$$

Con lo cual se obtiene, sin sorpresa alguna, la misma expresión de la varianza que al usar mínimos cuadrados. Además, observemos que el estimador de MV de la varianza depende de los parámetros a y b , pero no al revés.

En la práctica, consideraremos la minimización de la log-verosimilitud negativa (en vez de la maximización de la log-verosimilitud) en línea con la literatura y software dedicados a la minimización de funciones.

2.3. Regresión vía inferencia bayesiana

En el apartado anterior vimos el ajuste de modelos, estimación de parámetros o *inferencia*, mediante la maximización de la función de verosimilitud. Es decir, en base a un conjunto de observaciones \mathbf{y} asignamos el valor más *verosímil* al parámetro desconocido θ , dado por $\hat{\theta}^* = \arg \max L_{\mathbf{y}}(\hat{\theta})$. Es claro por

qué esta estimación puntual tiene sentido, desde el punto de vista de la probabilidad de los datos y de la mínima discrepancia contra el modelo real en base a la divergencia de Kullback-Leibler. Sin embargo, solo tomar el máximo ignora el resto de la información contenida en la función de verosimilitud, por ejemplo, si consideramos funciones de verosimilitud distintas (bimodales, con colas pesadas/livianas, asimétricas, discretas, etc.), todas esa propiedades no se reflejan en la estimación del parámetro θ si estas verosimilitudes coinciden en el máximo. Esta es una limitación fundamental no solo del método de máxima verosimilitud, sino que de la estimación puntual en general.

Otra limitación del paradigma de máxima verosimilitud es que no da la posibilidad de incorporar conocimiento experto, es decir, introducir sesgos necesarios para la inferencia. Por ejemplo, si sabemos que el parámetro desconocido tiene magnitud pequeña, o está lejos del origen, o es ralo, etc. Esta propiedad es necesaria en el análisis moderno de datos, pues en la ciencia de datos los esfuerzos son colaborativos y el conocimiento experto sin duda ayuda a resolver problemas de forma eficiente y con interpretabilidad.

El paradigma bayesiano busca conciliar estas dos desventajas del uso de máxima verosimilitud interpretando el parámetro θ como variable aleatoria, donde la disponibilidad de datos se interpreta como un evento que aporta evidencia sobre el valor de θ . Consecuentemente, el proceso de inferencia ahora se centra en encontrar la distribución condicional $p(\theta|\text{datos})$.

2.3.1. ¿Qué es ser bayesiano?

Thomas Bayes (c. 1701-1761) fue un matemático, filósofo y pastor presbiteriano inglés interesado en el cálculo y uso de las probabilidades. En ese entonces, no existía la diferencia entre probabilidad descriptiva (que caracteriza la generación de datos dado un modelo) e inferencial (que nos permite identificar un modelo dado un conjunto de observaciones). Como vimos en la clase pasada, el uso de probabilidades para estimar (o *inferir*) parámetros usando los propios datos recibía el nombre de *probabilidad inversa*. Tanto Bayes, como el matemático francés Pierre Simon Laplace, estaba al tanto de la siguiente relación para el problema de inferencia, la cual hoy conocemos como el *Teorema de Bayes*:

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)} \propto p(x|\theta)p(\theta), \quad (2.50)$$

donde $x \in \mathcal{X}$ es el valor de una observación y $\theta \in \Theta$ es un parámetro. En la expresión anterior podemos identificar las siguientes cantidades:

- el prior o distribución a priori: $p(\theta)$,
- la verosimilitud: $p(x|\theta)$,
- la distribución posterior: $p(\theta|x)$,
- la densidad marginal de x : $p(x) = \int_{\Theta} p(x|\theta)p(\theta)d\theta$.

Recordemos que en el problema de inferencia, los datos (en este caso x) son conocidos y fijos, mientras que el parámetro es variable (en realidad desconocido). Por esta razón, podemos escribir el lado derecho de la ec. (2.50), pues nos importa la distribución posterior como función de θ únicamente, tal como fue el caso de la función de verosimilitud en el apartado anterior. En algunas aplicaciones, conocer la versión proporcional de la posterior dada por $p(x|\theta)p(\theta)$ es suficiente para realizar distintos análisis, por ejemplo, cuando usamos Markov chain Monte Carlo.

Bajo cualquier punto de vista de inferencia estadística, e.g., bayesiano u otro, el espacio que contiene los valores de x debe tener suficiente *estructura* para definir probabilidades (o, equivalentemente, modelos); en particular aquellos de la forma $p(x)$ y $p(x|\theta)$. Adicionalmente, una característica particular del

enfoque bayesiano es que asume que el espacio donde se encuentra el parámetro θ , denotado Θ , también debe ser un espacio de probabilidad. El enfoque frecuentista, por el contrario, tiene un concepto de probabilidad totalmente distinto que resulta en la consideración de θ como un elemento fijo, como un índice o hipótesis cuyo valor queremos descubrir.

La inferencia bayesiana se sustenta en la noción de probabilidad como medida de incertidumbre, es decir, en una perspectiva subjetiva del conocimiento disponible sobre la generación del valor x . En este sentido, es posible identificar dos tipos de incertidumbre, la primera es la incertidumbre **aleatoria** y dice relación con la variabilidad con la que el sistema/modelo en cuestión genera el dato x . El segundo tipo es la llamada incertidumbre **epistemológica** y representa nuestra inhabilidad de conocer el modelo que genera los datos. Consecuentemente, el enfoque bayesiano hace la distinción entre ambos tipos de incertidumbre mediante los siguiente elementos:

- La verosimilitud $p(x|\theta)$: que modela la aleatoriedad del modelo, el cual produce datos de forma aleatoria incluso cuando el modelo es perfectamente conocido. Este tipo de incertidumbre no puede ser reducida observando datos. **Ejemplo:** En una urna con bolitas rojas y negras donde la probabilidad de elegir una bolita negra es θ , incluso si θ fuese conocido, la incertidumbre aleatoria del modelo resulta en la imposibilidad de predecir el color de la próxima bolita.
- La distribución a priori $p(\theta)$: que encapsula la incertidumbre epistemológica (lo que no sabemos) sobre el sistema, la cual pude ser reducida observando datos y calculando $p(\theta|x)$ mediante el Teorema de Bayes. Distintas distribuciones a priori llevarán a distintas distribuciones a posteriori, lo cual establece la subjetividad del enfoque bayesiano. **Ejemplo:** En la misma urna anterior, el no conocer θ es un ejemplo de incertidumbre epistemológica.

Una discusión trascendental desde los inicios del enfoque bayesiano apuntaba a cómo elegir la distribución a priori $p(\theta)$ para ser lo más objetivo posible, es decir, cómo no introducir sesgos en la inferencia (cálculo de la posterior) heredados de una elección subjetiva del prior — este era el objetivo de los llamados *bayesianos objetivistas*. La postura de Laplace con respecto de esta disyuntiva fue elegir simplemente un prior *no-informativo*, i.e., uno que no asigne más probabilidad a ningún valor de θ en particular, para lo cual se puede elegir un prior uniforme $p(\theta) \propto 1$. Sin embargo, posteriormente se descubrió que esta elección introduce sesgo en la inferencia de todas formas.⁶ Hacia fines del siglo XIX comenzó a surgir una demanda por un tratamiento i) objetivo de la inferencia que no dependiese de la elección del prior, algo que los *bayesianos objetivistas* no habían garantizado, y ii) riguroso y formal en el sentido de que fuese consistente con la teoría matemática. En este sentido, durante el siglo XX la inferencia estadística vio avances en la dirección de prescindir del uso del prior para evitar inferencias sesgadas, en particular, el foco estuvo en la maximización directa de la verosimilitud con respecto al *índice* θ , tal como vimos en la sección anterior. De forma más general, y a través de Fisher, Neyman y Pearson, fue posible construir un tratamiento formal de la inferencia estadística sobre la base de una interpretación frecuentista de la probabilidad, esto es, definir la probabilidad de un evento como el límite de la frecuencia de *casos favorables dividido por casos totales* cuando el número de realizaciones de un experimento tiende a infinito. Entonces, al considerar los parámetros como cantidades fijas y desconocidas, y al mismo tiempo, concebir la probabilidad como una medida de frecuencias y no de incertidumbre, el punto de vista frecuentista no busca asociar probabilidades a los parámetros desconocidos.

En la segunda mitad del siglo XX hubo un resurgimiento de la inferencia Bayesiana, en parte por el descontento de algunos estadísticos (y científicos en general) con las métricas frecuentistas estándar como los errores de Tipo-I & Tipo-II, y los llamados *p*-valores. Esto permitió utilizar la maquinaria desarrollada por la perspectiva frecuentista para formalizar el enfoque bayesiano, en particular, esta

⁶Esto es consecuencia de que el prior uniforme no es invariante bajo transformaciones uno-a-uno de θ , consecuentemente, si el modelo es re-parametrizado de forma no lineal y se mantiene el prior uniforme, entonces la posterior cambia.

combinación permitió construir priors no informativos invariantes bajo transformaciones una-a-uno (e.g., el prior de Jeffreys), como también definir el concepto de *consistencia*, i.e., si un estimador converge a la respuesta correcta y a qué velocidad ocurre esto. Esta unión entre los conceptos bayesianos y frecuentistas resulta en lo mejor de dos mundos: hace posible construir un modelo que es subjetivo, pues en la práctica queremos incorporar conocimiento experto en los problemas que estemos estudiando, pero al mismo tiempo tenemos una herramienta objetiva (pues el enfoque frecuentista es *automático* en el sentido que no requiere conocimiento experto) para asegurar que nuestros métodos son rigurosos desde una perspectiva matemática.

2.3.2. Elección de prior: conjugación

Desde ahora, nuestra misión será calcular y analizar la distribución posterior del parámetro θ dado un conjunto de datos D , nos referimos a esto como *inferencia bayesiana* a diferencia de la más general *inferencia estadística* en la cual a veces se consideran solo estimaciones puntuales de θ . La distribución posterior está dada, mediante el teorema de Bayes, por

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \quad (2.51)$$

donde $p(D|\theta)$ es la verosimilitud o el modelo que elegimos y $p(\theta)$ es la *distribución a priori* del parámetro y encapsula todos nuestros supuestos, creencias y sesgo sobre el espacio de parámetros (modelos) a considerar. De este modo, $p(\theta)$ funciona como ponderador de la verosimilitud de acuerdo a la importancia que se le dé a θ en el prior. Finalmente, recordemos que en el denominador encontramos la *distribución marginal de los datos* $p(D)$ que actúa como constante de normalización para el numerador, pues el numerador puede ser considerado como una densidad pero no de probabilidad. Esta constante de normalización puede ser calculada mediante el uso de la ley de probabilidades totales o, equivalentemente, imponiendo la restricción de que la expresión en la ecuación (2.51) debe integrar uno:

$$p(D) = \int p(D|\theta)p(\theta)d\theta. \quad (2.52)$$

En base a la forma explícita de $p(D|\theta)p(\theta)$, calcular esta integral puede ser un desafío considerable. Sin embargo, enfatizamos que como esta cantidad no depende del parámetro θ , no es necesario conocerla para explorar o aproximar la (forma de la) distribución posterior $p(\theta|D)$.

Entonces, con el modelo $p(D|\theta)$ acordado, solo nos queda elegir la distribución a priori, los cual es guiado por dos objetivos. En primer lugar, debemos encapsular lo que efectivamente que sabemos del parámetro θ , por ejemplo, en el caso de regresión lineal, podemos tener evidencia que los datos que observamos representan una cantidad que creciente en el tiempo, en cuyo caso, sabemos que $\theta \geq 0$. El segundo objetivo es obtener una forma *amigable* de la distribución posterior, en el sentido que esta sea un distribución con propiedades que deseemos, en particular, que la podamos calcular, evaluar, y samplear de ella. Una práctica usual en este sentido es elegir un prior $p(\theta)$ tal que la distribución posterior $p(\theta|D)$ están en la misma familia:

Definición 2.2 (prior conjugado). Diremos que el prior $p(\theta)$ es *conjugado* a la verosimilitud $p(D|\theta)$, cuando la posterior $p(\theta|D)$ está en la misma familia, es decir, tienen la misma distribución con parámetros distintos.

El uso de un prior arbitrario resulta en que la posterior tenga una forma arbitraria también, con lo que incluso si tanto el prior como la verosimilitud tienen formas *conocidas*, no tenemos ninguna garantía de que el posterior también la tenga y consecuentemente sea difícil de interpretar y calcular. Por el contrario, si usamos priors conjugados la distribución posterior pertenece a *la misma familia* del prior cuando vamos observando más datos, lo cual tiene un significado relevante: la actualización de prior a

posterior al actualizar la verosimilitud debido a la incorporación de datos (conocida como actualización bayesiana) es simplemente un cambio de parámetros, lo cual ofrece una clara interpretación (en el caso que los parámetros tengan significado como media, varianza, o alguna taza), y la nueva distribución ocupan la misma cantidad de memoria que el prior (pues la cantidad de parámetros no cambia). A continuación veremos dos ejemplos de priors conjugados y cómo se interpreta la variación que sufren los parámetros de la posterior al incorporar datos:

1. Modelo gaussiano Consideremos un conjunto de observaciones⁷ $\mathcal{D} = \{x_i\}_{i=1}^n \subset \mathbb{R}$ generadas independiente e idénticamente distribuidas (iid) por el modelo $\mathcal{N}(\mu, \sigma^2)$. Recordemos que la verosimilitud de la media y varianza respectivamente está dada por

$$L_{\mathcal{D}}(\mu, \sigma^2) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x_i - \mu)^2\right). \quad (2.53)$$

A continuación veremos priors conjugados para esta verosimilitud de forma incremental: primero cuando sola la media μ es desconocida, luego cuando solo la varianza σ^2 es desconocida y finalmente cuando ambos parámetros son desconocidos.

- (σ^2 conocido). Consideremos el prior sobre la media $p(\mu) = \mathcal{N}(\mu_0, \sigma_0^2)$ donde μ_0 y σ_0^2 son parámetros fijados (denominados hiperparámetros) y por lo tanto conocidos. Bajo este prior, la posterior está dada por:

$$\begin{aligned} p(\mu|\mathcal{D}) \propto p(\mathcal{D}|\mu)p(\mu) &\propto \left(\prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x_i - \mu)^2\right) \right) \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\left(-\frac{1}{2\sigma_0^2}(\mu - \mu_0)^2\right) \\ &\propto \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2 - \frac{1}{2\sigma_0^2}(\mu - \mu_0)^2\right), \end{aligned} \quad (2.54)$$

donde la proporcionalidad viene de ignorar la constante $p(\mathcal{D})$ en la primera línea e ignorar todas las constantes que no dependen de μ en la segunda línea. Recordemos que estas constantes para μ incluyen a la varianza de x , σ^2 , por lo que ignorar esta cantidad es solo posible debido a que estamos considerando el caso en que σ^2 es conocido.

Por otra parte, es necesario reordenar los términos dentro de la exponencial en la ec. (2.54), para verificar que efectivamente la posterior tiene estructura de distribución normal. En efecto:

$$p(\mu|\mathcal{D}) \propto \exp\left(-\frac{1}{2\sigma_n^2}(\mu - \mu_n)^2\right), \quad (2.55)$$

donde (ya definiremos μ_n y σ_n^2 en breve) como $p(\mu|\mathcal{D})$ debe integrar uno, la única densidad de probabilidad proporcional al lado derecho de la ecuación anterior es la Gaussiana de media μ_n y varianza σ_n^2 . Es decir, la constante de proporcionalidad necesaria para la igualdad en la expresión anterior es $\int_{\mathbb{R}} \exp\left(-\frac{1}{2\sigma_n^2}(\mu - \mu_n)^2\right) d\mu = (2\pi\sigma_n^2)^{n/2}$. Consecuentemente, confirmamos que el prior elegido era efectivamente conjugado con la verosimilitud gaussiana y la posterior está dada por la siguiente gaussiana:

$$p(\mu|\mathcal{D}) = \mathcal{N}(\mu; \mu_n, \sigma_n^2) = \frac{1}{(2\pi\sigma_n^2)^{n/2}} \exp\left(-\frac{1}{2\sigma_n^2}(\mu - \mu_n)^2\right), \quad (2.56)$$

⁷Observe que en esta sección no estamos solo enfocados en el problema de regresión, sino que cualquiera que requiera inferencia paramétrica bayesiana.

donde la media y la varianza están dadas respectivamente por

$$\mu_n = \frac{1}{\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}} \left(\frac{1}{\sigma_0^2} \mu_0 + \frac{n}{\sigma^2} \bar{x} \right), \quad \text{donde } \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (2.57)$$

$$\sigma_n = \left(\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2} \right)^{-1}. \quad (2.58)$$

Observación 2.1. La actualización bayesiana transforma los parámetros del prior de μ desde μ_0 y σ_0^2 hacia μ_n y σ_n^2 en las ecs. (2.57) y (2.58) respectivamente. Notemos que los parámetros de la posterior son combinaciones (interpretables por lo demás) entre los parámetros del prior y los datos, en efecto, la μ_n es el promedio ponderado entre μ_0 (que es nuestro candidato para μ antes de ver datos) con factor σ_0^{-2} y el promedio de los datos \bar{x} con factor $(\sigma^2/n)^{-1}$, que a su vez es el estimador de máxima verosimilitud. Es importante también notar que estos factores son las varianzas inversas—i.e., precisión—de μ_0 y de \bar{x} . Finalmente, observemos que σ_n es la *suma paralela* de las varianzas, pues si expresamos la ec. (2.58) en términos de *precisiones*, vemos que la precisión inicial σ_0^2 aumenta un término σ^2 con cada dato que vemos; lo cual tiene sentido pues con más información es la precisión la que debe aumentar y no la incertidumbre (en este caso representada por la varianza).

- (μ conocido). Ahora procedemos con el siguiente prior para la varianza, llamado Gamma-inverso:

$$p(\sigma^2) = \text{inv-}\Gamma(\sigma^2; \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)(\sigma^2)^{\alpha+1}} \exp(-\beta/\sigma^2) \quad (2.59)$$

esta densidad recibe dicho nombre pues es equivalente a modelar la precisión, definida como el recíproco de la varianza $1/\sigma^2$, mediante la distribución Gamma. Los hiperparámetros α y β son conocidos como parámetros de forma y de tasa (o precisión) respectivamente.

Con este prior, la posterior de la varianza toma la forma:

$$\begin{aligned} p(\sigma^2 | \mathcal{D}) &\propto \left(\prod_{i=1}^N \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x_i - \mu)^2\right) \right) \frac{\beta^\alpha}{\Gamma(\alpha)(\sigma^2)^{\alpha+1}} \exp(-\beta/\sigma^2) \\ &\propto \frac{1}{(\sigma^2)^{N/2+\alpha+1}} \exp\left(-\frac{1}{\sigma^2} \left(\frac{1}{2} \sum_{i=1}^N (x_i - \mu)^2 + \beta \right)\right) \end{aligned} \quad (2.60)$$

donde nuevamente la proporcionalidad ha sido mantenida debido a la remoción de las constantes. Esta última expresión es proporcional a una distribución Gamma inversa con hiperparámetros α_n y β_n , es decir:

$$p(\sigma^2 | \mathcal{D}) \sim \text{inv-}\Gamma(\sigma^2; \alpha_n, \beta_n) \quad (2.61)$$

Donde $\alpha_n = \frac{n}{2} + \alpha$ y $\beta_n = \frac{1}{2} \sum_{i=1}^N (x_i - \mu)^2 + \beta$.

Ejemplo: distribución posterior del modelo lineal y gaussiano

Recordemos que, para observaciones $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, el modelo de regresión lineal puede ser escrito en forma vectorial mediante

$$Y = \tilde{X}\theta + \epsilon, \quad (2.62)$$

donde $[\tilde{X}]_{i:} = [x_i^\top, 1]$, $[Y]_i = y_i$, $[\epsilon]_i = \epsilon_i \sim \mathcal{N}(0, \sigma^2)$, $\theta = [a; b]$.

Por lo tanto, su verosimilitud está dada por:

$$\begin{aligned} L(\theta, \sigma^2) &= \text{MVN}(Y; \tilde{X}\theta, \mathbb{I}\sigma^2) \\ &\propto (\sigma^2)^{-n/2} \exp\left(-\frac{1}{2\sigma^2}(Y - \tilde{X}\theta)^\top(Y - \tilde{X}\theta)\right), \end{aligned} \quad (2.63)$$

donde la distribución MVN denota la normal multivariada. Observe que esta última expresión es proporcional a una distribución Gamma-Inversa para σ^2 y proporcional a una MVN para θ . Consecuentemente, esta verosimilitud tiene los mismos priors conjugados que el modelo gaussiano en la ec. (2.37). Consideremos entonces el caso en que σ^2 es conocido y elegimos el prior gaussiano para θ dado por

$$p(\theta) \propto \exp\left(-\frac{1}{2\sigma^2}(\theta - \theta_0)^\top \Lambda_0(\theta - \theta_0)\right), \quad (2.64)$$

Entonces, se obtiene una distribución posterior dada por $\text{MVN}(\theta; \theta_n, \sigma^2 \Lambda_n^{-1})$ con parámetros

$$\theta_n = (\tilde{X}^\top \tilde{X} + \Lambda_0)^{-1}(\tilde{X}^\top Y + \Lambda_0 \theta_0) = (\tilde{X}^\top \tilde{X} + \Lambda_0)^{-1}(\tilde{X}^\top \tilde{X}(\tilde{X}^\top \tilde{X})^{-1} \tilde{X}^\top Y + \Lambda_0 \theta_0) \quad (2.65)$$

$$\Lambda_n = (\tilde{X}^\top \tilde{X} + \Lambda_0). \quad (2.66)$$

Es decir, la media posterior θ_n es un promedio ponderado entre la media a priori θ_0 y el estimador de máxima verosimilitud $(\tilde{X}^\top \tilde{X})^{-1} \tilde{X}^\top Y$. Observemos además cómo la varianza Λ_n se mueve desde la varianza a priori Λ_0 hacia $(\tilde{X}^\top \tilde{X})^{-1}$ a medida recibimos más observaciones, resultando en un modelo más preciso.

La Figura 9 muestra una implementación de la regresión lineal bayesiana, para el modelo

$$y = ax + b + \epsilon = 2x - 2 + \epsilon, \quad (2.67)$$

donde $\epsilon \sim \mathcal{N}(0, 0.5^2)$. Desde arriba hacia abajo, se han considerado $\{0, 1, 2, 50\}$ observaciones, donde cada columna de la figura muestra (de izquierda a derecha): los datos observados, la distribución conjunta de los parámetros a y b , las distribuciones marginales de los parámetros, y el modelo real (azul) junto a muestras del modelo posterior (rojo). Observe cómo rápidamente las distribuciones posteriores se concentran en los valores de los parámetros reales.

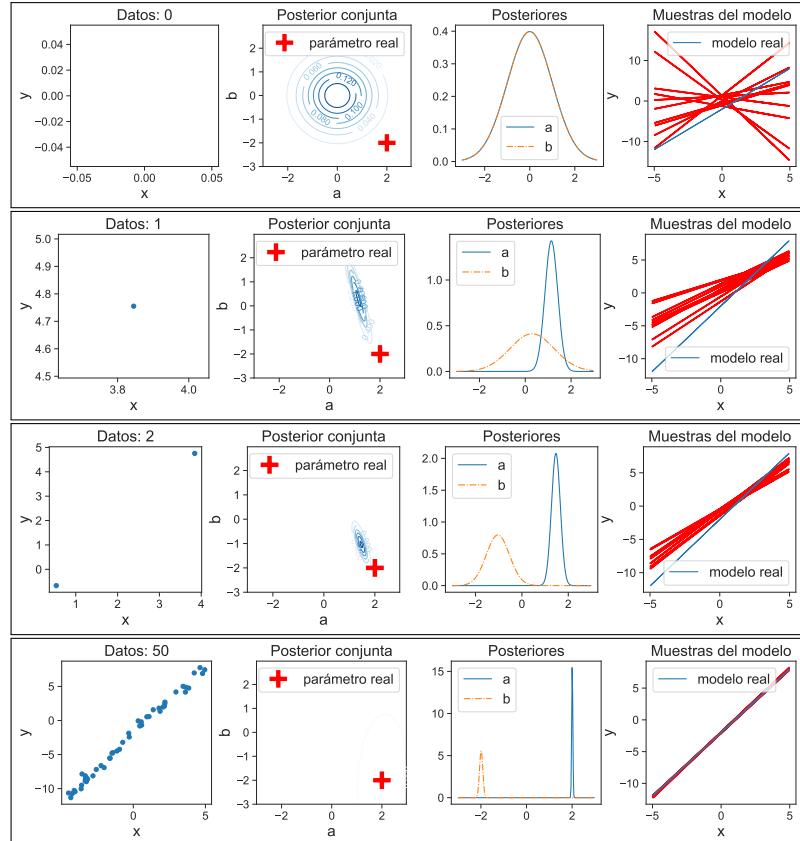


Fig. 9. Regresión lineal bayesiana

2. Modelo binomial Ahora ilustraremos la elección de la distribución a priori a través de un segundo ejemplo basado en el modelo binomial, el cual fue considerado en el artículo original de (Bayes, 1763). Al comienzo de su artículo, Bayes enuncia el problema que motiva su trabajo:

Given the number of times in which an unknown event has happened and failed: Required the chance that the probability of its (specific event) happening in a single trial lies somewhere between any two degrees of probability that can be named.

A pesar de lo confuso de la jerga usada por Bayes, al menos ya podemos notar que se hace una diferencia entre la probabilidad de los datos (*probability*) y la del modelo (*chance*). Si tradujéramos esta formulación del problema al español y con una notación moderna, diríamos:

Dados n lanzamientos Bernoulli iid con parámetro θ donde se han registrado k aciertos, ¿cuál es la probabilidad de que el parámetro θ se encuentre entre dos cotas dadas?

Consideremos entonces el evento de obtener “ k aciertos en n intentos”. Por ejemplo anotar k goles con n intentos de penales, u obtener k veces un número par al lanzar un dado n veces. La probabilidad de obtener entonces los “ k aciertos en n intentos” puede ser modelada mediante una distribución binomial, la cual asume que cada acierto es independiente y equiprobable con probabilidad θ (o bien, *Bernoulli*). La verosimilitud de este modelo (con parámetro θ) está dada por la distribución binomial de los datos (n lanzamientos y k aciertos) dada por

$$p(k|n, \theta) = \binom{n}{k} \theta^k (1 - \theta)^{n-k}. \quad (2.68)$$

Antes de proceder con el prior conjugado para esta verosimilitud, revisemos qué hizo T. Bayes. Él sabía que la distribución posterior era proporcional a la verosimilitud porque consideró—implícitamente—un prior **uniforme** para $\theta \in [0, 1]$ (el caso binomial fue el único considerado por Bayes). La elección del prior uniforme tiene sentido, pues θ es una probabilidad de la cual, *a priori*, sabemos nada. Consecuentemente, como solo conocemos una versión proporcional de la posterior a través del Teorema de Bayes, la posterior es necesariamente (recordemos que consideramos un prior uniforme)

$$p(\theta|n, k) = \frac{p(n, k|\theta)}{\int_0^1 p(n, k|\theta) d\theta} = \frac{\theta^k (1 - \theta)^{n-k}}{\int_0^1 \theta^k (1 - \theta)^{n-k} d\theta} = \frac{\theta^k (1 - \theta)^{n-k}}{\mathcal{B}(k+1, n-k+1)}, \quad (2.69)$$

donde $\mathcal{B}(\alpha, \beta) = \int_0^1 x^{\alpha-1} (1-x)^{\beta-1} dx$ es conocida como la función Beta.

Este resultado era al que Bayes llegó en su artículo, aunque no de forma explícita. Sin embargo, al no entregar una forma cerrada para esta probabilidad, el trabajo de Bayes era difícil de implementar. Actualmente, el cálculo de esta constante de proporcionalidad no es problemático⁸ pues sabemos que estas son densidades de probabilidad que integran 1, un concepto que no existía en la era de Bayes.

A simple vista, el prior uniforme propuesto por Bayes no es conjugado, pues la posterior no es uniforme. Sin embargo, al darle una mirada más detallada, podemos identificar que este sí es un caso particular de un prior conjugado. En efecto, consideremos el prior dado por una distribución Beta para θ , con parámetros (α, β) , con densidad dada por

$$p(\theta) = \text{Beta}(\theta; \alpha, \beta) = \frac{\theta^{\alpha-1} (1-\theta)^{\beta-1}}{\mathcal{B}(\alpha, \beta)}, \quad \theta \in [0, 1], \quad (2.70)$$

⁸En particular conocemos la relación entre la función Beta y la función Gamma, $\mathcal{B}(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$, que facilitan el cálculo de la función Beta.

donde la función Beta definida anteriormente actúa como constante de normalización. Notemos que eligiendo $\alpha = \beta = 1$, obtenemos que este prior es efectivamente la densidad uniforme entre 0 y 1. Además, observemos el rol de los parámetros de la distribución Beta, la cual está formada por la multiplicación de dos potencias de θ con ceros en $\theta = 0$ y $\theta = 1$, donde α y β representan, informalmente, los pesos relativos entre los aciertos y fallos respectivamente. De hecho, se prueba que $E(\theta|\alpha, \beta) = \frac{\alpha}{\alpha+\beta}$.

Para calcular la posterior que resulta del uso del prior Beta, veamos un caso un poco más general que el anterior. Consideremos las observaciones $\mathcal{D} = \{(n_i, k_i)\}_{i=1}^N$ correspondientes a N juegos independientes (no solo uno como el caso anterior), donde el i -ésimo juego consistió en n_i intentos y k_i aciertos. Con estos datos y el prior $Beta(\theta; \alpha, \beta)$ de la ec. (2.70), la (versión proporcional de la) distribución posterior de θ está dada por

$$\begin{aligned} p(\theta|\mathcal{D}) &\propto \left(\prod_{i=1}^N p(k_i|n_i, \theta) \right) p(\theta) \\ &\propto \left(\prod_{i=1}^N \binom{n_i}{k_i} \theta^{k_i} (1-\theta)^{n_i-k_i} \right) \theta^{\alpha-1} (1-\theta)^{\beta-1} \\ &\propto \theta^{\sum k_i + \alpha - 1} (1-\theta)^{\sum(n_i - k_i) + \beta - 1}, \end{aligned} \quad (2.71)$$

donde los símbolos de proporcionalidad se han mantenido debido a la remoción de constantes y hemos usado la notación compacta $\sum k_i = \sum_{i=1}^N k_i$. Notemos que la última expresión es proporcional a la definición de distribución Beta en la ecuación (2.70), por lo que ajustando la constante de proporcionalidad tenemos que, al igual que el prior, la posterior es Beta también:

$$p(\theta|\mathcal{D}) = Beta\left(\theta; \sum_{i=1}^N k_i + \alpha, \sum_{i=1}^N (n_i - k_i) + \beta\right). \quad (2.72)$$

Vemos entonces explícitamente cómo la distribución posterior es nuevamente una *mezcla* entre el prior y la verosimilitud en función de los parámetros, los cuales de prior a posterior han cambiado:

$$\alpha \rightarrow \alpha + \sum_{i=1}^N k_i \quad \beta \rightarrow \beta + \sum_{i=1}^N (n_i - k_i), \quad (2.73)$$

lo cual es consistente con la interpretación inicial de los parámetros de la distribución Beta: el peso de los aciertos (α) se modifica en función de la suma de todos los aciertos y el peso de los fracasos (β) se modifica en función de la suma de todos los fracasos.

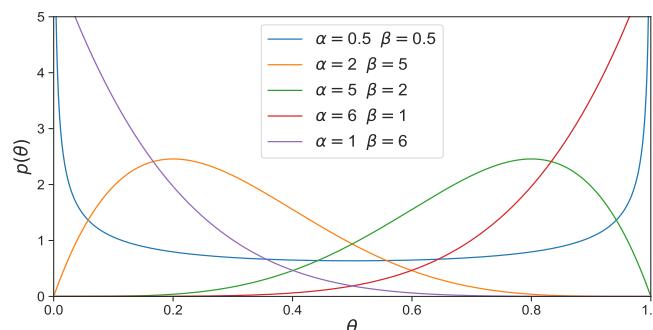


Fig. 10. Elasticidad de la distribución beta de acuerdo a distintos parámetros. Se observa la interpretación del parámetro α como aciertos y β como fracasos.

Adicionalmente, notemos que la magnitud de los parámetros va aumentando. La consecuencia de esto es que la varianza de la posterior va disminuyendo, pues

$$\mathbb{V}(\theta|\alpha, \beta) = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)} \rightarrow 0 \text{ cuando } \alpha, \beta \rightarrow 0 \quad (2.74)$$

con lo que la incertidumbre de la distribución posterior de θ va disminuyendo a medida que vemos más observaciones.

Ejemplo: modelo binomial

Apliquemos la actualización bayesiana en el modelo binomial. Consideremos la variable binomial k dada por

$$k \sim \binom{n}{k} \theta^k (1 - \theta)^{n-k}, \quad (2.75)$$

con probabilidad $\theta = 0.66$ (desconocida). Nuestro objetivo es ver cómo depende la distribución posterior de θ de la elección del prior a medida que vamos obteniendo más observaciones. Para esto, elegimos 20 priors Beta distintos para θ dados por

$$p_i(\theta) = \text{Beta}(\alpha_i, \beta_i), \quad i = 1, \dots, 20, \quad (2.76)$$

donde $\alpha_i, \beta_i \sim U(1, 10)$ (iid y discreto).

Ahora, realizaremos tres experimentos, en los cuales elegimos $n \in \{10, 100, 1000\}$ lanzamientos y calculamos los parámetros de las distribuciones a posteriori correspondientes a cada prior de acuerdo a la ecuación (2.73). En la Figura 11 podemos ver, desde izquierda a derecha, las distribuciones a priori elegidas y luego las posteriores correspondientes a 10, 100 y 1000 observaciones. Notemos cómo a medida obtenemos más y más datos, la elección del prior es vuelve irrelevante y, como es de esperar, la evidencia de los datos es suficiente para aprender el modelo exitosamente (pues $\theta = 0.66$) sin depender del prior.

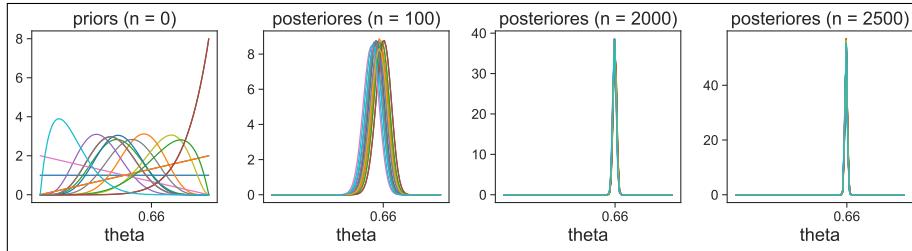


Fig. 11. Concentración de la distribución posterior en el modelo binomial.

2.3.3. Máximo a posteriori

Como ha sido el criterio en esta sección, mediante la distribución posterior podemos representar toda la información que nuestros sesgos y los datos aportan a la caracterización de un modelo (o parámetro), es decir, simetrías, barras de error, momentos, etc. En particular, es relevante verificar cómo podemos obtener estimaciones puntuales, en una forma similar a las obtenidas mediante mínimos cuadrados o máxima verosimilitud, a través del análisis de la distribución posterior. La motivación para la obtención de estimaciones puntuales viene de i) la necesidad de comparar dichas estimaciones con las obtenidas por MC, MCR y MV, y además ii) los casos donde en efecto necesitamos una estimación puntual y no distribucional, como el problema de toma de decisiones.

Hay distintas alternativas evidentes para extraer una estimación puntual del parámetro θ desde la distribución $p(\theta|\mathcal{D})$, como la media, la mediana y la moda, las cuales son equivalentes cuando la posterior es Gaussiana (o unimodal y simétrica en general). Siguiendo un criterio similar al de máxima verosimilitud consideraremos estimaciones puntuales mediante la maximización de la distribución posterior, consecuentemente, resumiendo la información de la posterior mediante su moda.

Definición 2.3 (Máximo a posteriori). Sea $\theta \in \Theta$ un parámetro con distribución posterior $p(\theta|\mathcal{D})$ definida en todo Θ , entonces nos referimos a estimación puntal dada por

$$\theta_{\text{MAP}} = \arg \max_{\Theta} p(\theta|\mathcal{D}), \quad (2.77)$$

como *máximo a posteriori (MAP)*.

Notemos que es posible encontrar el MAP incluso cuando solo tenemos acceso a una versión *proporcional* a la distribución posterior, un escenario usual en inferencia bayesiana, o también mediante la maximización del logaritmo de esta última. En efecto,

$$\theta_{\text{MAP}} = \arg \max_{\theta \in \Theta} p(\theta|\mathcal{D}) = \arg \max_{\theta \in \Theta} p(\mathcal{D}|\theta)p(\theta) = \arg \max_{\theta \in \Theta} \left(\underbrace{\log p(\mathcal{D}|\theta)}_{l(\theta)} + \log p(\theta) \right), \quad (2.78)$$

donde nuevamente vemos la maximización de la función de log-verosimilitud, pero ahora junto al log-prior. Es evidente de esta expresión que si el prior no depende de θ , es decir, si es uniforme, entonces el criterio MAP es equivalente al MV.

Ejemplo: máximo a posterior para el modelo lineal y gaussiano

En particular, para el modelo lineal y gaussiano que hemos considerado hasta ahora, podemos calcular θ_{MAP} para un prior gaussiano multivariado donde cada coordenada de θ tendrá un prior independiente de media cero y varianza σ_θ^2 (es decir, la matriz de covarianzas será diagonal). Asumiendo la varianza del ruido σ_ϵ^2 conocida, se tiene que:

$$\begin{aligned} \theta_{\text{MAP}}^* &= \arg \max_{\theta} p(Y|\tilde{X}, \theta)p(\theta) \\ &= \arg \max_{\theta} \left(\prod_{i=1}^N \mathcal{N}(y_i; \theta^\top \tilde{x}_i, \sigma_\epsilon^2) \right) \text{MVN}(\theta; 0, \sigma_\theta^2 \mathbb{I}) \\ &= \arg \max_{\theta} \left(\prod_{i=1}^N \frac{1}{\sqrt{2\pi}\sigma_\epsilon} \exp \left[\frac{-1}{2\sigma_\epsilon^2} (y_i - \theta^\top \tilde{x}_i)^2 \right] \right) \frac{1}{(\sqrt{2\pi}\sigma_\theta)^{M+1}} \exp \left(\frac{-\theta^\top \theta}{2\sigma_\theta^2} \right) \\ &= \arg \max_{\theta} \frac{1}{(\sqrt{2\pi}\sigma_\epsilon)^N} \frac{1}{(\sqrt{2\pi}\sigma_\theta)^{M+1}} \exp \left(\sum_{i=1}^N \frac{-1}{2\sigma_\epsilon^2} (y_i - \theta^\top \tilde{x}_i)^2 - \frac{\|\theta\|^2}{2\sigma_\theta^2} \right) \\ &= \arg \min_{\theta} \sum_{i=1}^N (y_i - \theta^\top \tilde{x}_i)^2 + \frac{\sigma_\epsilon^2}{\sigma_\theta^2} \|\theta\|^2. \end{aligned} \quad (2.79)$$

Observemos que esta expresión es equivalente al costo cuadrático regularizado de la ec. (2.14) con orden $p = 2$, es decir, la solución *máximo a posteriori* del modelo lineal y Gaussiano con prior Gaussiano es equivalente a la de mínimos cuadrados regularizados (con orden de regularización $p = 2$).

Si bien en la ec. (2.79) elegimos un prior Gaussiano, pudimos haber elegido un prior exponencial $p(\theta) \propto \exp(\gamma|\theta|)$, con lo que habríamos llegado a MCR con regularización $p = 1$ (o LASSO). Esto conecta claramente el uso de una distribución a priori dentro de la inferencia Bayesiana con el criterio general de regularización: El imponer un prior sobre θ es *promover*, mediante probabilidades relativas, algunas soluciones para θ ; mientras que, por el contrario, el uso de un regularizador *penaliza* algunas soluciones. Ajustando apropiadamente la función de regularización y la distribución a priori, podemos llegar a soluciones equivalentes en ambos casos, en particular, la elección de un prior uniforme equivale a un coeficiente de regularización ρ nulo. Sin embargo, debemos recordar que además de la estimación puntual MAP, el enfoque bayesiano entrega la distribución completa sobre el parámetro desconocido.

Un ejemplo de la conexión entre regularizadores y priors puede ser obtenido directamente del desarrollo anterior. La razón entre las varianzas del ruido y del prior en la ec. (2.79), dada por $\sigma_\epsilon^2/\sigma_\theta^2$, toma el mismo rol que el peso del regularizador ρ en la formulación de MCR en la ec. (2.14). Esto implica que un prior con varianza σ_θ^2 pequeña (cf. grande) es equivalente a un problema de MCR con un ρ grande (cf. pequeño), lo cual tiene sentido: θ puede ser “llevado a cero” mediante la asignación de un prior concentrado en cero o bien penalizando el magnitud de θ (MCR).

Desde ahora, podemos referirnos como MAP a las estimaciones puntuales en general pues, como acabamos de ver, esta es equivalente a MCR y al mismo tiempo contiene al criterio de máxima verosimilitud como caso particular (cuando el prior es uniforme). Además, para modelos generales (distintos al caso lineal y gaussiano) el MAP no podrá ser calculado de forma explícita imponiendo

$$\nabla_\theta \log p(\theta|\mathcal{D}) = 0, \quad (2.80)$$

sino que tendremos que considerar algoritmos de optimización. En particular consideraremos algoritmos basados en derivadas con iteraciones de la forma

$$\theta_{i+1} = \theta_i + \eta \nabla_\theta \log p(\theta_i|\mathcal{D}), \quad (2.81)$$

donde esperamos que la secuencia $\{\theta_i\}_i$, al avanzar en el *ascenso del gradiente de* $\log p(\theta_i|\mathcal{D})$, tienda al punto fijo de la ec. (2.81), es decir, $\nabla_\theta \log p(\theta|\mathcal{D}) = 0$ (ver método del gradiente en anexos).

Observación 2.2. En el resto del curso, veremos distintos modelos que requieren técnicas sofisticadas de optimización, las cuales permiten no solo *entrenar* dichos modelos (i.e., encontrar sus parámetros), sino también entender cómo funcionan; estos incluyen redes neuronales, máquinas de soporte vectorial y procesos gaussianos. Con lo visto hasta ahora podemos ver que el aprendizaje de máquinas, se sustenta en dos elementos fundamentales: el modelamiento probabilístico que permite definir *posibles* modelos, y el proceso de optimización que permite encontrar el *buen modelo*. Podemos entonces, a *grosso modo*, decir que “aprender es optimizar”.

Ejemplo: MAP para el modelo binomial

Siguiendo el ejemplo anterior de la distribución binomial, recordemos que al asumir un prior $p(\theta) = \text{Beta}(\theta; \alpha, \beta)$ para el parámetro θ , la posterior del mismo (dado un conjuntos de N experimentos con $\{n_i\}$ lanzamientos y $\{k_i\}$ aciertos) está dada por

$$\text{Beta}\left(\theta; \alpha + \sum_{i=1}^N k_i, \beta + \sum_{i=1}^N n_i - k_i\right). \quad (2.82)$$

Denotando los parámetros de la posterior mediante $\alpha_N = \alpha + \sum_{i=1}^N k_i$ y $\beta_N = \beta + \sum_{i=1}^N n_i - k_i$,

podemos encontrar el MAP mediante

$$\theta_{\text{MAP}} = \arg \max \theta^{\alpha_N - 1} (1 - \theta)^{\beta_N - 1} = \frac{\alpha_N - 1}{\alpha_N + \beta_N - 2},$$

pues el cálculo no es necesario ya que la moda de la distribución Beta es conocida. Además, la varianza posterior, está dada por

$$\mathbb{V}[\theta | \mathcal{D}] = \frac{\alpha_N \beta_N}{(\alpha_N + \alpha_N)^2 (\alpha_N + \beta_N + 1)}. \quad (2.83)$$

Es directo ver que, como función de las sumas de aciertos/fallos, cuando observamos muchos datos θ_{MAP} tiende a la razón entre aciertos y lanzamientos totales y $\mathbb{V}[\theta | \mathcal{D}]$ tiende a cero. Verificaremos esto numéricamente de la misma forma que el ejemplo anterior: consideremos 20 priors Beta con parámetros α y β aleatorios entre 1 y 10 y grafiquemos tanto θ_{MAP} como $\mathbb{V}[\theta | \mathcal{D}]$ en función de un número creciente de lanzamientos binomiales. La Figura 12 confirma nuestra intuición donde (con escala logarítmica para el eje x) podemos ver que con 10 lanzamientos, ya se ve una clara tendencia, mientras que con 100 lanzamientos la convergencia es indudable.

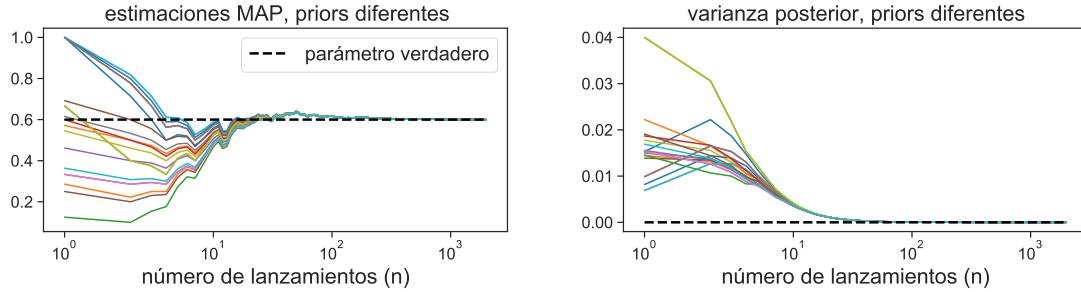


Fig. 12. Concentración de MAP y reducción de varianza posterior: modelo binomial.

2.4. Predicciones

Luego de haber discutido la estimación de parámetros, tanto desde el punto de vista puntual (MC, MCR, MV, MAP) como de estimación bayesiana, ahora veremos cómo estas estimaciones se utilizan para hacer predicciones.

Definamos en primer lugar el contexto para nuestra predicción. Además, introduciremos el concepto de *variable latente*, es decir, una variable dentro del modelo que no es observable, pero que nos interesa predecir. Esta definición es clave para estudiar la predicción: si bien la construcción del modelo para y tiene como objetivo modelar la generación de datos (θ) y sus observaciones (y), nuestro objetivo final es predecir la cantidad latente, no la observación futura; pues la observación contiene una perturbación asociada a su incertidumbre aleatoria. En el caso lineal, por ejemplo, denotamos la variable latente como $f = \theta^\top x$, donde el modelo está dado por

$$y = \theta^\top x + \epsilon = f + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2). \quad (2.84)$$

Denotaremos entonces mediante \hat{f}_* e \hat{y}_* las predicciones de la variable latente f y la observación y para una nueva entrada x_* , condicional a los datos observados \mathcal{D} . A continuación veremos que distintos supuestos sobre la estimación de θ resultan en distintas predicciones para estas cantidades, donde utilizaremos el modelo lineal/gaussiano de varianza conocida para ilustrar estas diferencias.

En primer lugar consideraremos las estimaciones puntuales, en cuyo caso realizamos un procedimiento conocido como *plug-in prediction*, en donde simplemente utilizamos el valor (puntual) la estimación del

parámetro dentro del modelo. En el ejemplo del modelo lineal, si hemos calculado el parámetro mediante máxima verosimilitud (denotado como θ_{MV}) entonces nuestro modelo lineal estimado es $y = \theta_{\text{MV}}^\top \tilde{x} + \epsilon$. Con lo que la predicción usual de la variable latente correspondiente a una entrada \tilde{x}_* es determinista (pues tanto θ_{MV} como x_* lo son) y dada por

$$\hat{f}_* = \theta_{\text{MV}}^\top \tilde{x}_*. \quad (2.85)$$

Por el contrario, si lo que quisiésemos estimar es efectivamente la variable aleatoria $y_*|\tilde{x}_*$ (la cual incluye el ruido de observación), esta predicción está dada por

$$\hat{y}_* \sim \mathcal{N}(\theta_{\text{MV}}^\top \tilde{x}_*, \sigma^2). \quad (2.86)$$

Gráficamente, es también usual representar esta predicción en términos de su esperanza y *barras de error*, las cuales para el caso gaussiano son explícitas para el 95 % en función de la desviación estándar del ruido ϵ . Es decir, con un 95 % de probabilidad, $\hat{y}_* \in [\theta_{\text{MV}}^\top \tilde{x}_* - 2\sigma, \theta_{\text{MV}}^\top \tilde{x}_* + 2\sigma]$.

A diferencia de las estimaciones puntuales, cuando el parámetro θ es estimado de forma bayesiana (disponemos de su distribución posterior) el modelo estimado en sí es una **distribución aleatoria** pues el modelo es una función determinista de un parámetro θ aleatorio. Podemos entonces identificar dos fuentes de incertidumbre en el modelo aprendido como discutimos al comienzo de esta sección: la epistemológica dada por la distribución posterior de θ , y la aleatoriedad dada por la distribución de ϵ . En nuestro ejemplo del modelo lineal y Gaussiano con varianza conocida el modelo aprendido está dado por

$$y = \theta^\top \tilde{x} + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2), \quad \theta \sim p(\theta|\mathcal{D}) = \mathcal{N}(\theta_N, \sigma^2 \Lambda_n^{-1}), \quad (2.87)$$

donde los parámetros de la posterior de θ están descritos en las ecs. (2.65)-(2.66). Podemos ver entonces que la relación que tenemos para y está *amarrada* a otras cantidades aleatorias (θ y ϵ), de las cuales nos debemos deshacer para producir la predicción. Esto es porque la introducción de dichas variables fue necesaria para aprender la relación entre x e y a través de nuestro modelo propuesto (o supuesto), y no tienen *necesariamente alguna relación con la realidad*. En la literatura clásica, estos parámetros son llamados *nuisance parameters* y su remoción, la cual es realizada a través del cálculo de una integral, es referido como *to integrate out*. Es difícil acordar una traducción al español de este concepto, el cual automáticamente sugiere que estamos “sacando” (*out*) estos parámetros mediante integración; a falta de un mejor término simplemente usaremos la expresión en inglés o su desafortunada traducción “desintegrar”. Por lo tanto, la distribución posterior de la variable latente está dada por la “desintegración” de θ con respecto a su propia distribución posterior, es decir,

$$\hat{f}_* \sim p(f_*|x_*, \mathcal{D}) = \int p(f_*, \theta|x_*, \mathcal{D}) d\theta = \int p(f_*|x_*, \mathcal{D}, \theta)p(\theta|\mathcal{D}, x_*) d\theta = \int p(f_*|x_*, \theta)p(\theta|\mathcal{D}) d\theta \quad (2.88)$$

donde, de izquierda a derecha:

- la primera igualdad pretende *hacer aparecer* $p(f|\theta)$ que es lo que realmente conocemos, donde $p(f|\mathcal{D})$ va a ser calculado mediante la desintegración de θ
- la segunda igualdad es simplemente la definición de distribución condicional
- la tercera igualdad elimina redundancias: dado θ y x_* , f_* no depende de los datos; de igual forma, la distribución posterior del parámetro depende solo de los datos pasados, no de un nuevo input ya que x_* no aporta información (no se conoce su salida).

En el caso lineal/gaussiano, el integrando de la expresión más a la derecha de la eq. (2.88) es la multiplicación de dos gaussianas. Si bien el cálculo de dicha integral es tedioso, esta tiene la forma de una

convolución entre dos gaussianas y sabemos⁹ que eso resulta en, nuevamente, una gaussiana. En el caso lineal, sin embargo, ni siquiera es necesario calcular dicha integral, pues podemos calcular la ley posterior de f_\star notando simplemente que $f = \theta^\top \tilde{x}_\star$ y que $\theta \sim \mathcal{N}(\theta_N, \sigma^2 \Lambda_n^{-1})$, lo cual da por linealidad:

$$\hat{f}_\star \sim p(f_\star | x_\star, \mathcal{D}) = \mathcal{N}(\theta_N^\top \tilde{x}_\star, \tilde{x}_\star^\top \sigma^2 \Lambda_n^{-1} \tilde{x}_\star). \quad (2.89)$$

Desde esta expresión es posible graficar el modelo mediante la determinación de las barras de error de los parámetros o bien generando una muestra para el parámetro desde su distribución posterior y luego graficar el modelo con dicha muestra — esto es equivalente a samplear desde la posterior del modelo.

Finalmente, para determinar la predicción de la observación ruidosa $y = f + \epsilon$, simplemente notemos que esta estará dada por la incorporación del estadístico del ruido a la expresión en eq. (2.89), en efecto,

$$\hat{y}_\star \sim \mathcal{N}(\theta_N^\top \tilde{x}_\star, \tilde{x}_\star^\top \sigma^2 \Lambda_n^{-1} \tilde{x}_\star + \sigma^2). \quad (2.90)$$

Observación 2.3. De la predicción bayesiana en forma de distribución de la ec. (2.89) podemos obtener (y es usualmente necesario) una predicción puntual dada por la media, la cual corresponde a

$$\mathbb{E}[f_\star | x_\star, \mathcal{D}] = \mathbb{E}[\theta | \mathcal{D}]^\top \tilde{x}_\star = \bar{\theta}^\top \tilde{x}_\star, \quad (2.91)$$

donde hemos denotado por $\bar{\theta}$ a la media posterior de θ . Consecuentemente, las estimaciones puntuales reportadas por el método bayesiano y de estimación puntual resultan en la misma predicción. Este fenómeno solo ocurre en situaciones particular, en este caso, debido a que el modelo es lineal y gaussiano.

Finalmente, la Fig. 13 muestra la implementación de las 4 formas de predecir usando el modelo lineal aplicado al dataset de grillos y temperatura usado inicialmente en la Fig. 6. Es posible ver como la predicción bayesiana da sutilmente distintas predicciones que en los extremos del modelo reportan más incertidumbre, mientras que la predicción de y considera tanto ruido que presenta una banda de incertidumbre *uniforme* a lo largo de las observaciones.

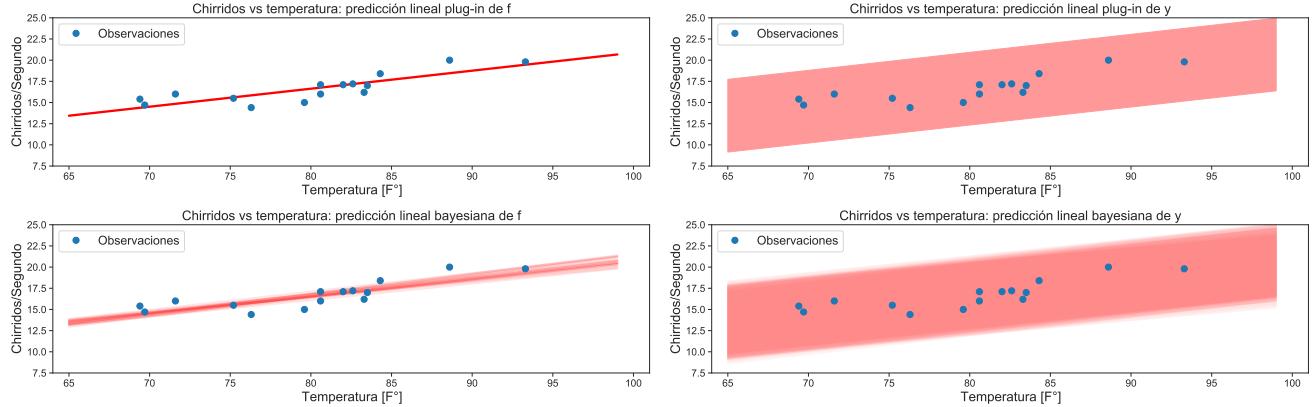


Fig. 13. Ejemplo de predicción usando el modelo lineal para la base de datos de grillos y temperatura. Desde la esquina superior izquierda: predicción plug-in de f , predicción plug-in de y , predicción bayesiana de f (20 muestras), predicción bayesiana de y (20 muestras).

2.5. Regresión no lineal

El modelo de regresión lineal visto en las secciones anteriores tiene diversas ventajas, en primer lugar su solución en el caso de mínimos cuadrados (o máxima verosimilitud en el caso de ruido gaussiano) puede ser encontrado de forma explícita y única. Además, sus parámetros permiten interpretar la relación entre

⁹Verificable fácilmente a través del teorema de la convolución.

cada una de las componentes de la variable de entrada y la variable de salida. Sin embargo, más allá de estas propiedades del modelo lineal, su alcance es muy limitado pues muchas veces necesitamos modelar fenómenos que no siguen una relación lineal.

El concepto de regresión lineal puede ser extendido a una contraparte no lineal en los casos particulares que conocemos a priori el tipo de relación entre las variables de entrada x y salida y (y esta no es lineal). Dicha extensión puede ser obtenida simplemente mediante la aplicación de una transformación (de parámetros fijos) a la variable independiente, e.g. $\phi(x)$, para luego construir un modelo lineal en la variable transformada $\phi = \phi(x)$ en lugar de en la variable original x .

Específicamente, consideraremos transformaciones a valores vectoriales de la variable independiente de la siguiente forma

$$\begin{aligned}\phi: \mathbb{R}^M &\rightarrow \mathbb{R}^D \\ x \mapsto \phi(x) &= [\phi_1(x), \dots, \phi_D(x)]^\top,\end{aligned}\tag{2.92}$$

donde $\phi_i : x \in \mathbb{R}^M \mapsto \phi_i(x) \in \mathbb{R}$ son funciones escalares $\forall i = 1, \dots, D$.

Con la introducción de la transformación $\phi(\cdot)$ es necesario hacer la distinción entre ambas variables de entrada. Nos referiremos entonces a x como datos/entradas crudos (*raw data/input*) y a la variable $\phi = \phi(x)$ como características (*features*). Recordemos que los modelos del capítulo anterior eran aplicados directamente sobre la entrada cruda x (o \tilde{x}), con lo que no existía distinción entre entradas crudas o características. Adicionalmente, la introducción del concepto de *característica* es motivado por la búsqueda de una representación de x que permite resolver el problema de regresión usando un modelo simple (y también interpretable) como el modelo lineal, a diferencia de la idea de diseñar un modelo muy complicado (no interpretable) que reciba directamente la entrada cruda x . Un ejemplo de esto es el caso en que x es una imagen: en este caso no es posible aplicar directamente un modelo lineal a la representación matricial de la imagen, sino que a una representación distinta de esta, o a sus *características*, es decir, sus bordes, formas, y otros patrones a identificar dentro de la imagen.

En la práctica, la función $\phi : x \mapsto \phi(x)$ es elegida en base al conocimiento *experto* que se tenga del problema de regresión a resolver; como su elección pretende extraer las características de interés y representarlas de una forma compatible con el modelo lineal, entonces, nos referiremos a la construcción *manual* de la función ϕ como *ingeniería de características*. Observemos entonces que la función ϕ puede tener (y en general tiene) parámetros, los cuales en esta primera instancia serán elegidos manualmente. Sin embargo, más adelante veremos modelos donde los parámetros de ϕ también pueden ser encontrados de forma automática (e.g., mediante máxima verosimilitud). En dicho caso, es posible interpretar que la ingeniería de características ya no es realizada de forma manual, sino que automatizada.

Características y modelos

El diseño de características es uno de los problemas fundamentales del AM. Esto porque podemos entender la construcción de modelos de aprendizaje supervisado como dos etapas: i) la identificación de las características relevantes y ii) cómo usar estas características para estimar el output (el modelo). En el modelo no lineal presentado aquí, es claro que la parte no lineal ϕ es la característica y la parte lineal es el modelo, sin embargo, en el caso general la línea entre ambas etapas es muy delgada. Por ejemplo, en una red neuronal de 100 capas, ¿qué corresponde a una característica y qué corresponde al modelo? Podríamos argumentar que en modelos más complicados como los de redes neuronales, existen varias interpretaciones de qué corresponde a

cada cosa, y con eso de paso interpretar qué está haciendo nuestro modelo. El hecho de que el diseño de características y el modelo se confundan es interesante también, pues quiere decir que las características se aprenden de igual forma que el resto del modelo; con lo que pasamos desde un diseño manual de características a una búsqueda automáticas de características.

2.5.1. Modelo lineal en los parámetros

Usando la nueva variable de características $\phi = \phi(x)$ como entrada a un modelo lineal, podemos definir el siguiente modelo de regresión lineal y gaussiano:

$$y = \theta^\top \phi(x) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2), \quad (2.93)$$

donde asumiremos que contamos con un conjunto de observaciones de la forma

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N, \quad (x_i, y_i) \in \mathbb{R}^M \times \mathbb{R}, \forall i = 1, \dots, N. \quad (2.94)$$

De la misma forma que vimos en el capítulo anterior, este modelo puede ser entrenado mediante la optimización de costo cuadrático:

$$J = \sum_{i=1}^N (y_i - \theta^\top \phi(x_i))^2 \quad (2.95)$$

lo cual es equivalente a máxima verosimilitud, pues el ruido de observación ϵ es gaussiano.

Además, se puede compactar el funcional utilizando la matriz de diseño al igual que en regresión lineal:

$$\Phi = \begin{bmatrix} \phi(x_1)^\top \\ \vdots \\ \phi(x_N)^\top \end{bmatrix} = \begin{bmatrix} \phi_1(x_1) & \dots & \phi_D(x_1) \\ \vdots & \ddots & \vdots \\ \phi_1(x_N) & \dots & \phi_D(x_N) \end{bmatrix} \quad Y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}, \quad (2.96)$$

con lo que el modelo y el funcional de costo puede ser expresados respectivamente mediante

$$Y = \Phi\theta + \epsilon, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \sigma_\epsilon^2 \mathbf{I}) \quad (2.97)$$

$$J = (Y - \Phi\theta)^\top (Y - \Phi\theta) = \|Y - \Phi\theta\|_2^2 \quad (2.98)$$

De forma análoga al modelo lineal estándar (el cual toma directamente la variable cruda x como input), la minimización del funcional J puede ser encontrado mediante la condición de primer orden $\nabla_\theta J = 0$ ya que el costo es estrictamente convexo. De este modo:

$$\theta^\star = (\Phi^\top \Phi)^{-1} \Phi^\top Y. \quad (2.99)$$

Las siguientes observaciones conectan el modelo no lineal de la ec. (2.93) con el modelo lineal estudiado en las secciones anteriores.

Observación 2.4. De la ec. (2.99) vemos que si elegimos $\phi(\cdot) = \text{id}(\cdot)$ recuperamos la expresión para mínimos cuadrados ordinarios. Esto permite interpretar el modelo de regresión no lineal con variables $\{(x_i, y_i)\}_{i=1}^N$ de \mathbb{R}^N a \mathbb{R} como una regresión lineal con variables $\{(\Phi(x_i), y_i)\}_{i=1}^N$ de \mathbb{R}^D a \mathbb{R} . Además, la razón por la que la forma de la solución es la misma que el caso lineal es porque si bien el modelo presentado en la ec. (2.93) es no lineal en la entrada x , este sí es *lineal en los parámetros* θ .

Observación 2.5. Notemos que la re-parametrización del modelo afín introducida en el capítulo anterior es un caso particular de la transformación Φ . Esto es directo de la siguiente construcción:

$$\Phi = [x, 1]^\top \quad (2.100)$$

$$y = a^\top x + b = \theta^\top \Phi(x) + \epsilon. \quad (2.101)$$

Observación 2.6. Finalmente, es posible considerar una extensión regularizada al modelo no lineal presentado en la ec. (2.93) mediante la consideración de un costo regularizado cuadrático (i.e., *ridge regression*) dado por

$$J_\rho = \|Y - \Phi\theta\|_2^2 + \rho \|\theta\|^2, \quad \rho \in \mathbb{R}^+. \quad (2.102)$$

En cuyo caso, la solución está dada por

$$\theta = (\Phi^\top \Phi + \rho \mathbb{I})^{-1} \Phi^\top Y. \quad (2.103)$$

Selección de características

Un buen conjunto de características no solo ayuda a una buena representación (y consecuentemente predicción) de nuestros datos, sino que extrae literalmente las *características* que son relevantes de x para determinar y . Por ejemplo, si necesitamos determinar la condición clínica de un paciente, para el cual tenemos una vasta colección de mediciones como peso, edad, género, dirección, número de teléfono, ocupación, color de ojos, salario, etc, muy probablemente muchas estas características no van a ser útil (por ejemplo, ocupación), con lo que el diseño de las funciones ϕ debe tomar en cuenta esto. Por el contrario, si necesitamos evaluar el mismo conjunto de individuos para un crédito de consumo, las características a considerar variarán y ahora probablemente el salario sí juegue un rol importante. Este concepto permite entender la estrecha relación entre el diseño de características y la *selección de características* y consecuentemente la *selección de modelos*, lo cual veremos en los siguientes capítulos. En particular, recordemos que usando el regulador LASSO, podemos llevar algunas de nuestras coordenadas a cero, realizando una selección automática de características.

2.5.2. Ejemplo de transformaciones

La selección de las características es fundamental para una representación apropiada de nuestros datos y consecuentemente para el desempeño del modelo de regresión no lineal. En efecto, si la familia de funciones ϕ_1, \dots, ϕ_D no captura la representación de las características de forma compatible con el modelo lineal, el esfuerzo por ajustar la parte lineal del modelo será infructuoso. En general, la elección de la base de funciones debe ser diseñada y evaluada caso a caso.

A continuación vemos algunos ejemplos de características que son usualmente usadas en la práctica por su generalidad (habilidad para replicar varios comportamientos) y/o simpleza.

Función Polinomial: $\phi = \{\phi_i\}_{i=0}^D$, donde $\phi_i(x) = x^i$, de tal forma que

$$\Phi = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^D \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \dots & x_N^D \end{bmatrix}. \quad (2.104)$$

La función polinomial es ampliamente usada ya que de acuerdo al teorema de Stone-Weierstrass, los polinomios son densos en $C([a, b])$ (funciones continuas sobre compactos), por lo que es posible aproximar cualquier función continua mediante un polinomio, donde la precisión puede ser tan pequeña como se necesite. Sin embargo, una desventaja de esta base es que tanto su entrenamiento como predicciones pueden ser inestables: para obtener una buena aproximación polinomial, generalmente se requiere un grado D alto, por lo que los valores de $\phi(x)$ crecen, obviamente, de forma *polinomial*.

Por otra parte, la interpolación polinomial sufre del fenómeno de Runge, por lo que al utilizar un grado elevado, es posible que el error de predicción en los bordes crezca indefinidamente.

Función Sinusoidal: $\phi = \{\phi_i\}_{i=0}^D$, donde $\phi_i(x) = \cos(i\frac{2\pi}{2T}(x - b_i))$. La variable i actúa como una frecuencia normalizada con respecto al período de oscilación T y b_i es un “offset” o fase. Esta transformación está dada por

$$\Phi = \begin{bmatrix} 1 & \cos(1\frac{2\pi}{2T}(x_1 - b_1)) & \dots & \cos(D\frac{2\pi}{2T}(x_1 - b_D)) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \cos(1\frac{2\pi}{2T}(x_N - b_1)) & \dots & \cos(D\frac{2\pi}{2T}(x_N - b_D)) \end{bmatrix}. \quad (2.105)$$

Hemos asumido que la fase para cada función ϕ_i , la frecuencia $i\frac{2\pi}{2T}$ y fase b_i son fijas. Una forma de evitar definir una fase, es considerar dos transformaciones por cada ϕ_i de la forma

$$\phi'_i(x) = \left[\sin\left(i\frac{2\pi}{2T}x\right), \cos\left(i\frac{2\pi}{2T}x\right) \right], \quad (2.106)$$

donde no es necesario definir la fase sino que esta es absorbida por el parámetro θ y aprendida.

Al igual que los polinomios, la base de senos y cosenos es también *universal* (más aún, forman una base de Hilbert de L^2 en el círculo). Sin embargo, una desventaja de la base senoidal es que solo puede replicar funciones periódicas, con un período máximo en este caso de T . En la práctica, veremos que un modelo sinusoidal se repetirá cada período, pudiendo llevar a conclusiones erróneas. Además, si efectivamente nuestros datos representan un comportamiento periódico pero el ruido de observación resulta en que cada período o *forma de onda* sean ligeramente distintos, puede ser difícil encontrar el período T . La identificación de este período es un desafío en sí mismo, conocido como *detección de periodicidades o frecuencias fundamentales*, usual en astronomía y procesamiento de audio.

Funciones constantes por partes: La siguiente característica está compuesta por “escalones”, los cuales valen “1” dentro de un conjunto específico y “0” en cualquier otro lugar. Sean $c_1, c_2, \dots, c_D \in \mathbb{R}$ una colección de valores crecientes, definimos la indicatriz I_A y las funciones de pertenencia C_i :

$$C_0(x) = I_{(-\infty, c_1)}(x) \quad (2.107)$$

$$C_i(x) = I_{[c_i, c_{i+1})}(x), \quad \forall i = 1, \dots, D-1 \quad (2.108)$$

$$C_D(x) = I_{[c_D, +\infty)}(x) \quad (2.109)$$

$$I_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} \quad (2.110)$$

De esta forma la base de funciones $\Phi = \{\phi_i\}_{i=0}^D$ queda definida por

$$\phi_i(x) = C_i(x), \quad \forall i = 0, \dots, D \quad (2.111)$$

$$\Phi(X) = \begin{bmatrix} 1 & C_0(x_1) & \dots & C_D(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & C_0(x_N) & \dots & C_D(x_N) \end{bmatrix}. \quad (2.112)$$

Las funciones contantes por partes son útiles cuando trabajamos con experimentos que involucran una discretización, por ejemplo, al modelar la altura/concentración/conductividad/temperatura en una superficie y no es posible parametrizar esta cantidad como función del (en este caso) espacio. Entonces, simplemente dividimos la superficie en una grilla y asignamos un valor distinto a cada celda de esta grilla. Es un método útil pero la cantidad de parámetros (dimensión de θ) crece con el número de celdas de la grilla, lo cual es prohibitivo para casos donde el rango de valores de x y/o su dimensión es medianamente alta.

Ejemplo: Predicción de pasajeros de una aerolínea)

Consideremos el problema de predecir la cantidad de pasajeros en una aerolínea, considerando distintas combinaciones de características. De forma incremental, tomaremos en consideración características polinomiales, senoidales, y senoidales con amplitud creciente. Es decir, denotando x en tiempo y y la cantidad de pasajero, consideraremos el siguiente modelo

$$y = \underbrace{\sum_{i=0}^3 \theta_i x^i}_{\text{parte polinomial}} + \underbrace{\sum_{i=1}^2 \alpha_i \exp(-\tau_i x^2) \cos(\omega_i(x - \psi_i))}_{\text{parte oscilatoria}}. \quad (2.113)$$

La motivación de este modelo es representar la tendencia de los datos mediante la componente polinomial y la oscilación anual mediante las componentes oscillatorias.

Además, hemos considerado 12 años de datos con frecuencia mensual (144 datos), de los cuales solo 9 años (108 datos) han sido usado para encontrar los parámetros del modelo y los 3 años restantes (36 datos) para validar nuestras predicciones. La Figura 14 muestra los datos y las estimaciones para 3 variantes del modelo mencionado: parte polinomial, parte polinomial y oscillatoria de amplitud fija, y finalmente, parte polinomial y oscillatoria de amplitud variable. En cada gráfico se muestran distintas opciones de cada modelo en gris con la mejor trayectoria en negro; se puede ver además en cada gráfico cómo disminuye el error de aproximación a medida que consideramos más características.

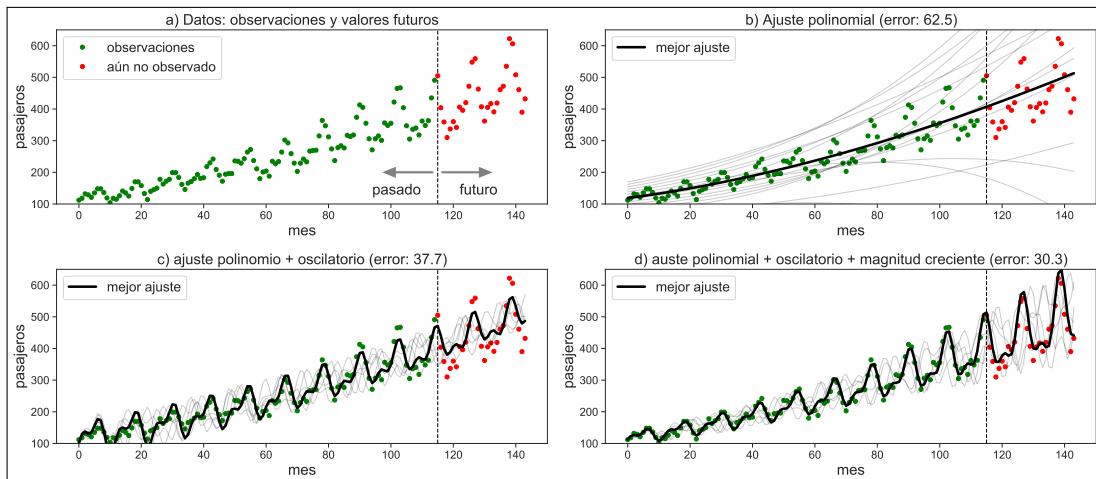


Fig. 14. Regresión no-lineal de cantidad de pasajeros en el tiempo. Arriba-izq: datos de entrenamiento, arriba-der: modelo con componente polinomial, abajo-izq: modelo con componentes polinomial y oscilatoria (amplitud fija), abajo-der: modelo con componentes polinomial y oscilatoria (amplitud creciente).

3. Clasificación

El problema de clasificación dice relación con la identificación del conjunto, categoría o *clase* a la cual pertenece un elemento en base a sus *características* o *features*. En el contexto del aprendizaje supervisado, el problema de clasificación puede ser visto como un caso particular del problema de regresión, donde el espacio en el que vive la variable y (salida o variable dependiente) es *categórico* y usualmente denotado por $\{0, 1\}$, para el caso binario, o bien $\{1, 2, \dots, K\}$ para el caso de clasificación multiclas.

3.1. Vecinos más cercanos (KNN)

Dado un conjunto de datos

$$\mathcal{D} = \{(x_i, c_i)\}_{i=1}^N \subset \mathbb{R}^M \times \{\mathcal{C}_i\}_{i=1}^K \quad (3.1)$$

donde, al igual que en el problema de regresión, x es la variable independiente y c es la variable dependiente o *clase*, una forma natural de clasificar una nueva entrada x_* será asignarle la clase correspondiente a los datos observados más similares a x_* .

Así, el modelo KNN funciona bajo esta lógica: que entradas similares tienen salidas similares. Bajo este modelo, la formulación del problema de clasificación es la siguiente:

Sea $N_k(x_*) \subset \mathcal{D}$ conjunto de los k vecinos más cercanos a x_* , es decir:

$$|N_k(x_*)| = k \quad \text{y} \quad d(x_*, x) \geq \max_{(x', y') \in N_k(x_*)} d(x_*, x'), \quad \forall (x, y) \in \mathcal{D} \setminus N_k(x_*) \quad (3.2)$$

Donde $d : \mathbb{R}^M \times \mathbb{R}^M \rightarrow \mathbb{R}_+$ corresponde a la métrica euclidiana.

Entonces, el clasificador KNN retorna la etiqueta más común:

$$\mathcal{C}(x) = \text{moda}(\{y' : (x', y') \in N_k(x_*)\}) \quad (3.3)$$

Si bien KNN pareciera un algoritmo heurístico, tiene fuertes propiedades de convergencia. Para enumerar esto, es necesaria la siguiente definición:

Definición 3.1 (clasificador óptimo de Bayes). dado $p(y|x)$, el clasificador óptimo de Bayes asigna la etiqueta más probable:

$$\bar{\mathcal{C}}(x) = \arg \max_{y \in \{\mathcal{C}_i\}_{i=1}^K} p(y|x) \quad (3.4)$$

En el clasificador anterior, la probabilidad de error corresponde a $e = 1 - p(\bar{\mathcal{C}}|x)$ y es una cota inferior para el error ya que ningún otro clasificador puede obtener un error menor. Dado que no se conoce $p(y|x)$, esta cota corresponde a una cota asintótica.

Por otra parte, para el clasificador KNN se tiene el siguiente resultado:

Teorema 3.1 (desigualdad de Cover-Hart). sea R^* el error (riesgo) del clasificador óptimo de Bayes y R el error asintótico de KNN, entonces R está acotado, más aún:

$$R^* \leq R \leq 2R^*(1 - R^*) \quad (3.5)$$

Sin embargo, aunque asintóticamente el clasificador funciona bien, la convergencia a dicha tasa de error es lenta y se ve afectada por la dimensión de la entrada (ver maldición de la dimensionalidad). Es por esto que es necesario formular otros clasificadores bajo otra perspectiva.

3.2. Clasificación lineal

Consideraremos en primera instancia el caso *binario*, es decir, solo dos clases ($K = 2$), ilustrado en la Fig. 15. Proponemos un modelo lineal para relacionar la variable independiente con su clase, es decir,

$$y(x) = a^\top x + b, \quad (3.6)$$

donde la asignación de la clase es de la siguiente forma: x será asignado a \mathcal{C}_1 si $y(x) \geq 0$ y será asignado \mathcal{C}_2 en caso contrario.

Nos referiremos al subconjunto que partitiona \mathbb{R}^M en clase \mathcal{C}_1 y clase \mathcal{C}_2 como *superficie/hiperplano/región de decisión*, la cual está definida por $y(x) = 0$. Entonces, como $x \in \mathbb{R}^M$, para la consideración del modelo lineal esta superficie de decisión corresponde a un hiperplano de dimensión $M - 1$.

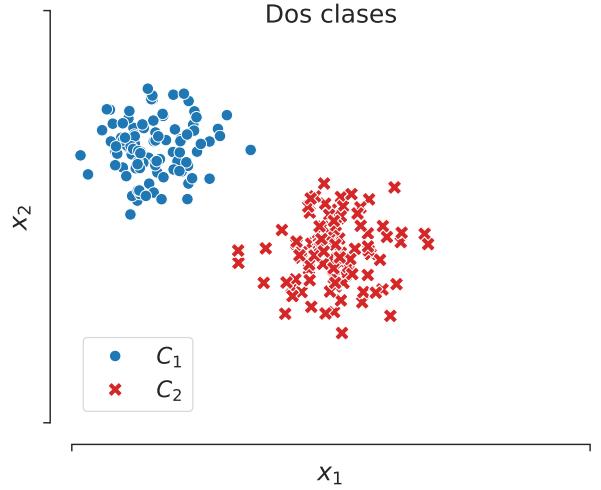


Fig. 15. Ejemplo del problema de clasificación binaria, donde la clase \mathcal{C}_1 está presentada en azul y la clase \mathcal{C}_2 en rojo.

Para resolver el problema de clasificación, debemos encontrar los parámetros a y b en la ec. (3.6). Para este fin, veamos que si x_1, x_2 son dos puntos en la superficie de decisión, entonces tenemos la siguiente relación para el parámetro a :

$$\begin{aligned} 0 &= y(x_1) - y(x_2) \\ &= a^\top x_1 + b - a^\top x_2 - b \\ &= a^\top (x_1 - x_2). \end{aligned} \quad (3.7)$$

Es decir, a es ortogonal a cualquier vector que esté contenido dentro de la región de decisión (ya que $x_1 - x_2$ corresponde a un vector director del hiperplano). Intuitivamente, podemos decir que el vector $a \in \mathbb{R}^M$ controla la pendiente (o inclinación) del hiperplano de decisión. Además, observemos que para cualquier x en la región de decisión, i.e., $y(x) = 0$, su proyección en el vector a está dada por

$$\|\text{proy}_a(x)\| = \|x\| \cos(\theta) = \|x\| \frac{a^\top x}{\|a\| \cdot \|x\|} = -\frac{b}{\|a\|} \quad (3.8)$$

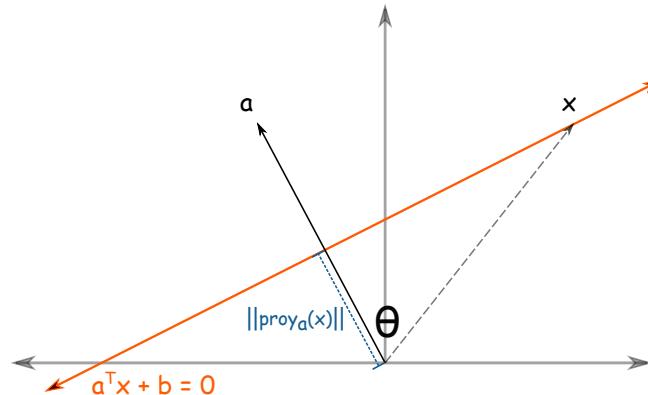


Fig. 16. Proyección de un punto sobre la región de decisión.

Donde se usó el hecho de que $\cos(\angle(x, y)) = \frac{\langle x, y \rangle}{\|x\|\|y\|}$.

Por lo tanto, se concluye que el parámetro b controla el desplazamiento (o ubicación) de la región de decisión, pues el lado izquierdo de la ecuación representa la distancia entre la región de decisión y el origen: $\|\text{proy}_a(x)\| = d(\{x : a^\top x + b = 0\}, 0)$.

Es posible también interpretar $y(x)$ como una distancia con signo entre un $x \in \mathbb{R}^M$ cualquiera y la superficie de decisión. Para ver esto, consideremos $x \in \mathbb{R}^M$ y descompongámoslo dos componentes: la primera denotada por x_\perp , la cual es la proyección ortogonal de x en el hiperplano de decisión, y la segunda que es perpendicular al hiperplano (y consecuentemente paralela al vector a) denotada por $r \frac{a}{\|a\|}$, donde r denota la distancia entre x y el hiperplano de decisión. Expresamos entonces

$$x = x_\perp + r \frac{a}{\|a\|}, \quad (3.9)$$

y observamos que

$$y(x) = a^\top x + b = a^\top \left(x_\perp + r \frac{a}{\|a\|} \right) + b = \underbrace{a^\top x_\perp + b}_{=0} + r \frac{a^\top a}{\|a\|} = r \|a\|. \quad (3.10)$$

Consecuentemente, vemos que $\forall x \in \mathbb{R}^M$, $r = \frac{y(x)}{\|a\|}$, es decir, $y(x)$ representa una medida con signo de la distancia entre el punto x y la región de decisión (ya que r lo es).

El caso de múltiples clases ($K > 2$) puede ser enfrentado mediante una extensión del caso binario. Una forma puede ser construir $K - 1$ clasificadores binarios, en donde cada uno resuelve el problema de discriminar los elementos de la clase \mathcal{C}_k del resto; esta técnica se llama *one-versus-rest* (OvR). Otra forma es construir $K(K - 1)/2$ clasificadores binarios, donde cada uno discrimina entre cada posible par de clases; esta técnica se llama *one-versus-one* (OvO). El problema con estas dos alternativas es que dejan regiones indefinidas en el espacio, pues solo consideran pares de clases que al ser agregados pueden ser incoherentes.

Una alternativa más robusta para resolver el problema de clasificación multiclasa es construir un clasificador para K clases que contiene K funciones lineales de la forma

$$y_k(x) = a_k^\top x + b_k, \quad k = 1, \dots, K. \quad (3.11)$$

Donde x es asignado a la clase \mathcal{C}_k si y solo si $y_k(x) > y_j(x), \forall j \neq k$, es decir:

$$\mathcal{C}(x) = \arg \max_k y_k(x). \quad (3.12)$$

Además, las K componentes de la partición del espacio generada por este clasificador son convexas ya que es intersección finita de semiespacios (poliedro). Además, ningún punto del espacio queda sin clasificar ya que la clase queda determinada por el máximo de un conjunto finito. Por otra parte, el conjunto de puntos con más de una clase asignada tiene medida cero ya que las fronteras de decisión son subconjuntos de codimensión 1 (hiperplanos).

Por último, tomando $K = 2$, se obtiene el clasificador binario.

3.2.1. Ajuste mediante mínimos cuadrados

Ya hemos planteado el modelo y analizado el rol y significado de cada uno de sus parámetros; ahora queda por estudiar cómo determinar dichos parámetros a y b , dado un conjunto de datos \mathcal{D} . Para esto consideraremos en primer lugar el enfoque de mínimos cuadrados, el cual es la medida de discrepancia

por excelencia y a primera vista parece una respuesta natural a este problema. Primero introduciremos un poco de notación para plantear el problema de forma clara.

Consideremos el punto $x \in \mathbb{R}^M$ con clase $c \in \{\mathcal{C}_k\}_{k=1}^K$. Usaremos la *codificación* $t \in \{0, 1\}^K$ para representar la pertenencia de x a su respectiva clase. Es decir,

$$c = \mathcal{C}_j \Leftrightarrow [t]_j = 1 \wedge [t]_i = 0, \quad i \neq j. \quad (3.13)$$

Este tipo de codificación es conocida como *one-hot encoding*.

Observación 3.1. Usamos esta codificación por dos razones. Primero, para poder dar valores numéricos a la clase y que podamos hablar de distancias entre ellos, pues si nuestras clases son “peras y manzanas” necesitamos poder determinar cuán cerca a cada una de ellas está nuestra estimación. En segundo lugar, como los valores asignados al vector t siempre serán puros 0’s y un solo 1, entonces todos los posibles valores de t están a la misma distancia unos de otros. De esta forma, si dos elementos tienen distintas clases, la distancia entre estas, cualesquiera sean, será 0 (clases iguales), o bien $\sqrt{2}$ (clases distintas). Esto ayuda a no introducir sesgos en la representación de la clase que puedan alterar el aprendizaje del modelo.

Asumiendo entonces un modelo lineal para cada clase \mathcal{C}_k , se tiene que

$$y_k(x) = a_k^\top x + b_k = \tilde{\theta}_k^\top \tilde{x}, \quad \text{donde } \tilde{x} = \begin{pmatrix} x \\ 1 \end{pmatrix} \in \mathbb{R}^{M+1}, \quad \tilde{\theta}_k = \begin{pmatrix} a_k \\ b_k \end{pmatrix} \in \mathbb{R}^{M+1} \quad (3.14)$$

Lo anterior se puede unir en un único sistema matricial:

$$\tilde{\Theta} = (\theta_1, \dots, \theta_K) \in \mathbb{R}^{(M+1) \times K} \implies y(x) = \begin{pmatrix} y_1(x) \\ \vdots \\ y_K(x) \end{pmatrix} = \tilde{\Theta}^\top \tilde{x}. \quad (3.15)$$

Donde ahora y corresponde al vector de *asignaciones de clases*. De esta forma, un punto x será asignado a la clase que tenga mayor $y_k = \tilde{\theta}_k^\top \tilde{x}$. Es decir, la clase de x es modelada como el $\arg \max_k y_k(x)$.

Con la notación establecida, ahora podemos enfocarnos en el entrenamiento del modelo. Para esto consideremos un conjunto de entrenamiento $\{(x_n, t_n)\}_{n=1}^N$. El enfoque de entrenamiento será el correspondiente a mínimos cuadrados asociado al error de asignación:

$$J = \sum_{i=1}^N \|t_i - \tilde{\Theta}^\top \tilde{x}_i\|_2^2 \quad (3.16)$$

Por otra parte, definiendo las siguientes matrices:

$$T = \begin{pmatrix} t_1^\top \\ \vdots \\ t_N^\top \end{pmatrix} \in \mathbb{R}^{N \times K}, \quad \tilde{X} = \begin{pmatrix} \tilde{x}_1^\top \\ \vdots \\ \tilde{x}_N^\top \end{pmatrix} \in \mathbb{R}^{N \times (M+1)} \quad (3.17)$$

se tiene el siguiente resultado:

Proposición 3.1.1. *Bajo la notación anterior, $J = \text{Tr}((\tilde{X}\tilde{\Theta} - T)^\top(\tilde{X}\tilde{\Theta} - T))$ y su mínimo es alcanzado en:*

$$\tilde{\Theta} = (\tilde{X}^\top \tilde{X})^{-1} \tilde{X}^\top T \quad (3.18)$$

donde Tr corresponde al operador traza: $A \in \mathbb{R}^{n \times n} \mapsto \text{Tr}(A) := \sum_{i=1}^n a_{ii}$.

Demostración.

$$J = \sum_{i=1}^N \|t_i - \tilde{\Theta}^\top \tilde{x}_i\|_2^2 = \sum_{i=1}^N \left\| (T - \tilde{X}\tilde{\Theta})_{i\cdot} \right\|_2^2 = \sum_{i=1}^N \sum_{j=1}^K (T - \tilde{X}\tilde{\Theta})_{ij} (T - \tilde{X}\tilde{\Theta})_{ij} \quad (3.19)$$

$$= \sum_{i=1}^N \sum_{j=1}^K (T - \tilde{X}\tilde{\Theta})_{ji}^\top (T - \tilde{X}\tilde{\Theta})_{ij} = \sum_{j=1}^K \left[(T - \tilde{X}\tilde{\Theta})^\top (T - \tilde{X}\tilde{\Theta}) \right]_{jj} \quad (3.20)$$

$$= \text{Tr} \left((\tilde{X}\tilde{\Theta} - T)^\top (\tilde{X}\tilde{\Theta} - T) \right). \quad (3.21)$$

Por otra parte:

$$\frac{\partial J}{\partial \tilde{\Theta}} = 2(\tilde{X}\tilde{\Theta} - T)^\top \tilde{X} = 0 \iff \tilde{\Theta}^\top \tilde{X}^\top \tilde{X} - T^\top \tilde{X} = 0 \iff \tilde{\Theta}^\top = T^\top \tilde{X} (\tilde{X}^\top \tilde{X})^{-1} \iff \tilde{\Theta} = (\tilde{X}^\top \tilde{X})^{-1} \tilde{X}^\top T \quad (3.22)$$

Y dado que J es estrictamente convexo, su mínimo se alcanza en su único punto crítico. ■

De acuerdo al lema anterior, la predicción de la clase para un nuevo input x está dada por $y(x) = T^\top \tilde{X} (\tilde{X}^\top \tilde{X})^{-1} \tilde{x}$.

Observación 3.2. Para una matriz no necesariamente cuadrada A se define la norma Frobenius de A como $\|A\|_F := \sqrt{\text{Tr}(A^\top A)}$. De este modo, el funcional J puede ser escrito como $J = \|\tilde{X}\tilde{\Theta} - T\|_F^2$, recuperándose la estructura de los funcionales utilizados hasta el momento: norma l_2 del error entre el regresor y la salida real.

Observación 3.3. Hay dos problemáticas conceptuales con este enfoque. En primer lugar, el uso de mínimos cuadrados es muy sensible a la presencia de puntos aislados (*outliers*). Dado el crecimiento cuadrático de la penalización, los puntos lejanos al promedio de los datos tienen una influencia mucho mayor. Esto ciertamente afecta considerablemente los resultados y no es coherente con el problema de clasificación, donde la estimación es correcta o incorrecta, pero no “más” o “menos” correcta. Este efecto sobre la solución es ilustrado en la Figura 17, en donde la presencia de puntos alejados de clase C_2 afecta el resultado (derecha), incluso donde el resultado original (izquierda) estaba correcto. En segundo lugar, las predicciones de clase $y(x)$ no tienen la forma de las etiquetas originales t , con lo cual es necesario una intervención “manual” en donde, dado un $y(x)$ debemos definir el $t(x)$ apropiado.

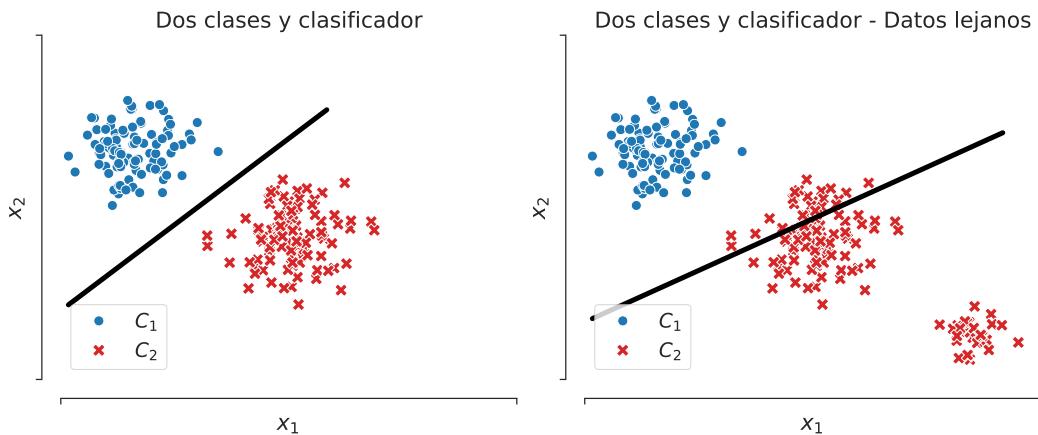


Fig. 17. Ejemplo ilustrativo sobre cómo los puntos lejanos de una clase pueden afectar incorrectamente los resultados.

3.2.2. El perceptrón

Las nociones básicas que hemos visto hasta ahora para lidiar con el problema de clasificación tienen dos problemas conceptuales. El primero es la falta de una métrica correcta para evaluar la bondad de nuestro modelo, esto es porque el criterios de mínimos cuadrados no es apropiado en la evaluación de una asignación de clases, donde no hay concepto de “más cerca”, sino que solo correcto/incorrecto. Además, todos los enfoques considerados en esta sección hasta este punto no tienen una *función de verosimilitud* apropiada que conecte el modelo (variables latentes) con la clase (observación) de forma coherente. Por el contrario, hasta ahora hemos considerado que por un lado tenemos el modelo lineal para clasificación y por otro lado está *nuestra decisión* de la clase en base a la salida del modelo lineal, por ejemplo si $y(x) > b$ decimos que x pertenece a la clase \mathcal{C}_1 .

La incorporación de esta función que conecta el modelo lineal con la clase, resulta en un *modelo lineal generalizado*, es decir, una modelo lineal conectado a una función no-lineal que llamaremos *función de enlace*. Sin embargo, el desafío más importante en esta construcción es que el modelo resultante ya no es lineal, ni en la entrada ni en los parámetros, pues una verosimilitud (función de enlace) lineal nunca nos llevará de un espacio de inputs (hemos asumido \mathbb{R}^M) al espacio de categorías $\{\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k\}$, es decir, necesitamos una no-linealidad “después” de la parte lineal, no antes como en el caso de los modelos lineales en los parámetros vistos en la sección de regresión no lineal.

Una forma de resolver estas problemáticas es mediante el uso del *Perceptrón* (Rosenblatt, 1958), un modelo de clasificación binario que tuvo mucha importancia en el área de reconocimiento de patrones. El Perceptrón consiste en una función no lineal fija usada para transformar x en un vector de características¹⁰ $\phi(x) \in \mathbb{R}^D$, que luego es usado para generar un modelo lineal *generalizado* con función de enlace no lineal $f(\cdot)$ de la siguiente forma:

$$y(x) = f(\theta^\top \phi(x)) \quad (3.23)$$

$$f(u) = \begin{cases} +1, & u \geq 0 \\ -1, & u < 0 \end{cases} \quad (3.24)$$

Opciones típicas para el vector $\phi(x)$ es concatenar la entrada con un intercepto para la recta, es decir $\phi(x) = [x^\top, 1]$ y $\theta \mapsto [\theta, b]^\top$ como vimos al comienzo del curso, o bien características no lineales como las vistas en la sección de regresión no lineal. El Perceptrón entonces asigna x a la clase \mathcal{C}_1 si $y(x) = +1$ y asignará x a la clase \mathcal{C}_2 cuando $y(x) = -1$. Notemos que para el caso que ϕ es lineal, este es el mismo clasificador presentado en la Sección 3.2, pero en este caso el criterio para asignar la clase es **parte del modelo**.

Una condición para determinar el parámetro $\theta \in \mathbb{R}^D$ en base a un conjunto de datos $\mathcal{D} = \{(x_i, t_i)\}_{i=1}^N$, es que para $x \in \mathcal{C}_1$ ($t = 1$), se cumpla que $\theta^\top \phi(x) > 0$, y para $x \in \mathcal{C}_2$ ($t = -1$) se cumpla $\theta^\top \phi(x) < 0$. Usando el hecho que las etiquetas están representadas por la codificación $t \in \{1, -1\}$, ambas condiciones pueden ser cubiertas por la expresión:

$$\theta^\top \phi(x_n) t_n > 0, \quad \forall (x_n, t_n) \in \mathcal{D}. \quad (3.25)$$

Podemos entonces satisfacer esta restricción mediante el “criterio del perceptrón”, el cual se basa en examinar los elementos de \mathcal{D} que fueron clasificados incorrectamente. Este criterio asocia a los puntos clasificados correctamente error 0 y a los puntos mal clasificados error $-\theta^\top \phi(x)t > 0$. De esta forma, si denotamos \mathcal{M} el conjunto de puntos mal clasificados, se debe minimizar la siguiente función objetivo:

¹⁰En este caso consideramos no linealidad antes y después de la parte lineal, sin embargo, considerar la entrada como x o como $\phi(x)$ es equivalente en base a lo visto en los modelos lineales en los parámetros.

$$J_P(\theta, x) = \mathbb{E} \left(-\theta^\top \phi(x) t(x) \mathbf{1}_{\theta^\top \phi(x) t(x) \leq 0} \right) \approx - \sum_{(x_i, t_i) \in \mathcal{D}} \theta^\top \phi(x_i) t_i \mathbf{1}_{\theta^\top \phi(x_i) t_i \leq 0} = - \sum_{(x_i, t_i) \in \mathcal{M}} \theta^\top \phi(x_i) t_i \quad (3.26)$$

Para la minimización de dicho funcional, se utilizará un método no determinístico que funciona mejor que el método del gradiente clásico (ver anexos) para este tipo de funciones.

Método del gradiente estocástico

En aprendizaje de máquinas por lo general se busca un parámetro óptimo que minimice el error de ajuste de acuerdo a una función de pérdida J . Dicho problema puede ser escrito de la forma:

$$\hat{\theta} = \arg \min_{\theta} \sum_{i=1}^n J(y_i, \hat{y}_\theta(x_i)) = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n J(y_i, \hat{y}_\theta(x_i)) \quad (3.27)$$

Donde y_i corresponde a la salida de x_i mientras que $\hat{y}_\theta(x_i)$ representa la predicción de la salida de x_i mediante un modelo de parámetro(s) θ .

Muchas veces es infactible encontrar el óptimo de forma analítica o bien, el algoritmo del gradiente clásico se queda atrapado en mínimos locales. Una forma distinta de ver el problema es considerar que $(x_i, y_i) \sim \mu$ iid para una distribución μ desconocida. Desde ese punto de vista, el problema se reduce a minimizar $\mathbb{E}(J(y, \hat{y}_\theta(x)))$. Este tipo de problemas puede ser escrito en general como

$$\min_{\theta} \mathbb{E}(f(\theta, X)), \quad X \sim \mu \text{ desconocida} \quad (3.28)$$

Una alternativa al método del gradiente clásico $\theta^{\tau+1} = \theta^\tau - \beta_{\tau+1} \nabla_{\theta} \mathbb{E}(f(\theta^\tau, X))$ consiste en utilizar las observaciones iid $(x_i)_{i \geq 1} \sim \mu$ al momento de iterar, considerando una observación por iteración en vez del funcional $\mathbb{E}(f(\theta^\tau, X))$ completo, es decir:

$$\theta^{\tau+1} = \theta^\tau - \eta_{\tau+1} \nabla_{\theta} f(\theta^\tau, x_{\tau+1}) \quad (3.29)$$

Donde ahora η_τ no necesariamente se debe calcular mediante otro problema de minimización, sino que pueden usarse secuencias heurísticas que en la práctica funcionan bien. Además, en cada iteración se necesita evaluar una sola vez $\nabla_{\theta} f$ y no hace falta calcular su esperanza.

Este método es conocido como método del gradiente estocástico (SGD), donde el término $\nabla_{\theta} f(\theta^\tau, x_{\tau+1})$ puede ser visto como un gradiente exacto perturbado:

$$\nabla_{\theta} f(\theta^\tau, x_{\tau+1}) = \nabla_{\theta} \mathbb{E}(f(\theta^\tau, X)) + \Delta_t \quad (3.30)$$

Donde $\Delta_t = \nabla_{\theta} f(\theta^\tau, x_{\tau+1}) - \nabla_{\theta} \mathbb{E}(f(\theta^\tau, X))$ cumple que $\mathbb{E}(\Delta_t) = 0$ ya que de acuerdo a la regla integral de Leibniz, $\nabla_{\theta} \mathbb{E}(f(\theta^\tau, X)) = \mathbb{E}(\nabla_{\theta} f(\theta^\tau, X))$.

Esta propiedad del gradiente con ruido es la que le permite al algoritmo del gradiente estocástico poder saltarse con mayor facilidad los mínimos locales ya que la evolución de θ_τ contiene una parte aleatoria que no depende del valor del gradiente en el punto.

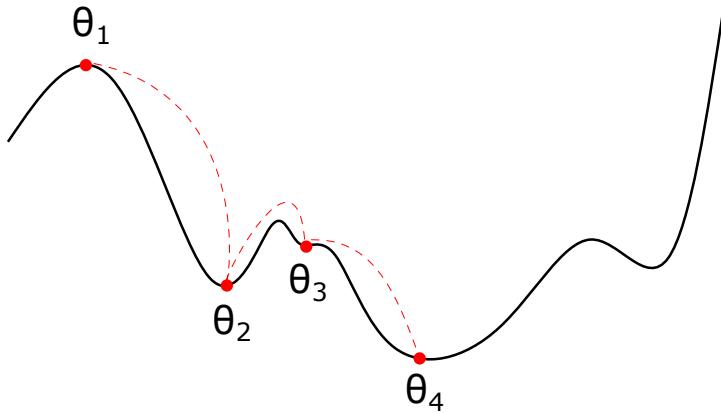


Fig. 18. Posibles iteraciones del algoritmo SGD. El algoritmo del gradiente clásico hubiese quedado atrapado en θ_2 ya que en dicho punto el gradiente es nulo por lo que no hay desplazamiento.

Otra de las ventajas que tiene este algoritmo es que permite entrenar modelos con datos a medida que van llegando (actualización en tiempo real) y no hace falta calcular el funcional de optimización utilizando todos los datos anteriores.

Si bien no está garantizada la convergencia a un óptimo global, este método funciona bien en la práctica mediante secuencias probadas de (η_τ) (learning rate). Una selección usual es usar η_τ constante o constante por tramos.

Un resultado teórico simple acerca del SGD es el siguiente:

Teorema 3.2 (Sigmund-Robins). *Bajo hipótesis razonables sobre f (regularidad en θ , integrabilidad de $\nabla_\theta f$ y cotas) y tasas de aprendizaje suficientemente pequeñas (por ejemplo, $\eta_\tau = 1/\tau$), la sucesión $(\theta^\tau)_{\tau \geq 1}$ converge c.s. al conjunto de puntos críticos de $\mathbb{E}(f(\theta, X))$.*

En particular, para f convexa, la sucesión converge al conjunto de argumentos minimizantes de $\mathbb{E}(f(\theta, X))$.

Para el problema de minimización del funcional del perceptrón, se puede utilizar el método del gradiente estocástico. En este caso, el algoritmo iterativo tiene la siguiente estructura:

$$\begin{aligned}\theta^{\tau+1} &= \theta^\tau - \eta_\tau \nabla_\theta J_P(\theta^\tau, x_i) \\ &= \theta^\tau + \eta_\tau \phi(x_i) t_i.\end{aligned}\tag{3.31}$$

Es importante notar que al actualizar el vector θ , el conjunto de puntos mal clasificados \mathcal{M} va a cambiar, pues (esperamos que) en cada iteración los elementos del conjunto de puntos mal clasificados vaya disminuyendo.

Por lo tanto, el algoritmo de entrenamiento para el perceptrón es el siguiente:

- i) se recorre el conjunto de puntos de entrenamiento $\{x_i\}_{i=1}^N$,
- ii) si el punto x_i fue clasificado correctamente el vector de pesos mantiene igual

- iii) si x_i fue clasificado incorrectamente, el vector θ^τ es actualizado según la ec. (3.31) con $\eta = 1$ mediante

$$\theta^{\tau+1} = \theta^\tau + \phi(x_i)t_i. \quad (3.32)$$

Es decir, el parámetro θ está paso a paso modificado en la dirección de las características $\phi(x_i)$ con multiplicador ± 1 en base a la clase verdadera de x_i hasta que todos los puntos de \mathcal{D} están bien clasificados.

3.3. Clasificación probabilística: modelo generativo

Los modelos que hemos revisado hasta este punto son del tipo *discriminativo*, es decir, modelan directamente la función $f : x \mapsto c$. Con una interpretación probabilística, esto es equivalente a modelar la probabilidad condicional $\mathbb{P}(\mathcal{C}_k|x)$, es decir, dado que conozco el input (o características de) x , cuál es la distribución de probabilidad sobre las clases. Sin embargo, hemos considerado métodos determinísticos, que solo asignan probabilidad 1 a una sola clase.

Un paradigma alternativo es considerar es un enfoque *generativo*, en el cual modelamos dos objetos: en primer lugar la “probabilidad condicional de clase” la cual representa cómo distribuyen los valores de los inputs x cuando la clase es, por ejemplo, \mathcal{C}_k , denotada por $\mathbb{P}(x|\mathcal{C}_k)$. En segundo lugar las “probabilidades de clase”, o el prior sobre clases, denotada $\mathbb{P}(\mathcal{C}_k)$. Luego, podemos calcular la densidad posterior sobre las clases dado un input x usando el Teorema de Bayes de acuerdo a

$$\mathbb{P}(\mathcal{C}_k|x) = \frac{\mathbb{P}(x|\mathcal{C}_k)\mathbb{P}(\mathcal{C}_k)}{\mathbb{P}(x)}. \quad (3.33)$$

Para el caso de 2 clases, es posible calcular la probabilidad de la clase \mathcal{C}_1 dado x de la forma:

$$\begin{aligned} \mathbb{P}(\mathcal{C}_1|x) &= \frac{\mathbb{P}(x|\mathcal{C}_1)\mathbb{P}(\mathcal{C}_1)}{\mathbb{P}(x)} \\ &= \frac{\mathbb{P}(x|\mathcal{C}_1)\mathbb{P}(\mathcal{C}_1)}{\mathbb{P}(x|\mathcal{C}_1)\mathbb{P}(\mathcal{C}_1) + \mathbb{P}(x|\mathcal{C}_2)\mathbb{P}(\mathcal{C}_2)} \\ &= \frac{1}{1 + \frac{\mathbb{P}(x|\mathcal{C}_2)\mathbb{P}(\mathcal{C}_2)}{\mathbb{P}(x|\mathcal{C}_1)\mathbb{P}(\mathcal{C}_1)}} \\ &= \frac{1}{1 + \exp(-r)} = \sigma(r). \end{aligned} \quad (3.34)$$

Donde hemos introducido la notación $r = r(x) = \ln\left(\frac{\mathbb{P}(x|\mathcal{C}_1)\mathbb{P}(\mathcal{C}_1)}{\mathbb{P}(x|\mathcal{C}_2)\mathbb{P}(\mathcal{C}_2)}\right)$ y la función logística definida mediante $\sigma(r) = \frac{1}{1+e^{-r}}$, la cual tiene propiedades que serán útiles en el entrenamiento, en particular:

$$\text{reflejo: } \sigma(-r) = 1 - \sigma(r) \quad (3.35)$$

$$\text{derivada: } \frac{d}{dr}\sigma(r) = \sigma(r)(1 - \sigma(r)) \quad (3.36)$$

$$\text{inversa: } r(\sigma) = \ln\left(\frac{\sigma}{1 - \sigma}\right). \quad (3.37)$$

Observación 3.4. Si bien la expresión de la distribución condicional en la ec. (3.34) parece una presentación antojadiza para hacer aparecer la función logística (sigmoide), pues $r = r(x)$ puede ser cualquier cosa. Sin embargo, veremos que existe una elección particular de las distribuciones condicionales de clase que lleva a un r que es efectivamente lineal en x . En general, nos referiremos a este clasificador como **regresión logística** en dicho caso, es decir, cuando $r(x) = a^\top x + b$.

Podemos ahora considerar el caso de múltiples clases $\{\mathcal{C}_1, \dots, \mathcal{C}_K\}$, donde un desarrollo similar al anterior resulta en:

$$\mathbb{P}(\mathcal{C}_i|x) = \frac{\mathbb{P}(x|\mathcal{C}_i)\mathbb{P}(\mathcal{C}_i)}{\sum_j \mathbb{P}(x|\mathcal{C}_j)\mathbb{P}(\mathcal{C}_j)} = \frac{\exp(s_i)}{\sum_j \exp(s_j)}, \quad (3.38)$$

donde hemos denotado $s_i = \log(\mathbb{P}(x|\mathcal{C}_i)\mathbb{P}(\mathcal{C}_i))$. La función que aparece al lado derecho de la ec. (3.38) se conoce como *exponencial normalizada* o *softmax*, y corresponde a una generalización de la función logística a múltiples clases. Además, esta función tiene la propiedad de ser una aproximación suave de la función máximo y convertir cualquier vector $s = [s_1, \dots, s_k]$ en una distribución de probabilidad, donde podemos hablar de “la probabilidad de ser clase \mathcal{C}_k ”.

3.3.1. Regresión logística

Analizaremos ahora los supuestos sobre el modelo generativo (i.e., las probabilidades de clase y condicionales) para encontrar un r –en la ec. (3.34)– que resulta en la bien conocida regresión logística. Consideraremos el caso binario donde las densidades condicionales de clase son Gaussianas multivariadas, dadas por

$$p(x|\mathcal{C}_k) \sim \mathcal{N}(\mu_k, \Sigma) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma|^{\frac{1}{2}}} \exp(-\frac{1}{2}(x - \mu_k)^\top \Sigma^{-1}(x - \mu_k)) \quad k \in \{1, 2\}. \quad (3.39)$$

Donde $\mu_k \in \mathbb{R}^M$ corresponde al centroide de la clase \mathcal{C}_k y $\Sigma \in \mathbb{R}^{M \times M}$ simétrica y definida positiva, corresponde a la matriz de covarianza de las clases (misma matriz para todas las clases). Para este caso, se tiene que para la ecuación (3.34):

$$r = \ln \left(\frac{\mathbb{P}(x|\mathcal{C}_1)\mathbb{P}(\mathcal{C}_1)}{\mathbb{P}(x|\mathcal{C}_2)\mathbb{P}(\mathcal{C}_2)} \right) = \ln \left(\frac{\exp(-\frac{1}{2}(x - \mu_1)^\top \Sigma^{-1}(x - \mu_1))\mathbb{P}(\mathcal{C}_1)}{\exp(-\frac{1}{2}(x - \mu_2)^\top \Sigma^{-1}(x - \mu_2))\mathbb{P}(\mathcal{C}_2)} \right) \quad (3.40)$$

$$= -\frac{1}{2}(x - \mu_1)^\top \Sigma^{-1}(x - \mu_1) + \frac{1}{2}(x - \mu_2)^\top \Sigma^{-1}(x - \mu_2) + \ln \left(\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)} \right) \quad (3.41)$$

$$= \frac{1}{2} \left(x^\top \Sigma^{-1}(\mu_1 - \mu_2) + (\mu_1 - \mu_2)^\top \Sigma^{-1}x - \mu_1^\top \Sigma^{-1}\mu_1 + \mu_2^\top \Sigma^{-1}\mu_2 \right) + \ln \left(\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)} \right) \quad (3.42)$$

$$= (\mu_1 - \mu_2)^\top \Sigma^{-1}x + \frac{1}{2} (\mu_2^\top \Sigma^{-1}\mu_2 - \mu_1^\top \Sigma^{-1}\mu_1) + \ln \left(\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)} \right) \quad (3.43)$$

$$= a^\top x + b \quad (3.44)$$

donde hemos usado la notación

$$a = \Sigma^{-1}(\mu_1 - \mu_2) \quad (3.45)$$

$$b = \frac{1}{2}(\mu_2^\top \Sigma^{-1}\mu_2 - \mu_1^\top \Sigma^{-1}\mu_1) + \ln \left(\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)} \right). \quad (3.46)$$

y el hecho de que Σ^{-1} es simétrica.

Lo anterior nos entrega la regresión logística (lineal) para el caso binario, donde al incorporar la expresión anterior en la ec. (3.34) obtenemos

$$p(\mathcal{C}_k|x) = \sigma(a^\top x + b) = \frac{1}{1 + \exp(-a^\top x - b)}. \quad (3.47)$$

Observación 3.5. La reducción de r a una forma lineal prueba que es necesario considerar matrices de covarianza Σ iguales para todas las clases, ya que de lo contrario no se podrían simplificar los términos cuadráticos $x^\top \Sigma^{-1}x$.

Observación 3.6. El modelo de clasificación binaria en la ec. 3.47 es conocido como *regresión logística* y es la consecuencia de asumir un modelo generativo Gaussiano con distintas medias pero la misma

varianza para una de las clases. Como consecuencia, la región de decisión se encuentra imponiendo que la probabilidad de ser clase \mathcal{C}_1 sea $1/2$ (y por ende, ser \mathcal{C}_2 también tiene probabilidad $1/2$), lo cual se tiene para

$$a^\top x + b = 0. \quad (3.48)$$

Ahora que hemos definido el modelo para nuestro problema de clasificación, aflora naturalmente la siguiente pregunta: ¿Cómo ajustar los parámetros de las condicionales a la clase y priors respectivamente? Para esto, reiteremos que los parámetros del modelos serán los de la probabilidad de clase $p(\mathcal{C}_k)$ y de la probabilidades condicionales de clase $p(x|\mathcal{C}_k)$. Respectivamente:

- Probabilidad de clase:

$$p(\mathcal{C}_1) = \pi, \quad p(\mathcal{C}_2) = 1 - \pi, \quad (3.49)$$

es decir, un parámetro π (por determinar).

- Probabilidad condicional de clase:

$$p(x|\mathcal{C}_k) = \mathcal{N}(\mu_k, \Sigma); k = 1, 2, \quad (3.50)$$

es decir, parámetros $\mu_1 \in \mathbb{R}^M, \mu_2 \in \mathbb{R}^M, \Sigma \in \mathbb{R}^M \times \mathbb{R}^M$ (por determinar) o, equivalentemente, $M + M + M(M + 1)/2 = M(M + 5)/2$ parámetros escalares (considerando que Σ es simétrica).

Denotaremos todos los parámetros mediante el parámetro agregado $\theta = \{\pi, \mu_1, \mu_2, \Sigma\}$.

Realizaremos el entrenamiento del modelo, i.e., encontrar θ en base a un conjunto de datos \mathcal{D} , mediante el método de máxima verosimilitud. Para esto, notemos que solo hemos definido “la probabilidad de ser clase \mathcal{C}_1 ”, noción que extendemos usando la codificación de clases $t \in \{0, 1\}$. Con esta codificación, donde la observación (x_i, t_i) corresponde a clase \mathcal{C}_1 con $t_i = 1$ y a clase \mathcal{C}_0 con $t = 0$, podemos expresar la verosimilitud con una observación mediante:

$$L_i(\theta) = p(x_i, t_i | \theta) = p(x_i, \mathcal{C}_1 | \theta)^{t_i} p(x_i, \mathcal{C}_0 | \theta)^{1-t_i}. \quad (3.51)$$

Esta expresión equivale a la probabilidad de ser clase \mathcal{C}_1 cuando $t_i = 1$ y a la probabilidad de ser clase \mathcal{C}_0 cuando $t_i = 0$. Consecuentemente, para un conjunto de datos \mathcal{D} de la forma

$$X = \begin{pmatrix} x_1^\top \\ \vdots \\ x_N^\top \end{pmatrix} \in \mathbb{R}^{N \times M}, \quad T = \begin{pmatrix} t_1 \\ \vdots \\ t_N \end{pmatrix} \in \{0, 1\}^N \text{ es decir, codificación } 0 - 1. \quad (3.52)$$

podemos escribir la verosimilitud mediante $L(\theta) = p(X, T | \theta)$, luego:

$$L(\theta) = \prod_{i=1}^N p(x_i, t_i | \theta) = \prod_{i=1}^N p(x_i, \mathcal{C}_1 | \theta)^{t_i} p(x_i, \mathcal{C}_0 | \theta)^{1-t_i} \quad (3.53)$$

$$= \prod_{i=1}^N (p(x_i | \mathcal{C}_1, \theta) p(\mathcal{C}_1 | \theta))^{t_i} (p(x_i | \mathcal{C}_0, \theta) p(\mathcal{C}_0 | \theta))^{1-t_i} \quad (3.54)$$

$$= \prod_{i=1}^N (\pi \mathcal{N}(x_i | \mu_1, \Sigma))^{t_i} ((1 - \pi) \mathcal{N}(x_i | \mu_2, \Sigma))^{1-t_i}. \quad (3.55)$$

Al igual que en el escenario de regresión lineal, para proceder con la optimización nos enfocamos en la log-verosimilitud, dada por:

$$l(\theta) := \log L(\theta) = \sum_{i=1}^N (t_i(\log(\pi) + \log(\mathcal{N}(x_i | \mu_1, \Sigma))) + (1 - t_i)(\log(1 - \pi) + \log(\mathcal{N}(x_i | \mu_2, \Sigma)))) \quad (3.56)$$

Aplicando entonces las condiciones de primer orden, tenemos que

- 1) Con respecto a π :

$$\begin{aligned}
\frac{\partial \log(L)}{\partial \pi} &= \sum_{i=1}^N \frac{t_i}{\pi} - \frac{1-t_i}{1-\pi} = 0 \\
\Rightarrow (1-\pi) \sum_{i=1}^N t_i &= \pi \sum_{i=1}^N (1-t_i) \\
\Rightarrow \sum_{i=1}^N t_i &= \pi N \quad \Rightarrow \quad \pi = \frac{\sum_{i=1}^N t_i}{N} = \frac{N_1}{N_1 + N_2}
\end{aligned} \tag{3.57}$$

Donde $N_i := \text{Card}(x : x \in \mathcal{C}_i)$. Por lo tanto, el EMV de pi colapsa a la regla de Laplace.

- 2) Con respecto a μ_1 :

$$\begin{aligned}
\frac{\partial \log(L)}{\partial \mu_1} &= \sum_{i=1}^N t_i \frac{\partial}{\partial \mu_1} \left(-\frac{1}{2} (x_i - \mu_1)^\top \Sigma^{-1} (x_i - \mu_1) \right) \\
&= \sum_{i=1}^N t_i (\Sigma^{-1} (x_i - \mu_1)) = \Sigma^{-1} \sum_{i=1}^N t_i (x_i - \mu_1) = 0 \\
\Rightarrow \sum_{i=1}^N t_i x_i &= \mu_1 \sum_{i=1}^N t_i \quad \Rightarrow \quad \mu_1 = \frac{1}{N_1} \sum_{i=1}^N t_i x_i = \frac{1}{N_1} \sum_{x_i \in \mathcal{C}_1} x_i
\end{aligned} \tag{3.58}$$

De forma análoga:

$$\mu_2 = \frac{1}{N_2} \sum_{x_i \in \mathcal{C}_2} x_i \tag{3.59}$$

Observación 3.7. La ec. (3.57) revela que el parámetro óptimo π es precisamente la razón entre la cantidad de elementos de la clase \mathcal{C}_1 y el total de datos, esto es porque π es la probabilidad de ser clase \mathcal{C}_1 ; lo mismo aplica para $(1-\pi)$ y clase \mathcal{C}_0 . Además, de las ecs. (3.58)-(3.59) vemos también que el estimador de MV para la media de las clases es la media muestral de los datos disponibles por cada clase. Queda la siguiente pregunta entonces: ¿cuál es el estimador de MV de Σ ?

3.3.2. Regresión logística v/s modelo generativo

Recordemos que los supuestos tomados sobre el modelo generativo para el problema de clasificación resultaron en:

$$p(\mathcal{C}_1|x) = \sigma(w^\top x) = \frac{1}{1 + e^{-w^\top x}}, \tag{3.60}$$

donde por claridad de notación hemos elegido la representación lineal ($w^\top x$) y no afín ($a^\top x + b$).

En el caso anterior se ha entrenado el modelo generativo completo, es decir, $\pi, \mu_1, \mu_2, \Sigma$, lo cual tiene la ventaja de tener solución en forma cerrada, sin embargo, puede ser innecesario cuando solo necesitamos conocer el peso w en la ecuación anterior. Otra forma de entrenar la regresión logística es considerar el modelo discriminativo en la ec. (3.60) y optimizar directamente la verosimilitud sobre este modelo para encontrar w ; en vez de todos los parámetros $\pi, \mu_1, \mu_2, \Sigma$ del modelo generativo.

Calculemos la verosimilitud de la regresión logística con datos $\mathcal{D} = \{(x_i, t_i)\}_{i=1}^N$, para hacer la notación más compacta denotamos $\sigma_i = \sigma(w^\top x_i)$. Entonces:

$$p((t_i)_{i=1}^N | (x_i)_{i=1}^N, w) = \prod_{i=1}^N p(t_i | x_i, w) = \prod_{i=1}^N p(\mathcal{C}_1 | x_i)^{t_i} p(\mathcal{C}_2 | x_i)^{1-t_i} = \prod_{i=1}^N \sigma_i^{t_i} (1 - \sigma_i)^{1-t_i} \quad (3.61)$$

Con lo que la log-verosimilitud está dada por

$$l(w) = \sum_{i=1}^N t_i \log(\sigma_i) + (1 - t_i) \log(1 - \sigma_i). \quad (3.62)$$

Notemos que este problema de optimización no exhibe una solución en forma cerrada, por lo que podemos resolverlo mediante gradiente, para lo cual es necesario calcular el gradiente de $l(w)$ respecto a w :

$$\nabla_w l(w) = \sum_{i=1}^N (t_i - \sigma_i) x_i, \quad (3.63)$$

lo cual nos da una regla de ajuste $\theta \mapsto \theta - \eta \sum_{i=1}^N (\sigma_i - t_i) x_i$, o bien

$$\theta \mapsto \theta + \eta(t_i - \sigma_i)x_i, \quad (3.64)$$

si tomamos los datos de “a uno” (método del gradiente estocástico).

Observación 3.8. Notemos que la regla de ajuste del parámetro θ en la ec. (3.64) recuerda el ajuste del Perceptrón en la ec. (3.31), donde los puntos mal clasificados se agregan como características al vector θ . Sin embargo, a diferencia del perceptrón, incluso los elementos bien clasificados (σ cercano a 1 ó 0) tomarán parte en la actualización, forzando una fuerte componente de sobreajuste al hacer tender θ a infinito.

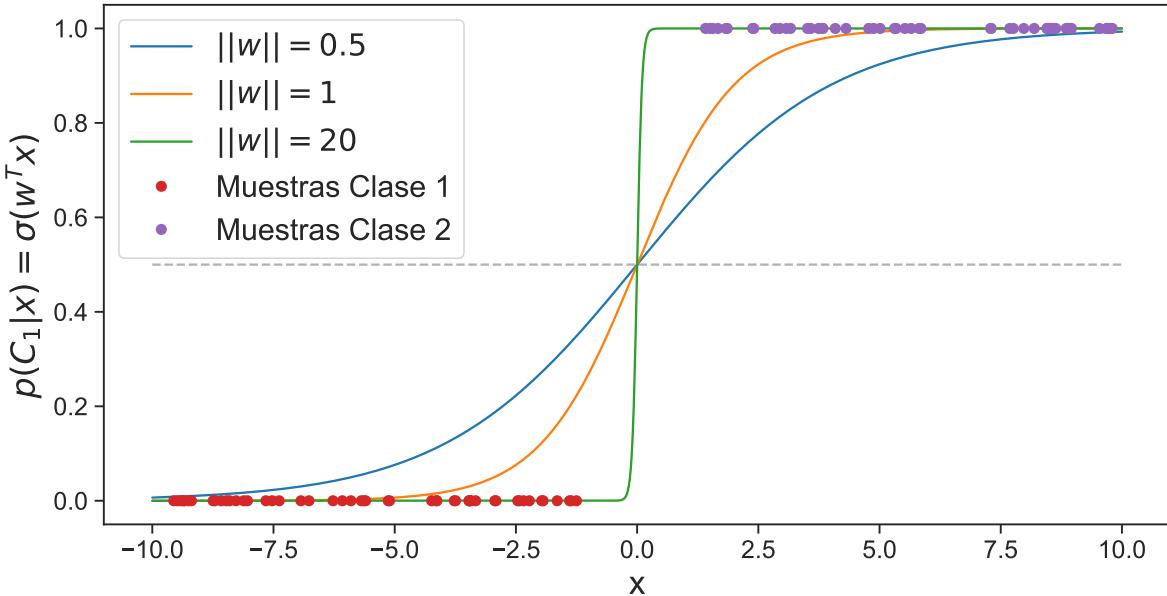


Fig. 19. En gris la frontera de decisión: una nueva entrada x_* será asignada a la clase \mathcal{C}_1 si $p(\mathcal{C}_1|x_*) > \frac{1}{2}$, en caso contrario, será asignada a \mathcal{C}_2 . Se observa que al entrenar con más muestras, $\|w\|$ crece por lo que el parámetro se sobreajusta a los datos y el clasificador converge a una función indicatriz.

4. Selección y evaluación de modelos

Dado un conjunto de datos, existen muchos posibles modelos para poder realizar el aprendizaje, por lo que surge la pregunta natural de qué modelo elegir. Una respuesta rápida a esta pregunta sería elegir el modelo que mejor se ajuste a los datos en el entrenamiento. El problema de utilizar este criterio es el sobreajuste, el cual puede ser observado en la Figura 20 donde se observa que en la tercera imagen, el modelo aprende oscilaciones que muy probablemente son generadas por los ruidos de las observaciones.

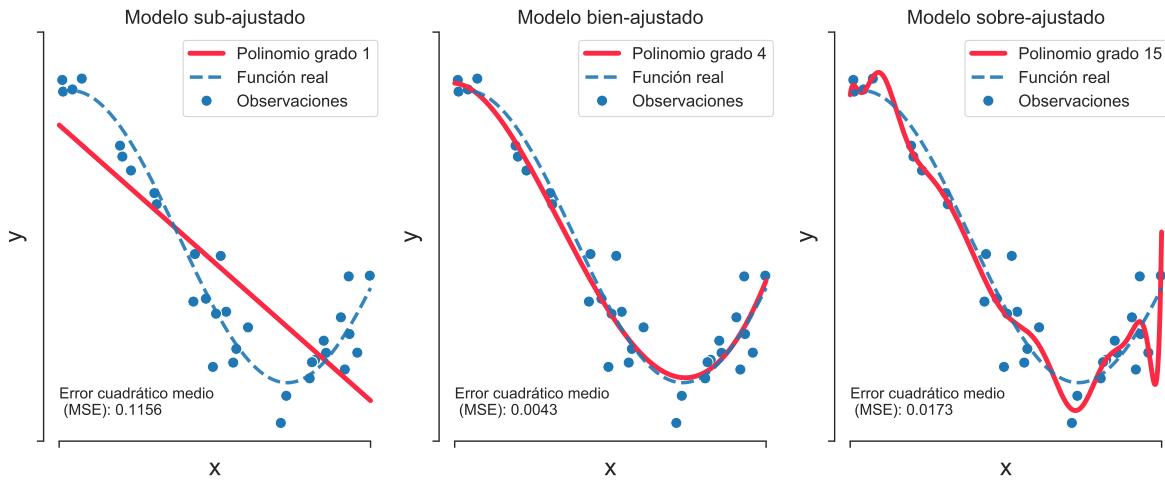


Fig. 20. Ejemplos de sub, sobre y correcto ajuste.

El problema de sobreajuste ocurre, por ejemplo, en el ajuste polinomial (donde se debe elegir un cierto grado de regresión para los datos) ya que es sabido, de acuerdo al teorema de interpolación polinómica de Lagrange, que para n puntos siempre existirá un polinomio de grado menor a n que pase exactamente por dichos puntos, por lo que es posible tener un error de ajuste nulo en los datos de entrenamiento, pero con un alto error de predicción en datos no vistos.

Teniendo en cuenta lo anterior, se hace natural escoger el modelo que mejor se ajuste a datos de un conjunto de test, es decir, que no hayan estado en el conjunto de entrenamiento. Así, buscamos el modelo que logre **generalizar** lo aprendido en el entrenamiento a datos desconocidos. Aún teniendo en cuenta, se hace necesario definir métricas para determinar *qué* es un buen modelo. Es esta la pregunta que se intentará resolver en este capítulo.

4.1. Descomposición sesgo-varianza

En el capítulo de regresión lineal se estudió el método de mínimos cuadrados regularizados, el cual generaba un regresor con un mayor error cuadrático medio (ECM) al evaluarlo dentro del conjunto de entrenamiento, pero un mejor desempeño que MC al evaluarlo fuera de muestra. Esto ocurría debido a que, si bien MCR perdía la propiedad de ser un estimador insesgado, la penalización en la norma del parámetro permitía disminuir la varianza del estimador, lo cual resultaba en una disminución del error de estimación debido a la descomposición sesgo-varianza.

El objetivo de esta sección será probar dicha descomposición para un modelo general.

Definición 4.1. Sea $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n \subset \mathbb{R}^N \times \mathbb{R}$ un conjunto de observaciones generadas por una función desconocida $f : \mathbb{R}^N \rightarrow \mathbb{R}$ mediante $y = f(x) + \epsilon$ donde ϵ es una v.a. (ruido) con $\mathbb{E}(\epsilon) = 0$ y $\text{Var}(\epsilon) = \sigma^2$. Sea $\hat{f}(\cdot | \mathcal{D})$ un estimador de f determinado a partir de \mathcal{D} , entonces, para un nuevo par (x, y) se tienen las siguientes definiciones:

- Error (cuadrático) esperado: $\mathbb{E}_{\mathcal{D}} \left((y - \hat{f}(x|\mathcal{D}))^2 \right)$.
- Sesgo del estimador: $\text{Bias}(\hat{f}(x|\mathcal{D})) := \mathbb{E}_{\mathcal{D}}(\hat{f}(x|\mathcal{D})) - f(x) = \mathbb{E}_{\mathcal{D}}(\hat{f}(x|\mathcal{D}) - f(x))$. Puede interpretarse como el error esperado del estimador.
- Varianza del estimador: $\text{Var}(\hat{f}(x|\mathcal{D})) = \mathbb{E}_{\mathcal{D}} \left((\hat{f}(x|\mathcal{D}) - \mathbb{E}_{\mathcal{D}}(\hat{f}(x|\mathcal{D})))^2 \right)$.

Observación 4.1. Notar que el error cuadrático esperado $\mathbb{E}_{\mathcal{D}} \left((y - \hat{f}(x|\mathcal{D}))^2 \right)$ no corresponde al error cuadrático medio (ECM) del estimador $\hat{f}(x|\mathcal{D})$ de $f(x)$, el cual viene dado por $\mathbb{E}_{\mathcal{D}} \left((f(x) - \hat{f}(x|\mathcal{D}))^2 \right)$, más bien podría interpretarse como el ECM visto como un estimador de y , aunque esto no es del todo correcto ya que y es aleatorio.

Observación 4.2. En todas las definiciones anteriores, la esperanza es tomada con respecto a \mathcal{D} , es decir, integra sobre los distintos $\{x_1, y_1\}, \dots, \{x_N, y_N\}$ generados a partir de la distribución conjunta $p(x, y)$.

De las definiciones anteriores, es posible notar que la pregunta que buscamos responder en este capítulo tiene dos tipos de respuestas:

- **Selección de Modelos:** Estimar el rendimiento de distintos modelos para elegir el mejor.
- **Validación de Modelos:** Habiendo elegido un modelo, estimar su error esperado en data desconocida.

Para lograr estudiar ambas respuestas, es de gran utilidad el siguiente teorema:

Teorema 4.1 (descomposición sesgo-varianza). *Sea $\hat{f}(x|\mathcal{D})$ un estimador de $f(x)$, entonces:*

$$\mathbb{E}_{\mathcal{D}} \left((y - \hat{f}(x|\mathcal{D}))^2 \right) = \text{Bias}^2(\hat{f}(x|\mathcal{D})) + \text{Var}(\hat{f}(x|\mathcal{D})) + \sigma^2 \quad (4.1)$$

Demostración. Para evitar sobrecargar la notación, se utilizará $\hat{f} = \hat{f}(x|\mathcal{D})$, $f = f(x)$ y $\mathbb{E} = \mathbb{E}_{\mathcal{D}}$. La demostración se hará forzando una completación de cuadrados y mostrando que los términos residuales son nulos para llegar a que $\mathbb{E} \left((y - \hat{f})^2 \right) = (\mathbb{E}(\hat{f}) - f)^2 + \mathbb{E} \left((\hat{f} - \mathbb{E}(\hat{f}))^2 \right) + \mathbb{E} \left((\epsilon - \mathbb{E}(\epsilon))^2 \right)$.

$$\begin{aligned} \mathbb{E} \left((y - \hat{f})^2 \right) &= \mathbb{E} \left((f + \epsilon - \hat{f})^2 \right) = \mathbb{E} \left(f^2 + \epsilon^2 + \hat{f}^2 + 2f\epsilon - 2f\hat{f} - 2\epsilon\hat{f} \right) \\ &= (\mathbb{E}^2(\hat{f}) - 2f\mathbb{E}(\hat{f}) + f^2) + \mathbb{E} \left(\hat{f}^2 - 2\hat{f}\mathbb{E}(\hat{f}) + \mathbb{E}^2(\hat{f}) \right) + \mathbb{E} \left((\epsilon - \mathbb{E}(\epsilon))^2 \right) \\ &\quad - 2\mathbb{E}(\epsilon\hat{f}) - 2\mathbb{E}^2(\hat{f}) + 2\mathbb{E}(\hat{f})\mathbb{E}(\hat{f}) \\ &= (\mathbb{E}(\hat{f}) - f)^2 + \mathbb{E} \left((\hat{f} - \mathbb{E}(\hat{f}))^2 \right) + \mathbb{E} \left((\epsilon - \mathbb{E}(\epsilon))^2 \right) - 2\mathbb{E}(\epsilon)\mathbb{E}(\hat{f}) \\ &= \text{Bias}^2(\hat{f}(x|\mathcal{D})) + \text{Var}(\hat{f}(x|\mathcal{D})) + \sigma^2 \end{aligned}$$

Donde se pudo usar $\mathbb{E}(\epsilon\hat{f}) = \mathbb{E}(\epsilon)\mathbb{E}(\hat{f})$ debido a que \hat{f} depende del espacio muestral \mathcal{D} y ϵ es el ruido asociado a una nueva muestra. ■

Esta descomposición muestra que la varianza intrínseca del ruido afectará directamente y de forma aditiva sobre el error de la predicción, imposibilitando realizar predicciones exactas bajo cualquier modelo aleatorio. Por otra parte, tal como se vio en mínimos cuadrados regularizados, se puede introducir sesgo en el modelo con el fin de disminuir la varianza y viceversa, lo cual crea la pregunta acerca de cuál es el par sesgo-varianza óptimo que minimiza el error total. Dicha pregunta se conoce como dilema sesgo-varianza (bias-variance tradeoff) y juega un papel importante en la selección de hiperparámetros del modelo.

De esta forma, la combinación sesgo-varianza crea un error total convexo tal como se puede observar en la siguiente figura:

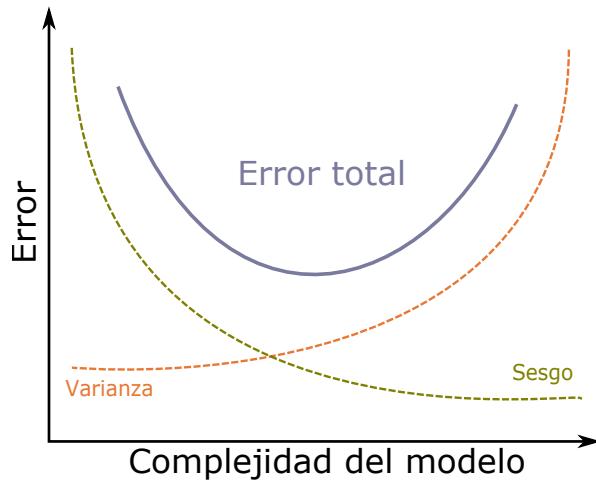


Fig. 21. Tradeoff entre el sesgo y la varianza. Se observa que el error total mínimo es alcanzado en un par (*sesgo, varianza*) específico.

4.2. Validación cruzada

Una primera forma de elegir y evaluar un modelo fuera de muestra, consiste en particionar el conjunto de datos \mathcal{D} en dos: un primer conjunto donde se realizará el entrenamiento, y otro donde se medirá el rendimiento del modelo de acuerdo a algún criterio predefinido (por ejemplo, el error cuadrático medio). Con el fin de evitar posibles sesgos provocados por una partición en específico, la evaluación de desempeño se debe realizar varias veces sobre conjuntos de validación distintos. De esta forma, al promediar los rendimientos de cada partición se obtiene un rendimiento estimado fuera de muestra, lo cual permite finalmente elegir un modelo, quedándose con aquel que reporte el menor error *out-sample*. Las distintas formas de mezclar y particionar los datos se conocen como validación cruzada.

4.2.1. Validación cruzada exhaustiva

En este tipo de validación cruzada, se prueban todas las posibles permutaciones de los datos al particionar el conjunto \mathcal{D} . Se tienen 2 técnicas exhaustivas:

- **leave p out (LpOCV):** el conjunto \mathcal{D} se partitiona dejando p elementos para validación y los $N - p$ elementos restantes se utilizan para entrenar el modelo. Este entrenamiento y cálculo de desempeño se repite $C_p^N = \frac{N!}{(N-p)!p!}$ veces, pasando por todos los posibles conjuntos de validación de tamaño p .
- **leave one out (LOOCV):** corresponde al caso anterior con $p = 1$. En este caso cada dato de \mathcal{D} es utilizado como único elemento de validación mientras el resto de los datos se utiliza para entrenar.

4.2.2. Validación cruzada no exhaustiva

- **k -fold:** el conjunto \mathcal{D} es dividido en k grupos de igual tamaño. Luego, uno de esos grupos es utilizado como validador y el resto como entrenamiento. Esto se repite k veces de forma de que todos los grupos sean validadores una y solo una vez.
- **Monte Carlo CV:** se realizan particiones binarias aleatorias de \mathcal{D} . Se entrena y evalúa usando el par de conjuntos creados en cada partición.

Observación 4.3. Una variante de la validación cruzada es dividir el conjunto \mathcal{D} en 3, donde los primeros dos conjuntos son utilizados para entrenamiento y validación, mientras que el tercero (conocido como test set) es utilizado para obtener una estimación real del desempeño fuera de muestra del modelo elegido a partir de los dos conjuntos anteriores. Esto se realiza ya que al considerar únicamente el desempeño en el conjunto de validación, por lo general se sobreestima el desempeño real fuera de muestra debido a que el modelo fue elegido precisamente tomando el que reporta el menor error dentro del conjunto de validación.

Si bien no hay una regla estándar que indique cómo particionar el conjunto, una división usual es utilizar el 50 % para entrenamiento y 25 % para validación y test.

4.3. Selección de modelo

Si bien la técnica de validación cruzada es bastante efectiva, tiene la limitación de requerir una gran cantidad de datos para poder realizar la partición de \mathcal{D} . Para los casos que en los que no se cuenta con una cantidad considerable de observaciones, se requieren herramientas más sofisticadas para poder tomar una decisión acerca de qué modelo elegir. Los dos criterios más usuales corresponden al criterio de información de Akaike y al criterio de información bayesiano.

4.3.1. Criterio de información de Akaike (AIC)

Sea $\mathcal{D} = (x_i)_{i=1}^N$ un conjunto de observaciones generadas por una distribución desconocida perteneciente a una familia paramétrica cuyos parámetros están en $\Theta \subset \mathbb{R}^d$. Bajo este modelo, se puede utilizar el estimador de máxima verosimilitud:

$$\hat{\theta} = \arg \max_{\theta \in \Theta} L(\theta | \mathcal{D}) = \arg \max_{\theta \in \Theta} l(\theta | \mathcal{D}) \quad (4.2)$$

Una forma de evaluar el desempeño real de este estimador es mediante el *riesgo de predicción*, el cual se ve reflejado en la log-verosimilitud de $\hat{\theta}$ sobre todas las posibles observaciones: $\mathbb{E}(l(\hat{\theta}|x))$. Dado que solo se cuenta con una cantidad finita de muestras, solo es posible obtener un riesgo empírico. El criterio de información de Akaike (AIC) busca ajustar este riesgo para obtener un estimador asintóticamente insesgado del riesgo real. Para esto, se tienen las siguientes definiciones para el estimador de máxima verosimilitud $\hat{\theta}$:

- **Riesgo empírico:** $R_{\mathcal{D}}(\hat{\theta}) = -\hat{l}$, donde $\hat{l} = l(\hat{\theta} | \mathcal{D})$ es la log-verosimilitud del EMV empírico.
- **Riesgo real:** $R(\hat{\theta}) = -\mathbb{E}(N \cdot l_0(\hat{\theta}))$, donde $l_0(\theta) = \mathbb{E}(l(\theta|x))$ corresponde a la log-verosimilitud de θ sobre todo el espacio muestral. Notar que se multiplica por N ya que en el riesgo empírico no se normalizó por N .

Definición 4.2 (AIC). Sea M un modelo estadístico d -paramétrico y $\mathcal{D} = (x_i)_{i=1}^N$ un conjunto de observaciones. El AIC del modelo (aproximado por \mathcal{D}) se define como

$$AIC(M, \mathcal{D}) := 2d - 2 \log \left(\hat{L}(\mathcal{D}) \right), \quad (4.3)$$

donde $\hat{L}(\mathcal{D})$ corresponde a la verosimilitud del EMV asociado a \mathcal{D} , es decir:

$$\hat{L}(\mathcal{D}) = \prod_{i=1}^N p(x_i | \hat{\theta}), \text{ para } \hat{\theta} = \arg \max_{\theta \in \Theta} L(\theta | \mathcal{D}). \quad (4.4)$$

4.3.2. Criterio de información bayesiano (BIC)

Otro enfoque para la selección de modelos corresponde al criterio de información bayesiano (o criterio de Schwarz). Dada una familia de modelos \mathcal{M} , se define un prior $p(m)$ para cada modelo $m \in \mathcal{M}$. Además, se define un prior $p(\theta|m)$ sobre los parámetros de cada modelo. El criterio de información bayesiano (BIC) elige al mejor modelo de acuerdo a la posterior $p(m|\mathcal{D})$, la cual viene dada de acuerdo al teorema de Bayes:

$$p(m|\mathcal{D}) = \frac{p(\mathcal{D}|m)p(m)}{p(\mathcal{D})} \propto p(\mathcal{D}|m)p(m). \quad (4.5)$$

De forma similar al criterio de Akaike, se puede calcular la verosimilitud del modelo $p(\mathcal{D}|m)$ mediante aproximaciones de Taylor, probando que es independiente del prior. La derivación de $p(\mathcal{D}|m)$ lleva a la siguiente definición:

Definición 4.3 (BIC). Sea M un modelo estadístico d -paramétrico y $\mathcal{D} = (x_i)_{i=1}^N$ un conjunto de observaciones. El BIC del modelo (aproximado por \mathcal{D}) se define como

$$BIC(M, \mathcal{D}) := d \cdot \log(N) - 2 \log(\hat{L}(\mathcal{D})) \quad (4.6)$$

Donde nuevamente $\hat{L}(\mathcal{D})$ corresponde a la verosimilitud del EMV asociado a \mathcal{D} .

En este caso, se vuelve a elegir el modelo que presente el menor BIC. Se observa que, al igual que AIC, BIC contiene una penalización sobre el número de parámetros por lo que también evita el sobreajuste a los datos.

Si bien existen otros métodos de selección de modelo (DIC, WAIC, entre otros), estos tienen una formulación más compleja que se escapa del alcance de este curso ya que se requieren herramientas adicionales como MCMC para el cálculo de distribuciones posteriores. Una derivación de AIC y BIC se puede encontrar en el apunte de Estadística.

4.4. Evaluación de modelos

Una vez elegido el mejor modelo de acuerdo a un criterio establecido (AIC o BIC), es deseable poder conocer el desempeño de dicho modelo. Existen varias formas de evaluar un modelo, una de ella podría ser simplemente evaluar la precisión de sus predicciones. A veces es natural fijarse en la precisión, como en los problemas de pronóstico. Otras veces la precisión es importante para evaluar diferentes modelos y elegir uno de ellos. En esta sección presentaremos dos maneras distintas de evaluar modelos, cada forma sirve en distintos escenarios, los cuales se discutirán a través de la predicción puntual, que resume la predicción de un conjunto de datos en una solo valor.

4.4.1. Error cuadrático medio

El ajuste del modelo a nuevos datos se puede resumir en una predicción puntual llamada error cuadrático medio, el cual está definido por:

$$MSE(\theta) = \frac{1}{n} \sum_{i=1}^N (y_i - \mathbb{E}(y_i|\theta))^2 \quad (4.7)$$

o su versión ponderada:

$$MSE(\theta) = \frac{1}{n} \sum_{i=1}^N \frac{(y_i - \mathbb{E}(y_i|\theta))^2}{\text{Var}(y_i|\theta)} \quad (4.8)$$

Esta forma de medir el error tiene la ventaja de ser fácilmente computable e interpretable, pero no es apropiada para modelos que están lejos de la distribución normal.

4.4.2. log-densidad predictiva o log-verosimilitud

Otra forma de realizar esta evaluación es utilizando el estadístico *log-densidad predictiva* $\log p(y|\theta)$ el cual es proporcional a error cuadrático medio si el modelo es normal con varianza constante. Estudiaremos el caso de un solo punto, para luego extrapolar a más de un punto.

Predictive accuracy para un punto: sea f el modelo real, y las observaciones (es decir, una realización del dataset y de la distribución $f(y)$), y llamaremos \tilde{y} a la data futura o un dataset alternativos que podemos ver. El ajuste predictivo out-of-sample para un nuevo punto \tilde{y}_i está dado por:

$$\log p_{\text{post}}(\tilde{y}_i) = \log \mathbb{E}[p(\tilde{y}_i|\theta)] = \log \int p(\tilde{y}_i|\theta)p_{\text{post}}(\theta)d\theta \quad (4.9)$$

Promedio de las distribuciones para un punto: al tener un dato nuevo \tilde{y}_i entonces se puede calcular el la log-densidad predictiva (elpd, por su sigla en inglés) para el nuevo punto:

$$\begin{aligned} \text{elpd} &= \mathbb{E}_f[\log p_{\text{post}}(\tilde{y}_i)] \\ &= \int \log p_{\text{post}}(\tilde{y}_i)f(\tilde{y}_i)d\tilde{y} \end{aligned} \quad (4.10)$$

Promedio de las distribuciones para datasets futuros: como usualmente, no se tiene solo un punto, se debe realizar la suma sobre el conjunto de puntos, calculando así la log-densidad predictiva puntual (elppd, por su sigla en inglés).

$$\text{elppd} = \sum_{i=1}^N \mathbb{E}_f[\log p_{\text{post}}(\tilde{y}_i)] \quad (4.11)$$

En la práctica, como siempre se tiene la distribución de todos los modelos y la expresión anterior requiere de esto, se suele calcular el estadístico sobre una estimación de un modelo $\hat{\theta}$ (como por ejemplo, el máximo de la función de verosimilitud):

$$\text{elppd}|\hat{\theta} = \sum_{i=1}^N \mathbb{E}_f[\log p_{\text{post}}(\tilde{y}_i|\hat{\theta})] \quad (4.12)$$

Finalmente, una última extensión de este estadístico es cuando se puede tener *draws* de la posterior, es decir, tenemos $\{\theta^s\}_{s=1}^S$, entonces el lppd computado es:

$$\text{computed lppd} = \sum_{i=1}^N \left(\frac{1}{S} \sum_{s=1}^S p(y_i|\theta^s) \right) \quad (4.13)$$

4.5. Promedio de modelos

Hay veces que no queremos elegir un solo modelo, puesto que quizás nos interesan las estimaciones de dos o más modelos. Para estos casos, se puede utilizar la técnica de *model averaging*, la cual consiste en utilizar una combinación lineal de distintos modelos.

Sea \mathcal{M} un conjunto de modelos donde cada modelo $m \in \mathcal{M}$ tiene asociada una distribución μ_m sobre el espacio muestral. Un promedio de modelos $\hat{\mu}$ es una combinación convexa de los modelos de \mathcal{M} , es decir:

$$\hat{\mu} = \sum_{m \in \mathcal{M}} c(m) \mu_m \quad (4.14)$$

Donde los pesos $(c(m))_{m \in \mathcal{M}} \subset \mathbb{R}_+$ cumplen que $\sum_{m \in \mathcal{M}} c(m) = 1$.

La principal dificultad para elegir los pesos, está en que se debe asegurar que la suma de los pesos sea unitaria y que estos sean todos positivos. Una forma de asegurar esto es aplicar una función f positiva sobre una función g (*score*) que evalúe el modelo, de modo que:

$$\sum_{m \in \mathcal{M}} (f \circ g)(m) > 0 \quad (4.15)$$

De esta forma, se pueden definir los pesos mediante normalización:

$$c(s) = \frac{(f \circ g)(s)}{\sum_{m \in \mathcal{M}} (f \circ g)(m)} \quad (4.16)$$

Para la función g se pueden utilizar los criterios AIC o BIC ya que entregan una evaluación del desempeño relativo del modelo. Por otra parte, usando $f(x) = \exp(x)$ los pesos $(c(m))_{m \in \mathcal{M}}$ vienen dados por la función softmax:

$$c_{AIC}(s) = \frac{\exp\left\{\frac{1}{2}\text{AIC}(s)\right\}}{\sum_{m \in \mathcal{M}} \exp\left\{\frac{1}{2}\text{AIC}(m)\right\}} \quad c_{BIC}(s) = \frac{\exp\left\{\frac{1}{2}\text{BIC}(s)\right\}}{\sum_{m \in \mathcal{M}} \exp\left\{\frac{1}{2}\text{BIC}(m)\right\}} \quad (4.17)$$

5. Support-vector machines

Una de las desventajas de los clasificadores lineales vistos en el Capítulo 3 (en el caso binario) es la falta de atención al margen de las clases, es decir, la región entre las muestras de ambas clases, pues en esta región se encontrará el hiperplano de decisión. Esta distancia es relevante, pues nos da un sentido de generalización, es decir, los elementos de la clase 1 *más parecidos* a los de la clase -1 no están justo en el borde de las clases, sino que existe una zona donde podrían haber nuevas muestras en torno a datos existentes de clase, e.g., 1 y aún estar dentro de la región de clase 1. El tratamiento de este concepto (o la falta de él) es claro en los métodos anteriores: en el perceptrón, todas las soluciones que dividen las muestras clases ± 1 son “igual de buenas”, es decir, el perceptrón es insensible al margen descrito arriba. Por otro lado, para los métodos que funcionan mediante gradiente descendente como mínimos cuadrados o regresión logística, este margen se maximiza *indirectamente*, lo cual conlleva a inestabilidades como en el caso en donde el parámetro de la regresión logística diverge.

Para resolver el problema descrito anteriormente consideraremos las **máquinas de soporte vectorial** (SVM, por sus siglas en inglés). Como veremos en este capítulo, además de resolver el problema de clasificación binaria y lineal, la solución de SVM puede ser ocupado en muchas más situaciones, por esta razón, dedicamos un capítulo completo a este método en vez de verlo como un método de clasificación más.

5.1. Idea general

Consideremos un problema de clasificación donde las clases son linealmente separables. Como podemos ver en la Fig. 22 (izquierda), en este caso existen diferentes hiperplanos (clasificadores lineales) que separan los datos de ambas clases de forma correcta. Cada una de estas soluciones define un modelo, donde dado un nuevo dato x_* , podemos evaluar a qué clase pertenece. Evidentemente, la asignación de clase usando cada uno de los clasificadores mostrados en la figura anterior (izquierda) son similares en la mayoría del espacio salvo la región cercana al límite de las clases; es precisamente en este lugar donde nos enfocamos en este capítulo.

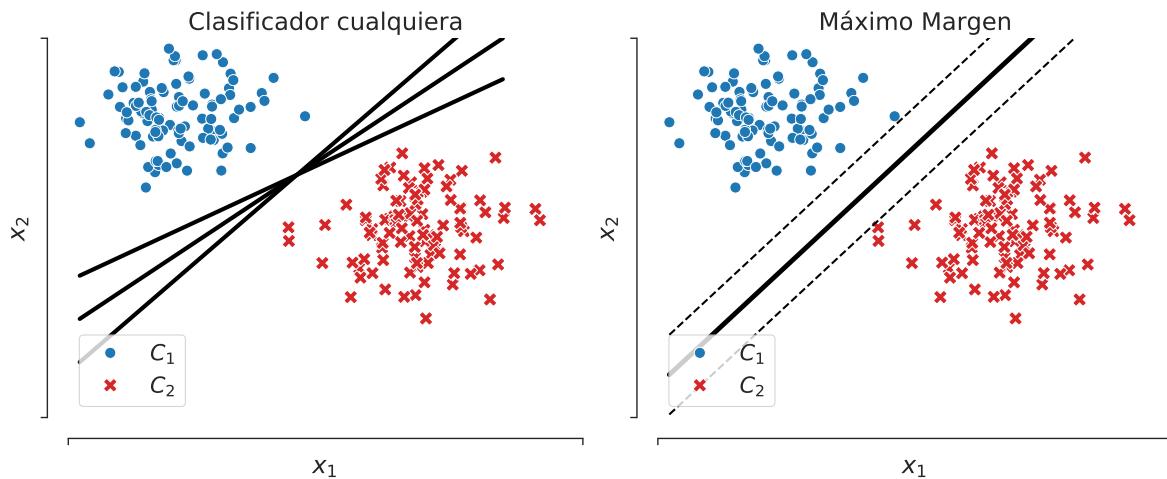


Fig. 22. Clasificadores lineales: varios clasificadores (izquierda) y clasificador de máximo margen (derecha)

Aquí entonces aflora la siguiente pregunta, como todas los clasificadores en la Fig. 22 (izquierda) separan de forma correcta los datos disponibles, ¿cuál de ellos debemos elegir? Elegiremos un *clasificador de máximo margen*, es decir, el clasificador que nos entregue la máxima separación entre los datos y las regiones definidas como correspondiente a cada clase. Una ilustración del clasificador de máximo margen

se presenta en la Fig. 22 (derecha). En donde se ve que encontrar este clasificador es equivalente a encontrar una *cinta* de ancho máximo que separe los datos de ambas clases.

El argumento de ocupar dicho criterio es la búsqueda de buenas propiedades de generalización. Intuitivamente, esta propiedad se obtiene debido a que asumiendo que los datos generados en cada clase provienen de una distribución latente, es de esperar que si se obtienen nuevos datos desde la misma distribución, estos estén cerca de los datos observados inicialmente. De este forma, con el máximo margen se pretende maximizar la probabilidad de que los nuevos datos de clase 1 (cf. -1) sean bien clasificados también.

Una propiedad clave del clasificador de máximo margen es que este queda definido únicamente por algunos datos, ver Fig. 22 (derecha). Esto inmediatamente resuelve el problema de desbalances de clase o de que las clases tengan formas distintas (problema mencionado en clasificación con mínimos cuadrado o con el discriminante de Fisher). Es decir, la solución de máximo margen se mantiene si agregamos datos (de cualquier clase) que están fuera del margen. Los datos (o vectores, pues recordemos que $x \in \mathbb{R}^M$) que definen el margen los llamaremos vectores de soporte (*support vectors*), cuya función es restringir la rotación y expansión del margen.

Finalmente, una diferencia fundamental entre un clasificador que se enfoca en el margen y otros métodos que hemos visto (e.g., naïve Bayes, regresión lineal, regresión logística) es que estos últimos aprenden incorporando todos los datos para formar la respuesta, no solo los que están en el margen. En cambio, SVM define el clasificador utilizando únicamente los vectores soporte ya que si bien todos los datos se deben analizar para encontrar los vectores soporte, los datos que no son vectores soporte no toman parte en la solución final (ni en las predicciones).

5.2. Formulación del problema

Denotemos un conjunto de entrenamiento $\{x_i\}_{i=1}^N$, con clases $\{1, -1\}$ linealmente separable y recordemos que un hiperplano de separación está definido mediante

$$\{x \in \mathbb{R}^n | w^\top x + b = 0\}, \quad (5.1)$$

donde $w \in \mathbb{R}^n$ es el vector perpendicular al hiperplano y $b \in \mathbb{R}$ es el *offset*. De esta forma, si $w^\top x + b > 0$ se le asignará a x la clase 1, mientras que si $w^\top x + b < 0$ se le asignará a x la clase -1.

El problema de clasificación consiste en encontrar dichos parámetros de acuerdo a algún criterio, en este caso es el criterio del máximo margen. Notemos en primer lugar que este problema no tiene solución única, pues si (w, b) son solución entonces $(\lambda w, \lambda b)$, $\lambda > 0$ también lo son. Para evitar esta *invarianza* con tal de que el problema tenga solución única, podemos imponer una restricción sobre los bordes del margen. Denotando vectores soporte de cada clase mediante x_+ y x_- , se impone que para todo vector soporte x_+, x_- , estos pertenezcan a su respectiva clase, es decir:

$$w^\top x_+ + b = 1 \quad (5.2)$$

$$w^\top x_- + b = -1 \quad (5.3)$$

donde estos vectores soportes puede no ser únicos. Observemos que si bien aún no sabemos cuales son los vectores de soporte, podemos imponer esta restricción de todas formas, las que se van a traducir en las restricciones del problema de optimización que definirá la solución del problema de clasificación.

Además, es importante ver que las restricciones anteriores implican que $w \neq 0$, lo cual será importante más adelante cuando haya que justificar que el funcional a maximizar es estrictamente cóncavo.

Notemos que las ecs. (5.1),(5.2) y (5.3) definen tres hiperplanos paralelos, pues todos tienen el mismo parámetro w . La Fig. 23 ilustra las muestras de cada clase junto con estos tres hiperplanos, donde además se muestra el vector unitario perpendicular a (todos) estos hiperplanos dado por $\frac{w}{\|w\|}$. El ancho del margen, denotado por m , es la distancia entre la región de decisión y cualquiera de las clases, y es igual a la mitad de la diferencia entre ambos vectores de soporte, proyectada en la dirección normal del hiperplano, es decir,

$$m = \frac{1}{2} \|\text{proj}_w(x_+ - x_-)\| = \frac{1}{2} \|x_+ - x_-\| \cos(\theta) \quad (5.4)$$

$$= \frac{1}{2} \|x_+ - x_-\| \left(\frac{w^\top (x_+ - x_-)}{\|w\| \cdot \|x_+ - x_-\|} \right) \quad (5.5)$$

$$= \frac{1}{2\|w\|} w^\top (x_+ - x_-) \quad (5.6)$$

Donde se usó el hecho de que $\cos(\angle(x, y)) = \frac{\langle x, y \rangle}{\|x\|\|y\|}$.

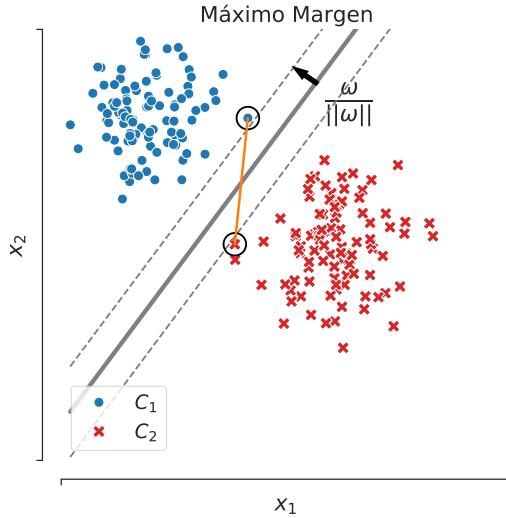


Fig. 23. Clasificación de máximo margen. Clases denotadas por puntos azules y cruces rojas, región de decisión en gris sólido y bordes del margen en línea punteada. La línea naranja representa la el vector diferencia $(x_+ - x_-)$ y la flecha negra el vector unitario $\frac{w}{\|w\|}$. Los vectores soporte están indicados con un círculo negro.

Veamos que el ancho del margen m no depende explícitamente de los vectores soporte al incorporar las restricciones en las ecs. (5.2)-(5.3):

$$\begin{aligned} m &= \frac{1}{2\|w\|} \left((w^\top x_+) - (w^\top x_-) \right) \\ &= \frac{1}{2\|w\|} ((1-b) - (-1-b)) \\ &= \frac{1}{\|w\|} \end{aligned} \quad (5.7)$$

Además, consideraremos la siguiente codificación para las clases:

$$y_i = +1 \Leftrightarrow w^\top x_i + b \geq +1 \quad (5.8)$$

$$y_i = -1 \Leftrightarrow w^\top x_i + b \leq -1 \quad (5.9)$$

En base a la expresión de la ec. (5.7) para el ancho del margen y la codificación de clases anterior, podemos formular el problema de clasificación de máximo margen mediante siguiente problema de optimización:

$$\begin{aligned} \max_{w,b} \quad & \frac{1}{\|w\|} \\ \text{s.a.} \quad & y_i(w^\top x_i + b) \geq 1, \quad i \in \{1, \dots, N\} \end{aligned} \quad (5.10)$$

Lo cual simplemente quiere decir que se está maximizando el ancho del margen, sujeto a que todas las muestras estén bien clasificadas. Es importante notar que este problema es factible solo si las clases son linealmente separable ya que restricción exige que todas las muestras queden bien clasificadas.

Para evitar problemas de diferenciabilidad del recíproco de la raíz cuadrada en el objetivo del problema de optimización anterior, sobretodo cuando w es cercano a 0, consideraremos la siguiente formulación equivalente del problema anterior:

$$\begin{aligned} (P) \quad \min_{w,b} \quad & \frac{1}{2} \|w\|^2 \\ \text{s.a.} \quad & y_i(w^\top x_i + b) \geq 1, \quad i \in \{1, \dots, N\} \end{aligned} \quad (5.11)$$

Este problema de optimización con restricciones puede ser resuelto mediante el método de Lagrange, donde resolvemos el *problema dual* (ver anexos), el cual tiene una estructura más “amigable” para resolver. Además, luego veremos que la resolución mediante este método es fundamental para la extensión no-lineal de SVM.

El lagrangiano del problema anterior está dado por:

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 + \sum_{i=1}^N \alpha_i (1 - y_i(w^\top x_i + b)) \quad (5.12)$$

donde hemos denotado los multiplicadores de Lagrange mediante $\alpha = (\alpha_1, \dots, \alpha_N)^\top$. El lagrangiano dual del problema viene dado por $\theta(\alpha) = \inf_{w,b} L(w, b, \alpha)$ y dado que L es convexo, basta aplicar la condición de primer orden:

$$\frac{\partial L}{\partial w} = w^\top - \sum_{i=1}^N \alpha_i y_i x_i^\top = 0 \implies \bar{w} = \sum_{i=1}^N \alpha_i y_i x_i \quad (5.13)$$

$$\frac{\partial L}{\partial b} = - \sum_{i=1}^N \alpha_i y_i = 0 \implies \sum_{i=1}^N \alpha_i y_i = 0 \quad (5.14)$$

Por lo tanto, el lagrangiano dual tiene la siguiente forma:

$$\theta(\alpha) = L(\bar{w}, \bar{b}, \alpha) = \frac{1}{2} \left\langle \sum_{i=1}^N \alpha_i y_i x_i, \sum_{j=1}^N \alpha_j y_j x_j \right\rangle + \sum_{i=1}^N \alpha_i - \sum_{i=1}^N \alpha_i y_i \left\langle \sum_{j=1}^N \alpha_j y_j x_j, x_i \right\rangle - b \sum_{i=1}^N \alpha_i y_i \quad (5.15)$$

$$= \frac{1}{2} \sum_{i,j=1}^N \alpha_i y_i \alpha_j y_j \langle x_i, x_j \rangle + \sum_{i=1}^N \alpha_i - \sum_{i,j=1}^N \alpha_i y_i \alpha_j y_j \langle x_j, x_i \rangle = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \quad (5.16)$$

Donde en la penúltima igualdad se usó el hecho de que $\sum_{i=1}^N \alpha_i y_i = 0$ de acuerdo a la segunda CPO sobre L .

Finalmente, el problema dual consiste en maximizar $\theta(\alpha)$ sujeto a que $\alpha \geq 0$, es decir:

$$(D) \quad \begin{aligned} & \underset{\alpha}{\max} && \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\ & \text{s.a.} && \sum_{i=1}^N \alpha_i y_i = 0 \\ & && \alpha_i \geq 0 \end{aligned} \tag{5.17}$$

La primera restricción se heredó de la CPO impuesta sobre L al calcular $\theta(\alpha)$. Este problema es del tipo QP (*quadratic programming*), para el cual existen variados métodos para resolverlo de manera óptima y eficiente. Por otra parte,

$$\sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle = \sum_{i,j=1}^N \alpha_i \langle y_i x_i, y_j x_j \rangle \alpha_j = \alpha^\top \Lambda \alpha, \tag{5.18}$$

Donde $\Lambda \in \mathbb{R}^{N \times N}$ corresponde a una matriz de Gram (en alguna base) por lo que es definida positiva ya que para $\alpha \neq 0$:

$$\alpha^\top \Lambda \alpha = \sum_{i,j=1}^N \langle \alpha_i y_i x_i, \alpha_j y_j x_j \rangle = \left\langle \sum_{i=1}^N \alpha_i y_i x_i, \sum_{j=1}^N \alpha_j y_j x_j \right\rangle = \left\| \sum_{i=1}^N \alpha_i y_i x_i \right\|^2 = \|\bar{w}\|^2 > 0 \tag{5.19}$$

Además, se tiene la siguiente propiedad:

Proposición 5.0.1. *A es definida positiva si y solo si $f(x) = x^\top A x + b^\top x + c$ es estrictamente convexa.*

Demostración. Basta verlo para la forma cuadrática $f(x) = x^\top A x$. Sea $\lambda \in (0, 1)$ y $x, y \in \mathbb{R}^n$ con $x \neq y$, entonces:

$$\begin{aligned} & (\lambda x + (1 - \lambda)y)^\top A (\lambda x + (1 - \lambda)y) < \lambda x^\top A x + (1 - \lambda)y^\top A y \\ \iff & \lambda^2 x^\top A x + (1 - \lambda)^2 y^\top A y + \lambda(1 - \lambda)x^\top A y + \lambda(1 - \lambda)y^\top A x < \lambda x^\top A x + (1 - \lambda)y^\top A y \\ \iff & \lambda(1 - \lambda)x^\top A x + \lambda(1 - \lambda)y^\top A y - \lambda(1 - \lambda)y^\top A y > 0 \\ \iff & x^\top A x + y^\top A y - x^\top A y - y^\top A x > 0 \\ \iff & (x - y)^\top A(x - y) > 0, \forall x, y \in \mathbb{R}^n : x \neq y \\ \iff & A \text{ es definida positiva} \end{aligned}$$

■

Luego, el funcional de SVM es estrictamente cóncavo, lo cual implica que el máximo del problema es único.

Una vez resuelta la formulación dual (i.e., se han encontrado los valores óptimos para α), la predicción de un nuevo punto x_* es de la forma

$$\hat{y}(x_*) = \text{sgn}(\bar{w}^\top x_* + b) = \text{sgn} \left(\left[\sum_{i=1}^N \alpha_i y_i \langle x_i, x_* \rangle \right] + b \right) \tag{5.20}$$

Finalmente, por el teorema de holgura complementaria, para α óptimo se tiene que

$$\alpha_i \left(1 - y_i (\bar{w}^\top x_i + b) \right) = 0, \quad \forall i \in \{1, \dots, N\} \implies \alpha_i = 0 \text{ para todo } x_i \text{ fuera del margen.} \tag{5.21}$$

y consecuentemente, x_i no aporta en la predicción \hat{y} . Esta propiedad es la que mencionábamos al principio: la predicción de clase solo depende de los vectores soporte, i.e., los que están en el margen. Esto ayuda a resolver el problema de optimización de manera más rápida, ya que en realidad solo algunas variables duales α_i serán no nulas (las correspondiente a los vectores que están en el borde del margen). Normalmente se ocupan heurísticas para encontrar y descartar rápidamente que vectores no son de soporte y así resolver un problema mas simple (ver, por ejemplo, el método de *Sequential Minimal Optimization*).

Observación 5.1. Notemos que las características $\{x_i\}$ solo aparecen en forma de productos internos entre ellas mismas en la solución del método SVM, lo cual se puede ver desde la formulación de problema de optimización dual en la ec. (5.17). Esto es clave para la extensión no lineal de SVM, pues no necesitamos operar directamente con los valores de las entradas o características, sino con los productos punto entre ellas. En particular, si tenemos N entradas de M dimensiones, solo necesitamos los $N(N+1)/2$ productos puntos entre ellos y no todos los NM valores, lo cual es particularmente relevante en caso que M sea muy grande o incluso cuando $M = \infty$ (e.g., cuando las características $\phi : x_i \mapsto \phi(x)$ son funciones).

5.3. Margen suave

El planteamiento anterior tiene dos debilidades: la primera es que nuestros datos no siempre serán separables, y la segunda es que, incluso si los datos son linealmente separables, nuestro clasificador puede ser muy sensible a nuevos datos. Esto es ilustrado en la Fig. 24, donde el clasificador de máximo margen sin *outlier* (izquierda) es muy distinto, o no-robusto, al obtenido en el caso de un *outlier* (encerrado en púrpura). En general, queremos que nuestro estimador sea robusto a este tipo de datos, tal como lo es la regresión lineal frente a observaciones ruidosas.

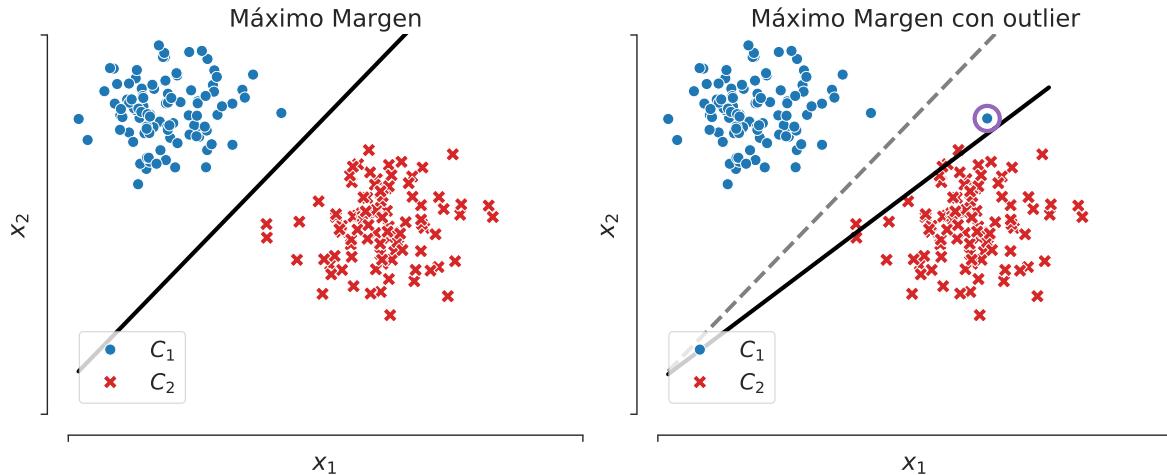


Fig. 24. Sensibilidad del máximo margen al agregar un nuevo dato: a la izquierda el caso sin *outlier* y a la derecha el caso con *outlier*, donde la solución anterior se muestra con la línea punteada.

Para considerar datos que son no posibles de clasificar con el clasificador lineal de máximo margen, podemos introducir las llamadas “variables de holgura” (*slack variables*). Estas tienen el objetivo de permitir al clasificador admitir algunos datos incorrectamente clasificados, aún maximizando el margen. Específicamente, esto se logra reemplazando las restricciones para los datos mal clasificados (considerados *outliers*) directamente en la formulación del problema de optimización, de esta forma, la *mayoría* de los datos se clasifican de forma correcta con la finalidad de tener un modelo robusto.

La formulación del problema de clasificación que *perdona* algunos datos mal clasificados mediante la utilización de variables de holgura $\{\xi_i\}_{i=1}^N$ es la siguiente:

$$(P) \quad \min_{w,b,\xi} \frac{1}{2} \|w\|^2 + c \sum_{i=1}^N \xi_i \quad (5.22)$$

s.a $y_i(w \cdot x_i + b) \geq 1 - \xi_i, \xi_i \geq 0, i \in \{1, \dots, N\}$

donde $c > 0$ es un hiperparámetro. Observemos que la introducción del término $c \sum_{i=1}^N \xi_i$ en la función de costo puede ser interpretada como una regularización, tal como lo hicimos con mínimos cuadrados. En efecto, de las restricciones del problema anterior, podemos ver que los ξ indican qué tan mal clasificado está un punto, donde:

- si $\xi_i = 0$, entonces el dato x_i está al lado correcto del plano y obtenemos el problema anterior
- si $0 < \xi_i < 1$, entonces x_i está al lado correcto del plano, pero está dentro del margen,
- si $\xi_i > 1$, entonces el punto esta al lado incorrecto del plano.

Estas tres condiciones son ilustradas en la Fig. 25.

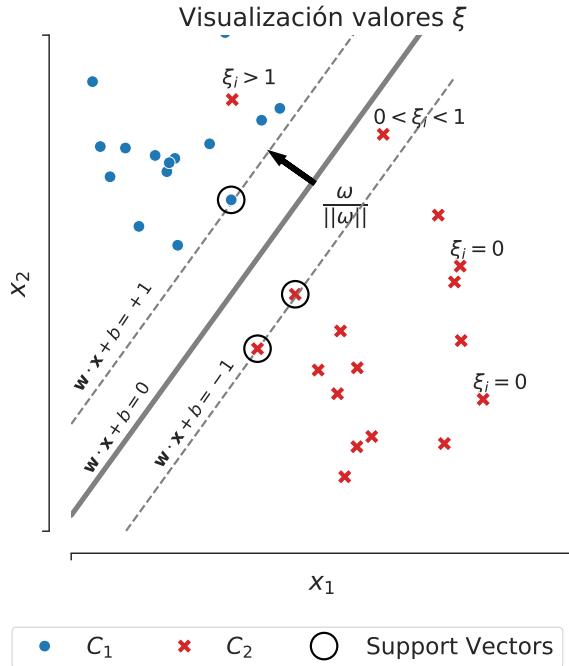


Fig. 25. Valores para las variables de holgura ξ en distintas regiones del espacio de entrada.

Procediendo de la misma forma que en el caso anterior, el dual de este problema es:

$$(D) \quad \max_{\alpha} \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j=1}^N \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \quad (5.23)$$

s.a $\sum_{i=1}^N \alpha_i y_i = 0$

$0 \leq \alpha_i \leq c$

Esta solución similar al caso anterior y también se puede resolver con técnicas de programación cuadrática, en particular, su solución también existe y es única. La diferencia entre ambas soluciones (con y sin

variables de holgura) está dada por el hecho de que los multiplicadores de Lagrange ahora están acotados por el hiperparámetro c , el que representa la importancia que se da a la suma de las variables de holgura versus el ancho del margen en la función de costo del problema de optimización.

Por último, ¿cómo definir el parámetro c ? Es posible responder esta pregunta en relación al *bias-variance tradeoff*. Notemos que $\xi_i > 1$ si la muestra x_i está mal clasificada, entonces el término $\sum_{i=1}^N \xi_i$ es una cota superior para la cantidad de muestras mal clasificadas. Consecuentemente, el hiperparámetro c es un coeficiente (*inverso*) de regularización, pues su magnitud controla el balance entre la maximización del margen y la cantidad muestras mal clasificadas. En particular, para un c muy alto la cantidad de muestras mal clasificadas tiene mucho peso relativo comparada contra el ancho del margen en el funcional de minimización, con lo que la solución óptima tendrá un margen muy angosto y pocas (si es que hay alguna) muestras mal clasificadas. En efecto, si $c \rightarrow \infty$, se recupera la formulación del *margen duro* y su misma solución de máximo margen (en el caso de que el problema sea efectivamente linealmente separable), pues solo se podrá resolver el problema si todas las variables de holgura son nulas. Por el contrario, para un c pequeño el margen tiene más importancia que la cantidad de datos incorrectamente clasificados, con lo que se encontrará un margen amplio con varias muestras mal clasificadas. Es de esperar que un c pequeño (margen amplio) tenga mejor capacidad de generalizar a nuevos datos con menor varianza, mientras que un c grande puede sobreajustar a los datos disponibles, reportando peor desempeño *out-of-sample*. Por esto hemos dicho que c es análogo a un coeficiente *inverso* de regularización, pues mientras más pequeño, más regular es la solución.

Al no conocer los estadísticos de los datos, en la práctica ajustamos el hiperparámetro c mediante validación cruzada.

5.4. Método de kernel

El clasificador SVM (tanto con margen duro o blando) tiene una propiedad muy beneficiosa en el caso linealmente separable: la solución siempre existe, es única y se puede encontrar. Sin embargo, no tenemos ninguna garantía en los casos no separables, los cuales afloran usualmente en la práctica. Consideremos el problema de clasificar una base de datos generadas con una función “o exclusivo”, o “XOR”, presentada en la Figura 26. Estos datos no son linealmente separables, con lo que al calcular el clasificador óptimo de SVM no llegaríamos a ninguna solución coherente ya que el problema es infactible, sin embargo, notemos que es posible diseñar una característica particular para este conjunto donde sí son separables.

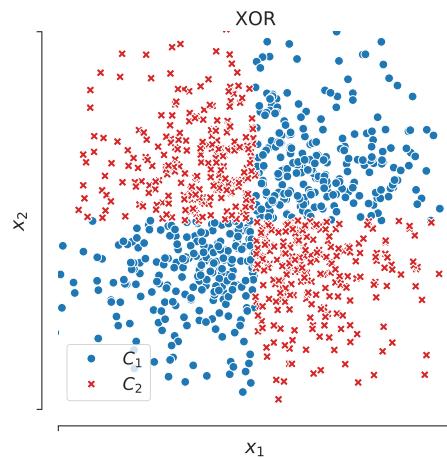


Fig. 26. Datos XOR: no son linealmente separables.

En efecto, consideremos el mapa desde \mathbb{R}^2 a \mathbb{R}^3 definido mediante

$$\phi : [x_1, x_2]^\top \mapsto [x_1, x_2, x_1 x_2]^\top \quad (5.24)$$

y observemos que este permite clasificar de forma lineal (y trivial) las clases del problema XOR: ambas quedan clasificadas mediante el plano $z = 0$ en \mathbb{R}^3 , esto es ilustrado en la Fig. 27. La función ϕ es un ejemplo de una ingeniería de características como las vistas en el capítulo de regresión no lineal, en este caso particular, la característica relevante era precisamente $x_1 x_2$.

Proyección XOR (incompleta!!)

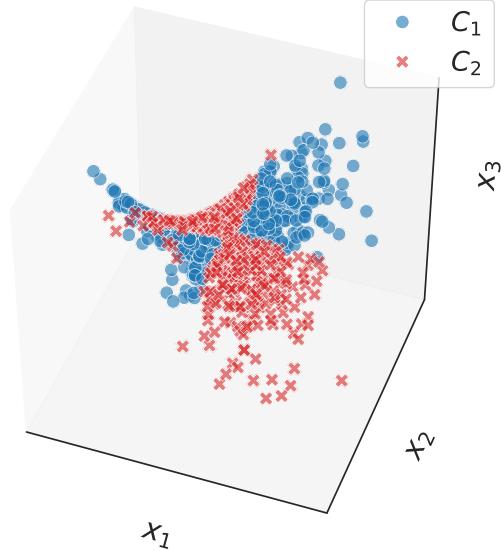


Fig. 27. Los puntos rojos y azules corresponden a los datos mapeados a través de ϕ . El plano $z = 0$ es capaz de separar puntos rojos y puntos azules en este nuevo espacio.

Observación 5.2. En el caso del problema XOR conocíamos la característica apropiada, sin embargo, en el caso general no es claro cuál debe ser “el buen ϕ ”. A pesar de esto, notemos que tanto para la formulación con margen duro o blando de SVM, la solución solo requerirá que seamos capaces de calcular los productos internos entre las características de cada entrada. Es decir, si consideramos un ϕ arbitrario para nuestro problema de clasificación, solo necesitaríamos calcular los productos de la forma

$$\langle \phi(x_i), \phi(x_j) \rangle \quad (5.25)$$

para todo x_i, x_j en el conjunto de entrenamiento.

Podemos entonces aplicar la siguiente intuición: seguramente no podemos encontrar el ϕ exclusivo de cada problema, sino que podemos utilizar uno (o una clase) que sea muy general, complejo, o “universal”. Esperamos que esta característica general nos sirva para una gran cantidad de problemas, donde tenemos la garantía de que lo podremos usar como parte de SVM si somos capaces de calcular su producto interno. Además, por “general” podemos entender “de alta dimensión”: podríamos pensar que para un problema en el cual no sabemos el buen ϕ , podemos considerar incorporar e incorporar características, con lo cual tendríamos una “característica agregada” de alta dimensión, esperando que alguno de los términos agregados sea el que efectivamente separa las clases. Si bien no está garantizada la separabilidad lineal al combinar características, al agregar dimensiones los datos están cada vez más separados debido a la maldición de la dimensionalidad y se puede probar que en dimensión infinita, siempre se podrá separar linealmente.

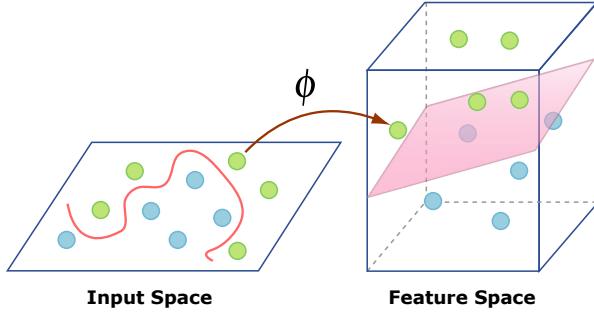


Fig. 28. Separación lineal en un espacio de dimensión mayor. Imagen obtenida de MIT OpenCourseWare (MIT 15.097 course).

Para encontrar estos ϕ generales, veamos la siguiente definición

Definición 5.1 (Mercer kernel). Un Mercer kernel es una función continua $K : X \times X \rightarrow \mathbb{R}$ tal que

- Es simétrica $K(x_1, x_2) = K(x_2, x_1)$
- Es definida positiva, es decir

$$\int_{X^2} K(x_1, x_2)g(x_1)g(x_2)dx_1dx_2 \geq 0,$$

para toda función $g : X \rightarrow \mathbb{R}$ continua. El nombre de esta propiedad derivada de su similitud con las matrices definidas positivas: si g se mira como vector de \mathbb{R}^X , entonces la expresión anterior representa a $g^\top K g \leq 0$.

De esta forma, se tiene el siguiente teorema de análisis funcional que sustenta la utilización de kernels en los algoritmos de aprendizaje automático:

Teorema 5.1 (teorema de Mercer (simplificado)). *Sea $K : X \times X \rightarrow \mathbb{R}$ un Mercer kernel, entonces existe un espacio de Hilbert $(\mathcal{H}, \langle \cdot, \cdot \rangle)$ y una función $\phi : X \rightarrow \mathcal{H}$ tal que:*

$$K(x_1, x_2) = \langle \phi(x_1), \phi(x_2) \rangle \quad (5.26)$$

Es decir, existe un mapa de características ϕ tal que $K(x_1, x_2)$ representa el producto interno (en algún espacio) de las características de x_1 y x_2 . Además, dicho espacio no es necesariamente de dimensión finita.

Observación 5.3 (truco del kernel). La introducción del concepto de kernel es fundamental en SVM. La definición y el teorema anterior nos dicen que para cualquier función simétrica y definida positiva K , existe una función ϕ , que puede ser incluso de dimensión infinita, tal que la evaluación del kernel en dos puntos cualquiera equivale al producto interno entre dos evaluaciones de ϕ en los mismos puntos, es decir, $K(x_1, x_2)$ representa un producto interno en algún espacio de características. Esto, sumado al hecho de que la solución de SVM solo requiere del cálculo de productos internos, nos permite construir *kernel SVMs*, donde parametrizamos directamente el producto interno en el problema de optimización mediante el kernel, pues esto da la garantía que el mapa de características ϕ existe. El caso interesante es cuando ocupamos un kernel que corresponde a un ϕ infinito dimensional, pues estamos efectivamente realizando la clasificación en un espacio de dimensión infinita pero con un procedimiento que solo requiere de una cantidad finita de cálculos, esto se llama *el truco del kernel*. Este truco puede aplicarse a cualquier algoritmo en donde las entradas solo aparezcan en la forma de productos punto, proceso que recibe el nombre de *kernelización*.

Veamos distintos tipos de kernels y sus propiedades.

- **Kernel polinomial:**

$$K_{pol}(x, y) = (c + x^\top y)^d \quad (5.27)$$

donde $c \geq 0$ es un parámetro libre y $d \in \mathbb{N}$ es el orden del polinomio. Para probar que dicha función es un kernel, basta reagrupar los términos buscando formar un producto interno. Para $x, y \in \mathbb{R}^m$, $d = 2$ se tiene que:

$$K_{pol}(x, y) = \left(c + \sum_{i=1}^m x_i y_i \right)^2 \quad (5.28)$$

$$= \sum_{i=1}^m (x_i^2)(y_i^2) + \sum_{i=2}^m \sum_{j=1}^{i-1} (\sqrt{2}x_i x_j)(\sqrt{2}y_i y_j) + \sum_{i=1}^m (\sqrt{2c}x_i)(\sqrt{2c}y_i) + c^2 \quad (5.29)$$

$$= \langle \phi_{pol}(x), \phi_{pol}(y) \rangle \quad (5.30)$$

donde

$$\phi_{pol}(x) = [x_1^2, \dots, x_m^2, \sqrt{2}x_1 x_2, \dots, \sqrt{2}x_m x_{m-1}, \sqrt{2c}x_1, \dots, \sqrt{2c}x_m, c]. \quad (5.31)$$

Es decir, al usar el kernel poliomial estamos implícitamente usando un mapa de características que contiene todos los monomios de grado hasta $d = 2$ (si $c > 0$) o bien todos los monomios de grado igual a $d = 2$ (en caso que $c = 0$). Más allá de esta ilustración, esto se cumple para cualquier $d \in \mathbb{N}$.

- **Función de base radial (RBF kernel)¹¹:** Este kernel está definido por

$$K_{RBF}(x, y) = \sigma^2 \exp \left(-\frac{\|x - y\|^2}{2l^2} \right). \quad (5.32)$$

El mapa de características que induce es de dimensión infinita y las fronteras que entrega son suaves (infinitamente diferenciables). Los hiperparámetros del kernel son σ^2 (controla la distancia promedio de la función con su media) y l (controla la oscilación de la curva).

- **Kernel periódico:**

$$K_{per}(x, y) = \sigma^2 \exp \left(-\frac{2 \operatorname{sen}^2 \left(\frac{\pi|x-y|}{p} \right)}{l^2} \right). \quad (5.33)$$

Este kernel es capaz de rescatar características periódicas en los datos (controlados por el parámetro p). Los otros parámetros cumplen la misma función que el kernel anterior.

Por último, el conjunto de kernels es cerrado bajo sumas y multiplicaciones, lo cual permite construir kernels más complejos combinando otros más simples. Esto será de gran utilidad en procesos gaussianos.

5.4.1. Kernel ridge regression

Veamos que es directo kernelizar el método de mínimos cuadrados regularizados visto en la Sección 2.1.2. En particular, consideremos el caso de regularización cuadrática, el cual, vimos que tiene solución

$$\theta_{MCR} = \left(\tilde{X}^\top \tilde{X} + \rho \mathbb{I} \right)^{-1} \tilde{X}^\top Y \quad (5.34)$$

y consecuentemente, reporta una predicción en base a un nuevo input x_\star dada por $\hat{y}_\star = \theta_{MCR}^\top x_\star$. Como podemos ver, la interacción de las entradas $(\tilde{x}_i)_{i=1}^N$ no aparece en la forma de productos internos (lo cual

¹¹también conocido como kernel exponencial cuadrático o gaussiano.

es necesario para la kernelización) ya que $(\tilde{X}^\top \tilde{X})_{ij} = \langle \tilde{X}_{i\cdot}, \tilde{X}_{j\cdot} \rangle = \langle \tilde{X}_{i\cdot}, \tilde{X}_{i\cdot} \rangle$, es decir, es el producto interno de las columnas de \tilde{X} y no de los datos (recordar que $\tilde{x}_i = \tilde{X}_{i\cdot}$). Sin embargo, se tiene la siguiente propiedad de inversión:

Proposición 5.1.1. *La solución de mínimos cuadrados es equivalente a*

$$\theta_{MCR} = \tilde{X}^\top (\tilde{X}\tilde{X}^\top + \rho\mathbb{I})^{-1} Y \quad (5.35)$$

Demostración. Usando la fórmula de Woodburry (ver anexos)

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \quad (5.36)$$

considerando $A = \rho\mathbb{I}$, $U = \tilde{X}^\top$, $C = \mathbb{I}$ y $V = \tilde{X}$, se tiene que:

$$\theta_{MCR} = \left(\frac{1}{\rho}\mathbb{I} - \frac{1}{\rho}\tilde{X}^\top \left(\mathbb{I} + \frac{1}{\rho}\tilde{X}\tilde{X}^\top \right)^{-1} \frac{\tilde{X}}{\rho} \right) \tilde{X}^\top Y = \frac{1}{\rho}\tilde{X}^\top \left(\mathbb{I} - \left(\mathbb{I} + \frac{1}{\rho}\tilde{X}\tilde{X}^\top \right)^{-1} \frac{\tilde{X}\tilde{X}^\top}{\rho} \right) Y \quad (5.37)$$

$$= \frac{1}{\rho}\tilde{X}^\top \left(\mathbb{I} - \left(\mathbb{I} + \frac{1}{\rho}\tilde{X}\tilde{X}^\top \right)^{-1} \left(\mathbb{I} + \frac{1}{\rho}\tilde{X}\tilde{X}^\top \right) + \left(\mathbb{I} + \frac{1}{\rho}\tilde{X}\tilde{X}^\top \right)^{-1} \right) Y \quad (5.38)$$

$$= \frac{1}{\rho}\tilde{X}^\top \left(\mathbb{I} + \frac{\tilde{X}\tilde{X}^\top}{\rho} \right)^{-1} Y = \tilde{X}^\top (\tilde{X}\tilde{X}^\top + \rho\mathbb{I})^{-1} Y \quad (5.39)$$

■

Donde ahora $\tilde{X}\tilde{X}^\top$ sí corresponde a un producto externo de las entradas:

$$(\tilde{X}\tilde{X}^\top)_{ij} = \langle \tilde{X}_{i\cdot}, \tilde{X}_{j\cdot} \rangle = \langle \tilde{X}_{i\cdot}, \tilde{X}_{i\cdot} \rangle = \langle \tilde{x}_i, \tilde{x}_j \rangle \quad (5.40)$$

Además,

$$\hat{y}_\star = \theta_{MCR}^\top x_\star = Y^\top (\tilde{X}\tilde{X}^\top + \rho\mathbb{I})^{-1} \tilde{X} \tilde{x}_\star \quad (5.41)$$

donde $(\tilde{X}x_\star)_i = \langle \tilde{x}_i, x_\star \rangle$, lo cual muestra que las entradas solo aparecen en la predicción en forma de productos internos.

Observación 5.4 (Costo computacional: dimensión v/s cantidad de datos). Como $\tilde{X} \in \mathbb{R}^{N \times M}$, donde N es la cantidad de datos y M es la dimensión de los datos de entrada, entonces la matriz a invertir en la ec. (5.34) es de tamaño $M \times M$, mientras que la matriz a invertir en la ec. (5.35) es de tamaño $N \times N$. Como el costo de invertir una matriz es de orden cúbico (dependiendo del método) en su dimensión, será preferible utilizar una de las dos formulaciones en base a qué es mayor, la dimensión o el número de datos.

Por otro lado, sea ϕ un mapa de características (i.e., una función que recupera las características de una entrada x), denotando las características por $\phi_i = \phi(x_i)$ y $\phi_\star = \phi(x_\star)$, se puede hacer la regresión sobre las características:

$$\Theta = \begin{pmatrix} \phi_1^\top \\ \vdots \\ \phi_N^\top \end{pmatrix} \implies \hat{y}_\star = Y^\top (\Theta\Theta^\top + \rho\mathbb{I})^{-1} \Theta \phi_\star \quad (5.42)$$

Luego, si K es un kernel asociado al mapa de características ϕ , es decir $K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$, se tiene que

$$\hat{y}_\star = Y^\top \left(\Theta \Theta^\top + \rho \mathbb{I} \right)^{-1} \Theta \phi_\star = Y^\top \left(K(\tilde{X}, \tilde{X}) + \rho \mathbb{I} \right)^{-1} K(\tilde{X}, x_\star), \quad (5.43)$$

donde se ha hecho abuso de notación al usar argumentos matriciales en el kernel:

$$K(\tilde{X}, \tilde{X})_{ij} = (\Theta \Theta^\top)_{ij} = \langle \phi_i, \phi_j \rangle = K(x_i, x_j) \quad K(\tilde{X}, x_\star)_i = (\Theta \phi_\star)_i = \langle \phi_i, \phi_\star \rangle = K(x_i, x_\star) \quad (5.44)$$

Alternativamente, esta predicción puede ser reordenada para dar

$$\hat{y}_\star = \sum_{i=1}^N h_i K(x_i, x_\star), \quad (5.45)$$

donde hemos denotado el vector $h \in \mathbb{R}^N$ de la forma $h_i = \left(Y^\top \left(K(\tilde{X}, \tilde{X}) + \rho \mathbb{I} \right)^{-1} \right)_i$. Hay varias observaciones relevantes que podemos hacer con respecto al resultado anterior.

Observación 5.5 (Infinitas características). Como hemos descrito la predicción de MCR en forma de productos internos, podemos reemplazar las entradas x en la descripción anterior por una característica $\phi(x)$, de dimensión D . Luego, si el kernel K tiene asociado un mapa de dimensión infinita, la regresión se hará sobre dicho mapa, buscando separar la entrada en un espacio de dimensión infinita.

Observación 5.6 (Kernel como similitud). Notemos de la expresión anterior que si el kernel $K(x, x')$ es una medida de similitud entre x y x' , entonces la predicción de y_\star es una combinación lineal de h_i , donde los ponderadores de cada h_i son mayores para los datos x_i que se parecen (con respecto a K) a la nueva entrada x_\star . A su vez, los valores h_i son una transformación lineal de las entradas Y , con lo que la predicción de *kernel ridge regression* combina linealmente salidas conocidas en base a cuán similar a los datos históricos es una nueva entrada.

Observación 5.7 (Modelo no paramétrico). Como vimos anteriormente, el hecho de que la predicción en la ec. (5.45) solo dependa del kernel permite evitar definir explícitamente el mapa de características ϕ , sino que solo necesitamos el kernel K . Esto permite utilizar un kernel (e.g., exponencial cuadrático) que corresponde a un ϕ de dimensión infinita, lo que resulta en el parámetro θ siendo también de dimensión infinita. A pesar de la característica *infinita* de este método, solo necesitamos hacer un cálculo finito para representar \hat{y}_\star , esto se puede interpretar como la extracción de la información suministrada por los datos que tenemos $\mathcal{D} = \{x_1, \dots, x_n\}$, los cuales son finitos a pesar de que el modelo tenga una “capacidad” infinita (ver el Teorema del representante). Una consecuencia de esto es que la predicción en la ec. (5.45) depende de todos los elementos de \mathcal{D} , a diferencia de todos los modelos que hemos visto hasta ahora, donde la información de los datos (independiente de cuántos fueran) quedaba resumida en un parámetro de dimensión fija. Nos referimos a este tipo de modelo (con infinitos parámetros) como *no paramétricos* debido a que su tratamiento no puede ser de la misma forma que los modelos con parámetros finitos.

Observación 5.8 (Complejidad variable). Finalmente, el hecho que los modelos no paramétricos tengan una cantidad de términos creciente en la cantidad de observaciones implica que su complejidad y desempeño son crecientes también. Esto está de acuerdo con la intuición de que un modelo no debería tener una complejidad o capacidad fija, sino que flexible a medida que se va considerado una mayor evidencia.

5.4.2. Kernel SVM

Volvamos ahora a SVM para convertir un problema no linealmente separable en uno linealmente separable (en algún espacio) mediante una *kernelización*, es decir, reemplazando las entradas x por un mapa de características $\phi(\cdot)$ de alta (o infinita) dimensión. Como vimos anteriormente, si la formulación de SVM está expresada en forma de productos internos (y sí que lo está), a través del truco del kernel

podemos parametrizar directamente dichos productos internos con un kernel $K(\cdot, \cdot)$ sin la necesidad de definir explícitamente el mapa ϕ .

Reemplazando entonces las entadas por las características igual que en el caso de regresión de ridge, la kernelización del SVM con margen suave tiene una formulación primal dada por

$$(P) \quad \begin{aligned} & \min_{w,b} \quad \frac{1}{2} \|w\|^2 + c \sum_{i=1}^N \xi_i \\ & \text{s.a.} \quad y_i(w^\top \phi(x_i) + b) \geq 1 - \xi_i, \quad i \in \{1, \dots, N\} \end{aligned} \tag{5.46}$$

Mientras que su formulación dual tiene la forma

$$(D) \quad \begin{aligned} & \max_{\alpha} \quad \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \langle \phi(x_i), \phi(x_j) \rangle \\ & \text{s.a.} \quad \sum_{i=1}^N \alpha_i y_i = 0 \\ & \quad 0 \leq \alpha_i \leq c \end{aligned} \tag{5.47}$$

Podemos ocupar el truco del kernel para parametrizar directamente el producto interno $\langle \phi(x_i), \phi(x_j) \rangle$ mediante $K(x_i, x_j)$. Con esto, el problema de optimización en el dual se convierte en

$$\begin{aligned} & \max_{\alpha} \quad \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ & \text{s.a.} \quad \sum_{i=1}^N \alpha_i y_i = 0 \\ & \quad 0 \leq \alpha_i \leq c \end{aligned} \tag{5.48}$$

Observación 5.9 (Kernel SVM). Esta formulación, al igual que el SVM lineal original, es un QP con solución única, pues el kernel K es definido positivo y por ende el funcional de optimización es cuadrático y cóncavo. Consecuentemente, una vez calculadas las $N(N+1)/2$ evaluaciones de la forma $K(x_i, x_j)$, lo único que queda es resolver un problema QP. Es particularmente relevante ver que kernel SVM tiene el mismo orden de complejidad computacional que el SVM lineal.

La Fig. 29 muestra la implementación de kernel SVM para dos kernels: a la izquierda se ocupó un kernel polinomial de grado 3, dicho kernel se caracteriza por su curvatura en los límites del clasificador, mientras que a la derecha se ocupó un kernel RBF, el cual es capaz de encontrar fronteras de decisión más curvadas (infinitamente diferenciables).

Finalmente, observemos que los métodos de kernel tienen una tremenda ventaja computacional con respecto a sus contrapartes sin kernel. Transformar los datos a un espacio de mayor dimensión definiendo explícitamente $\phi(x)$ puede ser muy costoso, o incluso imposible, en el caso que el espacio de llegada sea infinito-dimensional. Usando kernels, por el contrario, dicha función queda implícita en el problema, y el método tiene un costo finito a pesar de que el modelo efectivamente opera en un espacio de características infinito (en el caso de RBF por ejemplo).

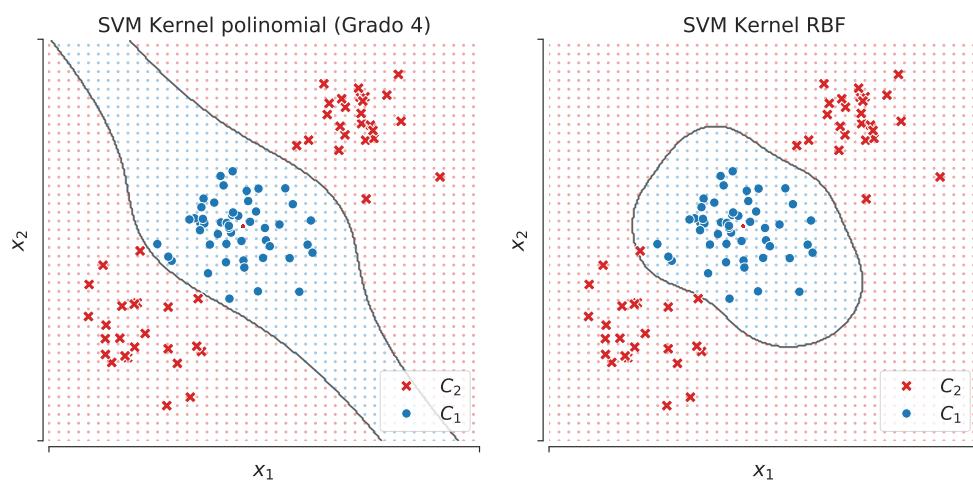


Fig. 29. Clasificación usando kernel SVM (margen suave) con distinto kernels: polinomial a la izquierda y RBF a la derecha.

6. Modelos de función de base adaptativa

6.1. Introducción

Esta sección se refiere a una familia general de modelos que llamaremos modelos de **función de base adaptativa**, que tienen la forma:

$$f(x) = w_0 + \sum_{k=1}^K w_k \phi_k(x). \quad (6.1)$$

A la función $\phi_k(x)$ se le dice la m -ésima función de base, la cual variará en función de los datos. Esta familia general de modelos incluye los modelos a estudiar en esta sección: árboles, bosques, modelos basados en bagging y boosting, como también las redes neuronales y las sumas generales de modelos (Murphy, 2022).

6.2. Árboles

6.2.1. Motivación con caso regresión

Antes de definir árboles de manera formal, construiremos una intuición para el caso de regresión. Consideremos una función de una variable. Una manera de aproximar tal función es hacer una interpolación usando funciones constantes. Una primera idea puede ser aproximar la función entera simplemente por su media.

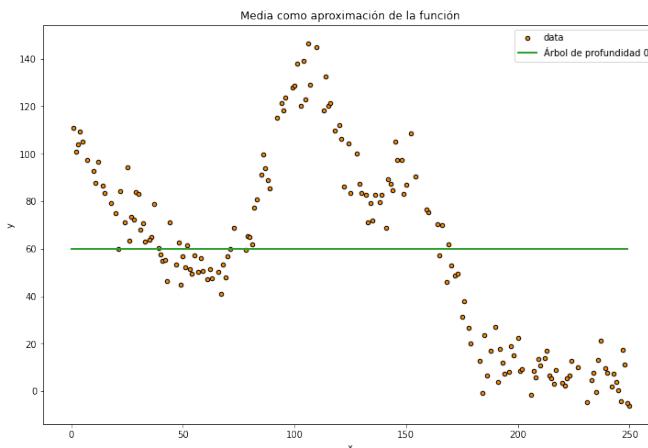


Fig. 30. Aproximación de una función usando su media (árbol trivial no ajustado).

Otra manera, más bien voraz, de enfrentar el problema es realizar la interpolación con todos los puntos. Esto resultará con seguridad en un sobreajuste de los datos.

Una manera más inteligente consiste en agrupar ciertos puntos y hacer una interpolación usando la media de estos. El desafío es encontrar una partición conveniente del dominio de los datos de modo que al predecir un punto de nuestra función, tomar la media de los datos de entrenamiento para el subconjunto correspondiente resulte en una buena aproximación.

6.2.2. Algoritmo *CART*

Los árboles (de regresión) corresponden a ejecutar lo anterior de manera recursiva. Primero, necesitamos una algún criterio que nos indique si realizar un corte vale o no la pena. Para esto podemos definir el costo para un conjunto D de datos como sigue:

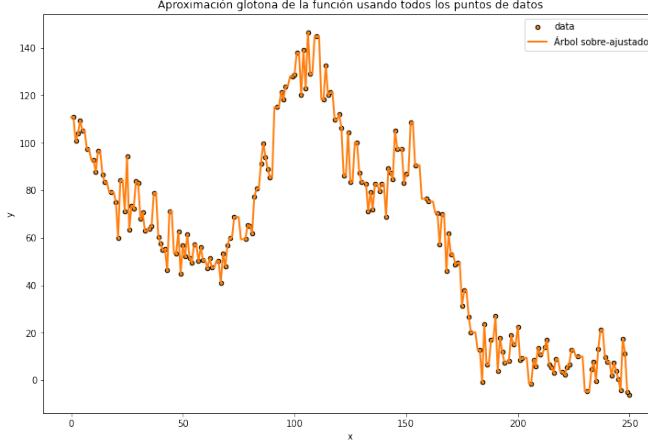


Fig. 31. Aproximación de una función usando todos los puntos de entrenamiento (árbol sobreajustado).

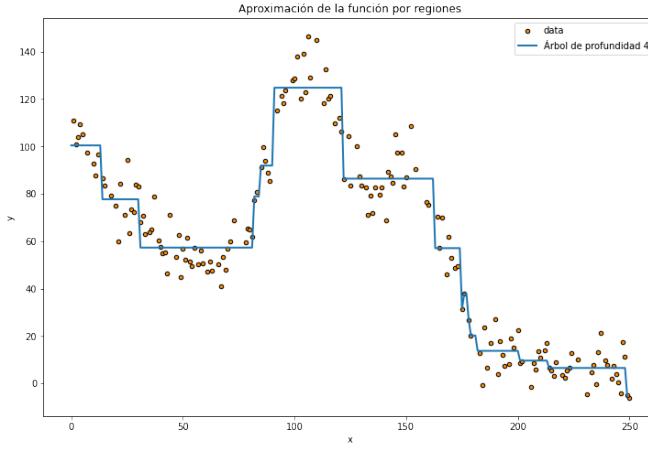


Fig. 32. Aproximación de una función con un árbol de profundidad adecuada.

$$cost(D) = \sum_{i \in D} (y_i - \bar{y})^2, \quad (6.2)$$

con $\bar{y} = \frac{1}{|D|} \sum_{i \in D} y_i$. Notemos como este costo es proporcional a la varianza empírica del conjunto D , con lo cual pedir bajo costo en los grupos de una partición se traduce en pedir que los datos estén cercanos a la media de tal subconjunto. Con esto podemos armar un algoritmo de base recursivo que genere un árbol de regresión.

Algoritmo 1 Ajuste de árboles (CART)

```

1: function AJUSTEARBOLE(nodo, D, profundidad)
2:    $j^*, t^* = \arg \min_{j \in \{1, \dots, m\}, t \in \Gamma_j} [cost(D_L(j, t)) + cost(D_R(j, t))]$ 
    $D_L(j, t) = \{(x^{(i)}, y^{(i)}) : x_j^{(i)} \leq t\}, D_R(j, t) = \{(x^{(i)}, y^{(i)}) : x_j^{(i)} < t\}$ 
3:   if criterio_de_parada(costo, profundidad, DI, DD) then: return nodo
4:   else
5:     nodo.izquierda = ajusteArbol(nodo, DI, profundidad + 1)
6:     nodo.derecha = ajusteArbol(nodo, DR, profundidad + 1)
return nodo

```

Nótese que esto no necesariamente encontrará el árbol binario óptimo. Se prefiere este método voraz pues ajustar un árbol binario óptimo es un problema NP completo. En particular notemos como vamos separando coordenada por coordenada, lo cual nos hace ganar en interpretabilidad. Por otro lado, el criterio de parada lo discutiremos más adelante.

Siguiendo el algoritmo anterior obtendremos una partición. Podemos enumerar los nodos de 1 hasta K, con lo cual recuperamos la forma de base adaptativa:

$$f(x) = \mathbb{E}[y|x] = \sum_{k=1}^K w_k \phi_k(x), \quad (6.3)$$

con $\phi_k(x) = \mathbf{1}_{D_k}(x)$ y $w_k = \frac{1}{|D_k|} \sum_{x^{(i)} \in D_k} y^{(i)}$. Como se señaló en la intuición, escogeremos la media de los datos en el subconjunto como predicción. Esto se justifica pues aquel valor es el que minimiza el error cuadrático. Podríamos también aprender un modelo simple (por ejemplo un regresor de mínimos cuadrados) de manera local en cada partición, sin embargo como hemos escogido la partición de manera que minimice la varianza, es razonable pensar que la media será una aproximación suficientemente buena.

6.2.3. Criterios de corte para clasificación

En el algoritmo CART hicimos uso de la función costo, que nos mostraba cuánto variaban los miembros de un intervalo respecto de la media. Podemos generalizar este mismo principio para clasificación, donde intentaremos caracterizar la impureza de etiquetas para un conjunto de manera adecuada. Primero tomemos el vector de probabilidades de pertenencia a una clase, condicionado a estar en un nodo. Sea \mathcal{C} el conjunto de clases,

$$\hat{\pi}_c(D) = \frac{1}{|D|} \sum_{x^{(i)} \in D} \mathbf{1}_{y=c}(y^{(i)}) \quad , \forall c \in \mathcal{C}. \quad (6.4)$$

Usando esto, predecir la probabilidad de que un punto pertenezca a cada clase estará dado por el vector de fracciones empírica $\hat{\pi}$ correspondiente al nodo al cual pertenezca el dato en cuestión. Usemos este mismo vector para definir los criterios de impureza por nodo (ignoraremos la dependencia de D en el vector de probabilidades para simplificar la notación) (Breiman y cols., 1984).

- **Tasa de error**

Sea $\hat{y} = \arg \max_{c \in \mathcal{C}} \hat{\pi}_c$ la clase más probable. El error estará dado por

$$cost(D) = \frac{1}{|D|} \sum_{x^{(i)} \in D} \mathbf{1}_{y=\hat{y}}(y^{(i)}) = 1 - \hat{\pi}_{\hat{y}} \quad (6.5)$$

El problema de este criterio es su poca sensibilidad a cambios en el vector de probabilidad. Los siguientes dos criterios mejoran esta situación.

- **Gini**

Corresponde a la tasa de error esperado:

$$cost(D) = \sum_{c \in \mathcal{C}} \hat{\pi}_c (1 - \hat{\pi}_c) = 1 - \sum_{c \in \mathcal{C}} \hat{\pi}_c^2 \quad (6.6)$$

- **Entropía**

También llamada log-pérdida y muchas veces denotada por $H(\hat{\pi})$, esta métrica está dada por:

$$cost(D) = - \sum_{c \in \mathcal{C}} \hat{\pi}_c \log(\hat{\pi}_c). \quad (6.7)$$

Esta elección de perdida tiene justificación en la Teoría de la Información. En particular, su uso como criterio de corte equivale a la minimización de la entropía cruzada.

Consideremos el ejemplo de clasificación binaria. Notemos que para las tres el máximo está cuando tenemos 50/50 de datos para cada clase en el nodo en cuestión, que es justamente el caso de mayor heterogeneidad de los datos. Por el contrario, los valores son cero cuando el conjunto tiene miembros de una sola clase. La sensibilidad antes mencionada se desprende de esto. En el caso de clases impuras, el error de clasificación está siempre por debajo de los otros criterios, que castigan más fuertemente la impureza.

6.2.4. Evitar sobreajuste: detención temprana y poda

Una primera estrategia para evitar el sobreajuste del árbol es la **detención temprana**, i.e., evitar que se siga particionando el espacio. A continuación mencionaremos algunos de los varios criterios. Estos se aplican como un criterio a verificar antes de seguir partiendo nodos en el algoritmo 1 (CART):

- Máxima profundidad: se puede imponer una profundidad máxima, de modo que una vez alcanzada esta no se separe más el nodo en cuestión.
- Número mínimo en nodo: si un nodo contiene muy pocos datos entonces partirlo puede resultar en nodos con muy pocos datos (a este fenómeno se le llama fragmentación de datos). Podemos considerar o bien un entero o bien un porcentaje mínimo de los datos totales.
- Mínima reducción de costos: Dados los lados izquierdos y derechos de un corte, consideremos la reducción en costo como:

$$\Delta = \text{cost}(D) - \left[\frac{|D_L|}{|D|} \text{cost}(D_L) + \frac{|D_R|}{|D|} \text{cost}(D_R) \right] \quad (6.8)$$

Esta métrica de reducción de costos normalizada nos permite definir un criterio de parada donde no cortaremos el nodo si los nodos resultantes no inducen una reducción significativa.

Los criterios anteriores inducen hiperparámetros que pueden ser escogidos en una búsqueda de grilla.

La **poda** es otra estrategia para evitar el sobreajuste y disminuir la complejidad del árbol. Esta consiste en ajustar un árbol uno (posiblemente con criterios de parada anteriormente expuestos) y luego podarlo, que corresponde a elegir un sub-árbol (i.e., “podar” nodos). El conjunto de todos los subárboles de un árbol de decisión es potencialmente muy elevado. Es por esto que se elegirá un conjunto adecuado de subárboles, de modo que sea razonable comparar su rendimiento para elegir la mejor opción.

Primero consideraremos una métrica $R(T)$ que nos dé una noción de costo para un árbol T . Típicamente se usará la suma de las tasas de errores (definidas anteriormente para clasificación y regresión) sumadas para cada hoja. Consideremos que el tamaño de un árbol T es su número de hojas y denotemos aquello por $|T|$. Lo anterior nos permite definir una métrica que incorpora tanto el error como el tamaño de un árbol:

$$R_\alpha(T) = R(T) + \alpha|T| \quad (6.9)$$

Esta expresión se puede interpretar como agregar una penalización por complejidad si pensamos α como costo en complejidad de un nodo terminal. Notemos que a medida que se aumenta α más estaremos prefiriendo un árbol con menos hojas, por ende la métrica es sensible a tal hiperparámetro. Usaremos este hecho para construir un algoritmo que seleccione árboles adecuados de los cuales seleccionar el mejor, donde la idea se resume a continuación:

Dado el árbol original T_0 , el objetivo será construir una sucesión de árboles T_0, T_1, \dots, T_m , disminuyendo en cada paso el número de nodos terminales y donde T_m es simplemente el nodo raíz. Para aquello notemos que pese a que el número de subárboles de T_0 es potencialmente grande, siempre es un número finito. Con esto, si $T(\alpha)$ es el árbol que minimiza el costo R_α , entonces al aumentar α , aquel

árbol seguirá siendo el óptimo hasta llegar a un punto de salto α' , en el cual un nuevo árbol $T(\alpha')$ se convierte en el mínimo y así sucesivamente.

Este punto se encuentra guardando los costos y tamaños de subárboles dados por podar en algún nodo. Cuando podemos esto consistirá en deshacer la separación hecha en el nodo en cuestión, con lo cual nos quedará la estimación que teníamos para el subconjunto original sin separar partes derecha e izquierda. La idea es encontrar iterativamente el nodo más débil que podamos podar.

A continuación el algoritmo de poda (Ripley, 2008), luego del cual podemos enunciar los resultados que lo justifican. Precisemos que T_t se refiere al subárbol cuya raíz (nodo del cual salen todas las ramas) es el nodo t .

Algoritmo 2 Poda de costo-complejidad

```

1: function SUCESIONARBOLES( $T$ )
2:   set  $k = 0$ ,  $T_0 = T$ ,  $\alpha = \infty$ 
3:   for  $t$  in nodos no terminales desde abajo hacia arriba do:
4:     calcular  $R(T_t)$  y  $|T_t|$  sumando sobre los descendientes e incluyendo contribuciones en  $t$ 
5:     set  $g(t) = \frac{R(t) - R(T_t)}{|T_t| - |t|}$ 
6:     set  $\alpha = \min(\alpha, g(t))$ 
7:   for  $t$  in nodos no terminales de arriba hacia abajo do:
8:     if  $g(t) = \alpha$  then
9:       Reemplazar  $T_t$  por  $t$  (podar)
10:      set  $k = k + 1$ ,  $\alpha_k = \alpha$  y  $T_k = T$ 
11:      if  $T$  es un árbol de un sólo nodo then
12:        return  $\alpha_1, \dots, \alpha_m, T_1, \dots, T_m$ 
13:      else
14:        Ir a paso (3)

```

Notemos que la función g nace de querer despejar aquel α que le de suficiente peso al tamaño del árbol de modo que $R_\alpha(T_t) = R_\alpha(t)$, esto es

$$R(T_t) + \alpha|T_t| = R(t) + \alpha|t| \iff \alpha = \frac{R(t) - R(T_t)}{|T_t| - |t|}. \quad (6.10)$$

Proposición 6.0.1 (Consistencia de poda costo-complejidad). *Sea $g(t, T) = \frac{R(t) - R(T_t)}{|T_t| - |t|}$ para un nodo t y un subárbol T_t con raíz en t .*

1. *El resultado de podar en un nodo t si $R_\alpha(t) \leq R_\alpha(T_t)$ al visitar los nodos de abajo hacia arriba, el árbol resultante es*

$$T(\alpha) = \arg \min_{\tilde{T} \leq T} R_\alpha(\tilde{T}) \quad (6.11)$$

2. *Sea $\tilde{\alpha} = \min\{g(t, T) : t \text{ es nodo no terminal de } T\}$, podar en todos aquellos nodos que cumplan $g(t, T) = \tilde{\alpha}$ resulta en $T(\tilde{\alpha})$. Además, $g(t, T(\tilde{\alpha})) > \tilde{\alpha}$ para todo nodo t no terminal en $T(\tilde{\alpha})$.*

3. *Para $\beta > \alpha$, $T(\beta)$ es subárbol de $T(\alpha)$ y es el resultado de β -poder $T(\alpha)$.*

Hasta ahora lo único que hemos hecho es definir una secuencia de árboles conveniente, sin embargo esto nos deja la responsabilidad de elegir un buen árbol para nuestro estimador final. Lo que tenemos hasta ahora son:

$$T = T_0 < T_1, \dots, T_{m-1} < T_m, \quad (6.12)$$

donde $<$ denota la relación “ser subárbol”. Además tenemos una sucesión $\alpha_0 < \alpha_1 < \dots < \alpha_m$ (donde en este caso $<$ es la relación “menor a” usual en los números reales). Es de esperar que el error de

entrenamiento aumente a medida que aumentamos α . Este no es necesariamente el caso en un conjunto de testeo, pues es probable que los árboles con muchas hojas sean resultado de un sobreajuste. Una manera razonable de escoger un árbol final es tomar aquel que tenga menos error en un conjunto de entrenamiento o validación. También podemos hacer uso de técnicas como validación cruzada.

6.2.5. Interpretabilidad

Nos hemos restringido al ajuste de un árbol, sin embargo es útil pensar en las ventajas que puede tener este tipo de modelos respecto a otros vistos en el curso. Por su naturaleza, los árboles tienen una buena capacidad de interpretabilidad.

Usando las variables que se usaron para cortar cada nodo y los valores para el corte óptimo, podemos explicar la diferencia en estimaciones obtenidas. A esto lo llamamos capacidad de interpretación, y es importante a la hora de usar aprendizaje de máquinas para decisiones con repercusión en el mundo real.

En este caso se ha ajustado un árbol para clasificar imágenes de dígitos escritos a mano (dataset *mnist*). Acá, cada variable corresponde a un pixel en particular, que en La Figura 33 están destacados en rojo. Cuando se llegan a hojas del árbol se logra, en muchos casos, distinguir dígitos. Por otro lado, nodos intermedios denotan conjuntos impuros (con varios dígitos posibles), de los cuales es necesario realizar un corte. Se vislumbran entonces que pixeles son más importantes a la hora de distinguir dígitos y que desencadenan en la decisión tomada por el árbol de clasificación.

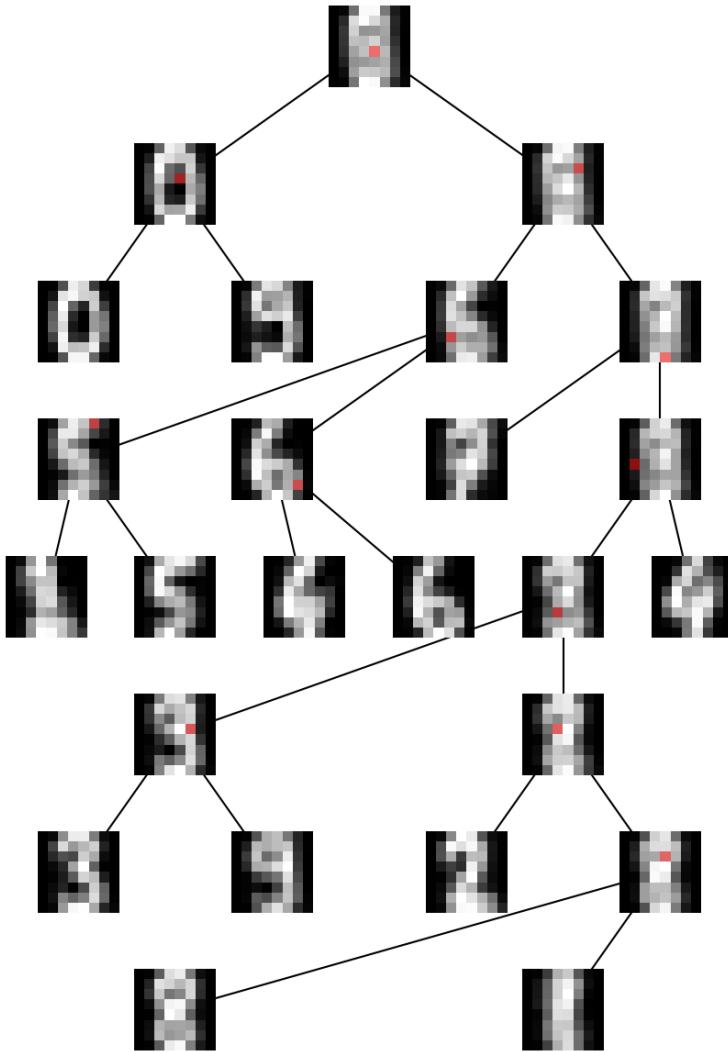


Fig. 33. Visualización de un árbol de clasificación para el dataset *mnist*.

6.3. Bagging

Esta sección tiene como objetivo mostrar un método para ensamblar modelos generales (Breiman, 1996). Antes de discutirlo, necesitamos la noción del concepto *bootstrapping*.

6.3.1. Método Bootstrapping

En palabras simples, *bootstrapping* es un procedimiento que consiste en escoger aleatoriamente puntos de datos con repetición, desde el conjunto de datos original. La repetición de esto nos permite acceder a varias distribuciones que intentan aproximar la original.

Con estas distribuciones, que serán típicamente de menor tamaño, se espera capturar la variabilidad de los datos. Un ejemplo de uso es ajustar modelos a las diferentes distribuciones de *bootstrap*. Una idealización de lo anterior se muestra en La Figura 35, donde se usan dos re-muestreos (con repetición) para el ajuste.

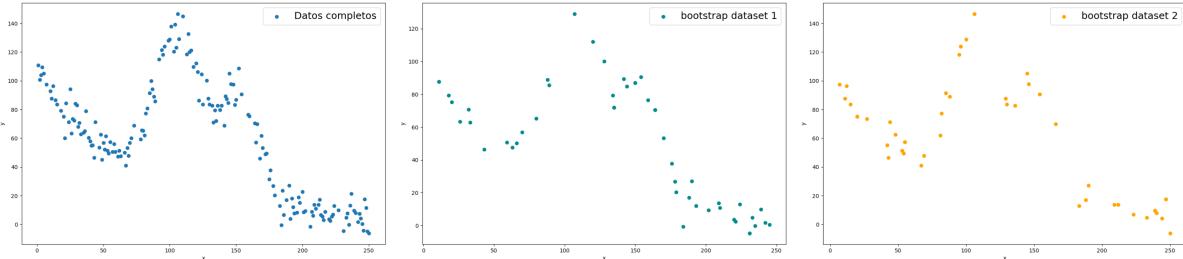


Fig. 34. Ejemplo de conjuntos de datos muestreados con *bootstrapping*.

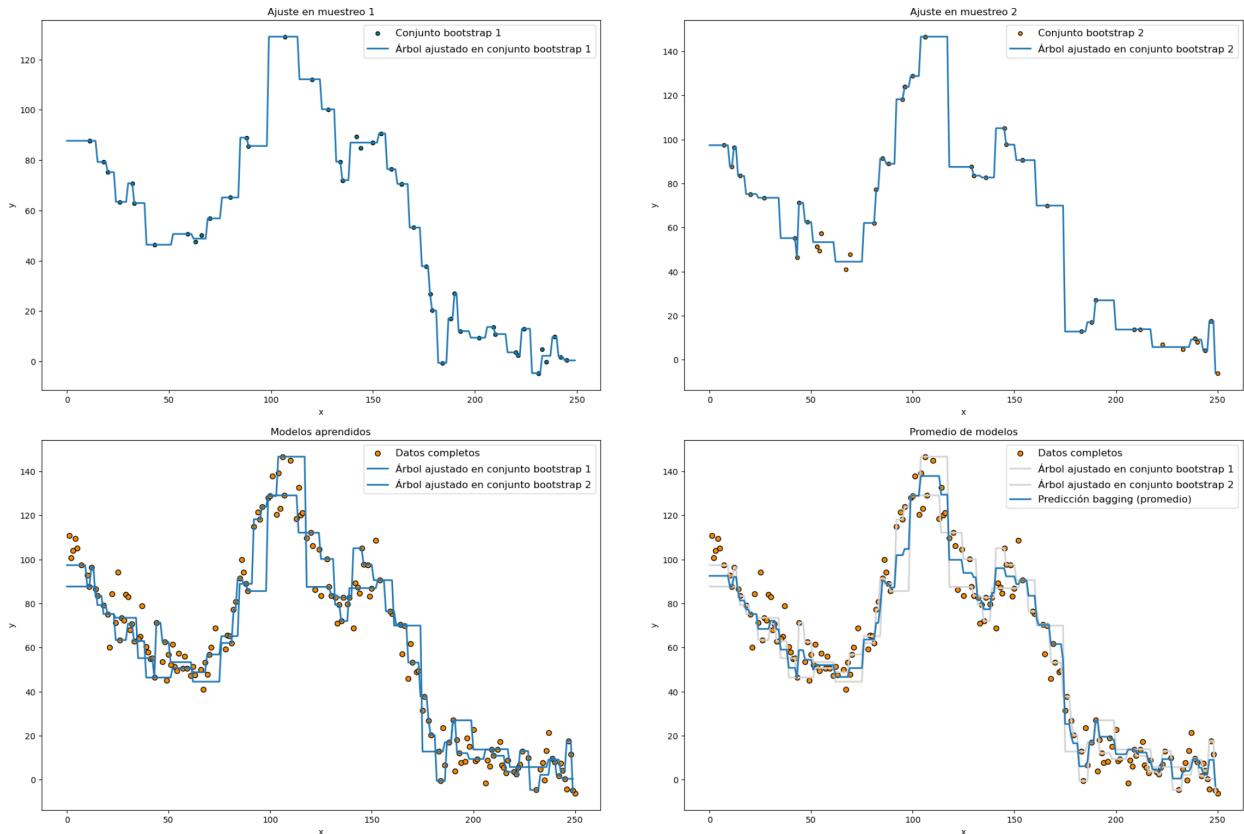


Fig. 35. Ejemplo de predicción *bagging* (promedio) usando dos modelos de árboles.

Podemos luego ver la diferencia de los estimadores versus los datos originales. Hacer esto varias veces nos puede dar una noción punto por punto de la varianza en ciertos puntos del input. Es interesante notar que si en vez de samplear desde los datos originales considerásemos un modelo con ruido Gaussiano y muestreamos de esa distribución obtendríamos el equivalente modelo ajustado con mínimos cuadrados a medida que consideramos más distribuciones.

El caso general es consistente con encontrar un modelo que maximice la verosimilitud. Usaremos este principio para definir *Bagging*, que será usar la información de varios modelos aprendidos en distintas distribuciones de *bootstrap*.

6.3.2. Bagging: agregación de modelos

Sea $\{\mathcal{D}^{(b)}\}_{b=1}^B$ una colección finita de conjuntos de datos tomados con el método *bootstrap*, es decir, muestreando repetidamente del conjunto de datos original $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$ con repetición. Supongamos que en cada conjunto ajustamos un modelo, con lo cual tenemos B modelos: $\{\varphi(x, \mathcal{D}^{(b)})\}_{b=1}^B$.

El estimador resultante de usar el método *bagging* en cada caso estará dado por:

$$\varphi_{\text{bagging}}(x) = \frac{1}{B} \sum_{b=1}^B \varphi(x, \mathcal{D}^{(b)}). \quad (6.13)$$

El nombre ***bagging*** proviene de la combinación de ***bootstrap*** y ***aggregating***, lo último correspondiendo a agregación de modelos, lo cual viene de usar el promedio. En el caso de clasificación, lo usual es votar, i.e., tomar la clase que sea preferida por la mayor cantidad de modelos.

Es natural preguntarse en qué casos será conveniente usar *bagging* por sobre un modelo ajustado en el conjunto de datos original. La respuesta a esto dependerá de la sensibilidad de los modelos de base. En efecto denotemos

$$\varphi_{\text{bagging}}(x) = \mathbb{E}_{\tilde{D} \sim D} [\varphi(x, \tilde{D})], \quad (6.14)$$

donde $\tilde{D} \sim D$ denota que el conjunto de datos usado para ajustar el modelo fue tomado usando muestreos que distribuyen de acuerdo a la distribución de probabilidad del modelo original. Usando esto, podemos acotar el error esperado usando que:

$$\mathbb{E}_{\tilde{D} \sim D} [y - \varphi(x, \tilde{D})]^2 = y^2 - 2y\mathbb{E}_{\tilde{D} \sim D} [\varphi(x, \tilde{D})] + \mathbb{E}_{\tilde{D} \sim D} [\varphi^2(x, \tilde{D})]. \quad (6.15)$$

Además, gracias a la desigualdad de Cauchy-Schwartz, podemos acotar el tercer término:

$$\mathbb{E}_{\tilde{D} \sim D} [\varphi^2(x, \tilde{D})] \geq \mathbb{E}_{\tilde{D} \sim D} [\varphi(x, \tilde{D})]^2. \quad (6.16)$$

Con esto queda:

$$\mathbb{E}_{\tilde{D} \sim D} [y - \varphi(x, \tilde{D})]^2 \geq (y - \mathbb{E}_{\tilde{D} \sim D} [\varphi(x, \tilde{D})])^2 \quad (6.17)$$

$$= (y - \varphi_{\text{bagging}}(x))^2. \quad (6.18)$$

Luego considerando todos los datos obtenemos que el error cuadrático promedio de los modelos será mayor al error cuadrático del estimador *bagging*. El margen de mejora depende solamente de cuán fuerte es la desigualdad de la inecuación 6.16, que podemos re-escribir como

$$\text{Var}_{\tilde{D} \sim D} [\varphi(x, \tilde{D})] = \mathbb{E}_{\tilde{D} \sim D} [\varphi^2(x, \tilde{D})] - \mathbb{E}_{\tilde{D} \sim D} [\varphi(x, \tilde{D})]^2 \geq 0. \quad (6.19)$$

Dicho de otro modo, la varianza del modelo para las distribuciones de *bootstrap* serán esenciales para que existan ganancias significativas. El método *bagging* es entonces recomendable para modelos **inestables**, i.e., que varíen más bruscamente con cambios en los datos. Para modelos que no varíen demasiado con las distintas distribuciones *bagging* no introduce mejora (y en la práctica puede empeorar la predicción).

6.3.3. Bosques aleatorios y variaciones

En principio es razonable pensar que la técnica *bagging* funcionará con árboles, pues estos son relativamente sensibles a los datos utilizados (sobretodo si no podamos o si no usamos criterios de parada exigentes). Justamente cuando tenemos árboles suficientemente profundos, podemos asumir que tendrán un sesgo muy pequeño. La técnica *bagging* no afecta el sesgo de los estimadores, mejora sus resultados más bien reduciendo la varianza.

Supongamos que aprendemos B árboles, cada uno en un conjunto de datos generados con *bootstrapping*. Teóricamente cada estimador $\varphi(x, \mathcal{D}^{(b)})$ tendrá la misma varianza, que podemos denotar σ^2 , pues

los estimadores son independientes e idénticamente distribuidos. En la práctica los estimadores no son necesariamente independientes. Sea ρ la correlación dos-a-dos de los estimadores. En este caso, la varianza del estimador *bagging* (promedio) está dada por:

$$\rho\sigma^2 + \frac{1 - \rho}{B}\sigma^2,$$

a diferencia del caso i.i.d., para el cual la varianza final es

$$\frac{1}{B}\sigma^2.$$

En ambos casos la varianza se reduce al aumentar B , sin embargo el término $\rho\sigma^2$ nos restringe la reducción de varianza. La idea de los bosques aleatorios (conocidos como *random forest* en inglés) será reducir la correlación de los estimadores para reducir la varianza sin afectar fuertemente el sesgo, y de esta forma reducir el error. Haremos esto seleccionando aleatoriamente variables con las cuales aprender cada árbol, como se señala en Algoritmo 3 (Breiman, 2001).

Algoritmo 3 Bosques Aleatorios

```

1: function BOSQUEALEATORIO( $D, B$ )
2:   for  $b \in \{1, \dots, B\}$  do
3:     Obtener  $\mathcal{D}^{(b)}$  de  $D$  con el método bootstrap.
4:     Ajustar un árbol  $T_b$  en  $\mathcal{D}^{(b)}$  del modo siguiente:
5:       for nodo in  $T_b$  do
6:         seleccionar  $m$  variables de las  $p$  posibles
7:         seleccionar la mejor variable y corte
8:         partir del nodo según el corte
return  $\{T_b\}_{b=1}^B$ 
```

Para realizar las predicciones finales usamos:

$$\varphi_{RF}(x) = \frac{1}{B} \sum_{b=1}^B T_b(x),$$

o bien la clase con más votos en el caso de clasificación.

Lo aleatorio de los bosques aleatorios está en la selección de m variables. El número m es un parámetro en sí mismo, pero suele usarse $p/3$ para regresión y \sqrt{p} para clasificación, donde p es el número de parámetros.

Tanto en el caso de bosques aleatorios como en *bagging* general, se dice que los estimadores no sobre-ajustan. Si bien esto es cierto en teoría, en la práctica es usual que muchas variables no sean útiles para la predicción. El efecto de la selección de variables hace que muchas veces se ignoren las pocas variables que si pueden explicar los datos. Es posible que muchos de los árboles tengan una expresividad innecesaria, al usar variables inútiles, y por ende resulten en una mala generalización.

6.3.4. Árboles extremadamente aleatorios

Una alternativa a los bosques aleatorios es usar un promedio de modelos pero aprendidos en el mismo conjunto de datos (i.e., sin el método bootstrap). La “extrema” aleatoriedad proviene de modificar el ajuste de arboles del siguiente modo:

- Usar un subconjunto aleatorio de variables (como en bosques aleatorios).
- Para cada variable del subconjunto generar cortes de modo aleatorio, en vez de seleccionar el mejor de todos. De aquellos cortes generados, se escoge el mejor de acuerdo al criterio empleado.

A este método se le denomina árboles extremadamente aleatorios (*ExtraTrees* en inglés). Aparte de la decorrelación de los estimadores, *ExtraTrees* tiene una mejor eficiencia computacional que los árboles clásicos (Geurts, Ernst, y Wehenkel, 2006).

6.4. Boosting

En las sección *bagging* aprendimos como mejorar conjuntamente una colección de estimadores, ajustados cada uno por separado. En tal configuración asumimos que los modelos son insesgados y nos enfocamos en reducir varianza. En esta sub-sección veremos el método *boosting*, que ajusta un conjunto de modelos sesgados de manera adaptativa para reducir el sesgo del estimador final.

6.4.1. Motivación: algoritmos fuertes versus débiles

Antes de adentrar en detalles, recordemos las preguntas teóricas cuyas respuestas decantaron en el método final. En el análisis matemático de modelos predictivos es usual trabajar en el marco conceptual PAC, que significa probablemente aproximadamente correcto. En tal contexto, solemos seleccionar estimadores que con alta probabilidad tengan un bajo error de generalización (que sean aproximadamente correctos). En términos matemáticos, solemos tomar $\epsilon > 0$, $0 < \delta < 1$ de modo que nuestro algoritmo se dirá PAC-aprendible si logra tener un error de a lo más ϵ con probabilidad $1 - \delta$ (Schapire y Freund, 2012).

Por otro lado, podemos considerar un paradigma alternativo en el cual dado $\gamma > 0$ exijamos un modelo que tenga un error $\frac{1}{2} - \gamma$ con probabilidad menor a δ . Dicho de otro modo, en vez de pedir una alta probabilidad de precisión arbitraria, pedimos que la probabilidad de que nuestro algoritmo sea peor que 50 % sea baja. A un algoritmo que cumpla esto se le llamará débilmente PAC-aprendible (en contraste con PAC-aprendible, que también se le denomina fuertemente PAC-aprendible en este contexto).

La intuición nos dice que como la débil PAC-aprendibilidad es menos “exigente” que la fuerte, habrán más algoritmos débiles que algoritmos fuertes. Esto resulta ser falso. *boosting* en su sentido original es un algoritmo que al tomar un algoritmo débil es capaz de convertirlo en uno fuerte (en el sentido de PAC aprendibilidad). La existencia de algoritmos de *boosting* implica la equivalencia de la débil y fuerte PAC-aprendibilidad.

6.4.2. Algoritmo AdaBoost

Consideremos el caso en el que tenemos acceso a clasificadores de base que son débiles, en el sentido de ser ligeramente mejores que una elección por azar. Esta ligera mejora respecto de un modelo trivial es la única exigencia que se le da a los clasificadores de base. Estos serán tratados como caja negra y serán llamados como sub-rutina para mejorar la predicción final.

La clave del éxito para los métodos boosting es escoger los conjuntos de entrenamientos para los estimadores de base de tal modo que estos sean “forzados” a inferir nueva información de los datos que no estaba presente anteriormente. Al escoger puntos de datos en los cuales el rendimiento de los algoritmos de base son incluso peores que su rendimiento débil usual, podemos ajustar modelos débiles en ellos para acercarnos al rendimiento fuerte deseado. Esto se ilustra en La Figura 36.

Supongamos que tenemos un conjunto de datos $D = \{(x_i, y_i)\}_{i=1}^N$, donde N será el tamaño del dataset. Consideraremos únicamente el caso de clasificación binaria donde las etiquetas están dadas por el conjunto {-1, 1}. Supongamos que tenemos un modelo ϕ_m que consideramos débil. Usando la heurística anterior, queremos ajustar un nuevo modelo débil ϕ_{m+1} .

Usaremos la pérdida exponencial: $L(\phi) = e^{-y_i\phi(x_i)}$. Buscaremos minimizar esa pérdida, usando el estimador:

$$\phi(x) = \phi_m(x) + \alpha_{m+1}\phi_{m+1}(x),$$

con lo cual, debemos resolver

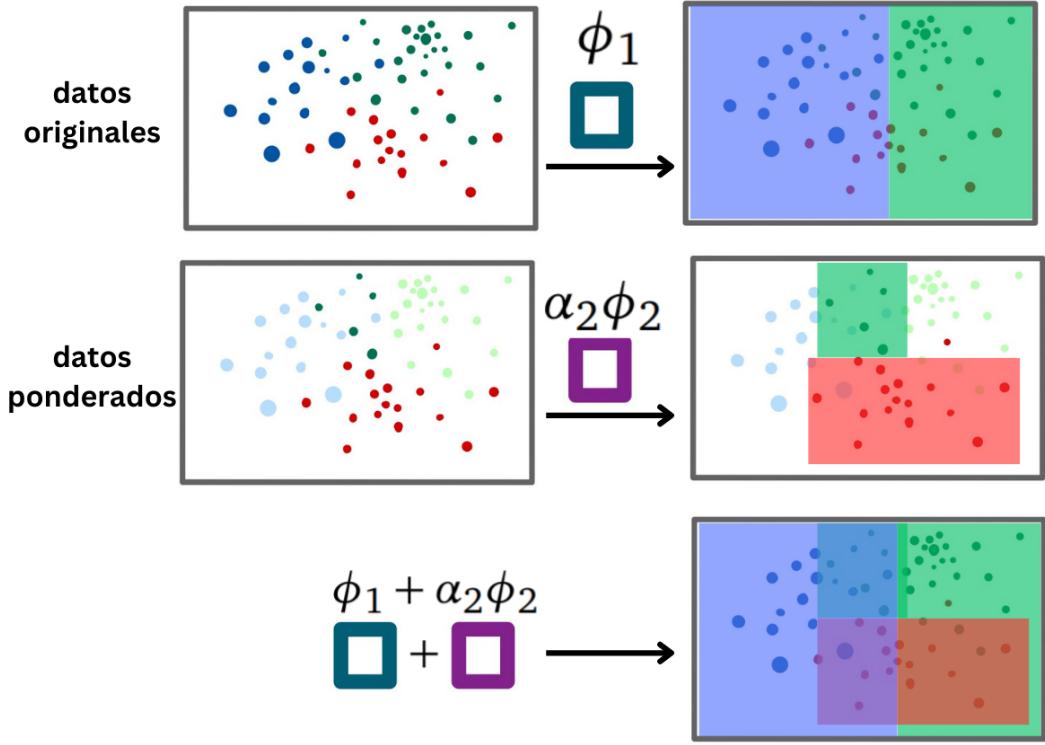


Fig. 36. Intuición del método *boosting*.

$$\min_{\alpha_{m+1}} L(\phi(x)) = \min_{\alpha_{m+1}} \sum_{i=1}^N e^{-y_i(\phi_m(x_i) + \alpha_{m+1}\phi_{m+1}(x))} \quad (6.20)$$

$$= \min_{\alpha_{m+1}} \sum_{i=1}^N w_i^{(m)} e^{\alpha_{m+1}\phi_{m+1}(x)}, \quad (6.21)$$

donde definimos $w_i^{(m)} = e^{-y_i\phi_m(x_i)}$ para $i = 1, \dots, N$, que lo podemos interpretar como el dataset original ponderado por el error (exponencial) del modelo ϕ_m .

Proposición 6.0.2. El α_{m+1} que minimiza la función anterior está dado por

$$\alpha_{m+1} = \frac{1}{2} \log\left(\frac{1 - \epsilon_m}{\epsilon_m}\right), \quad (6.22)$$

$$\text{donde } \epsilon_m = \frac{\sum_{y_i \neq \phi_m(x_i)} w_i}{\sum_{i=1}^N w_i}.$$

El desarrollo anterior resulta en el algoritmo 4 (Freund y Schapire, 1997; Hastie, Tibshirani, y Friedman, 2001). Se demostró que puede convertir clasificadores débiles en un clasificador débil (para el caso binario).

Algoritmo 4 AdaBoost

```

1: function ADABoost( $D, M$ )
2:   Set  $w_i = \frac{1}{N} \forall i = 1, \dots, N$  (con  $N$  tamaño del dataset)
3:   for  $m = 1, \dots, M$  do
4:     Entrenar un modelo débil  $\phi_m$  minimizando  $\sum_{y_i \neq \phi_m(x_i)} w_i$ 
5:     Set  $\epsilon_m = \frac{\sum_{y_i \neq \phi_m(x_i)} w_i}{\sum_{i=1}^N w_i}$ 
6:     Set  $\alpha_m = \frac{1}{2} \log(\frac{1-\epsilon_m}{\epsilon_m})$ 
7:     Set  $w_i = w_i \cdot \exp(-y_i \phi_m(x_i) \alpha_m)$ 
8:     Actualizar  $\{w_i\}_{i=1}^N$  de modo que  $\sum_{i=1}^N w_i = 1$ 
return  $\phi(x) = \text{signo}\left(\sum_{m=1}^M \alpha_m \phi_m(x)\right)$ 

```

Recordando las nociones de débil y fuerte aprendibilidad, podemos enunciar el siguiente resultado:

Teorema 6.1 (Boosting). *Una clase objetivo \mathcal{C} es (eficientemente) debilmente aprendible, si y solo si es (eficientemente) fuertemente aprendible*

Es evidente que un modelo fuerte es en particular débil. Por el contrario, la conversa (débil PAC-aprendibilidad implica fuerte PAC-aprendibilidad) es resultado de aplicar el algoritmo AdaBoost a modelos de base débiles como caja negra. Esta equivalencia fue presentada por Yoav Freund y Robert Schapire, quienes ganaron el prestigioso premio Gödel gracias a esta contribución.

Si bien la demostración del teorema tiene elementos que escapan el enfoque de este curso, enunciamos a continuación un resultado que nos da una idea de la capacidad de AdaBoost para reducir el error (en el conjunto de entrenamiento) a medida que aumentamos la cantidad de estimadores.

Teorema 6.2. *Sea $\gamma_m = \frac{1}{2} - \epsilon_m$ y sea D_1 una distribución inicial arbitraria sobre el dataset de entrenamiento. El error de entrenamiento (ponderado por los pesos) del clasificador combinado con respecto a D_1 está acotado por:*

$$\mathbb{P}_{i \sim D_1}(\phi(x_i) \neq y_i) \leq \prod_{m=1}^M \sqrt{1 - 4\gamma_m^2} \leq e^{-2 \sum_{m=1}^M \gamma_m^2}. \quad (6.23)$$

Notemos que esta cota depende de γ_m . Recordemos que ϵ_m es la tasa de error. Mientras mas grande sea este, más pequeño será γ_m y por ende más cercano a 1 será $\sqrt{1 - 4\gamma_m^2}$ (pero de cualquier modo tenemos un número menor a uno). El teorema muestra un decrecimiento exponencial del error en el número de estimadores M . Esto garantiza bajo error incluso cuando el margen entre una elección azarosa (error $\frac{1}{2}$) y el error de los estimadores es pequeño.

6.4.3. Modelamiento aditivo por etapas

La justificación teórica que nos lleva al algoritmo de AdaBoost es en realidad un caso particular del método de modelamiento aditivo por etapas hacia adelante (*forward stagewise additive modelling* en inglés) (Hastie y cols., 2001). En efecto, este método toma un modelo de base (cuyos parámetros son a determinar) y genera iterativamente un modelo sumando el modelo de base ponderado por algún factor. Optimizamos tanto los parámetros del modelo como el factor en cada iteración. Esto es, partiendo de $f_0(x) = 0$, repetimos:

- Para $m = 1, \dots, M$ resolver

$$(\beta_m, \gamma_m) = \arg \min_{\beta, \gamma} \sum_{i=1}^N L(y_i, f_{m-1}(x_i) + \beta b(x_i, \gamma)),$$

donde b es nuestro modelo de base parametrizado por γ .

- Actualizar la sucesión de funciones

$$f_m(x) = f_{m-1}(x) + \beta b(x; \gamma_m).$$

Esta configuración nos permite usar una variedad de funciones de perdida distintas, así como también variadas familias de modelos. Una restricción obvia es que la combinación de estas sea fácilmente optimizable, pues la minimización a pasos puede ser difícil de computar.

Notemos que al usar el error cuadrático estamos minimizando (en cada paso m):

$$\sum_{i=1}^N (y_i - f_{m-1}(x_i) + \beta b(x_i, \gamma))^2 = \sum_{i=1}^N (r_{mi} - \beta b(x_i, \gamma))^2.$$

donde hemos usado la notación $r_{mi} = y_i - f_{m-1}(x_i)$, que denota el residuo de la función f_{m-1} respecto al valor real y_i . Notemos como nuestro problema se reduce al ajuste de un modelo a los residuos del modelo en el paso anterior. Esto coincide con la noción de *boosting* inicial (aprender donde nos equivocamos).

6.4.4. *GradientBoosting*: boosting como descenso de gradiente funcional

Volviendo al caso general, nos podemos plantear la situación en la cual no busquemos minimizar con respecto a ponderadores y parámetros, si no más bien busquemos una función que minimice la perdida que estemos usando.

$$\hat{f} = \arg \min_f L(f).$$

En esta versión funcional, podemos construir un algoritmo que refleje el caso general. Esta vez podemos apoyarnos en el uso del algoritmo de descenso de gradiente para acercarnos al minimizado del costo. Informalmente podemos escribir:

$$f_{n+1} = f_n - \rho_n \nabla L(f_n).$$

para alguna sucesión de pasos $\{\rho_n\}_{n \geq 0}$ y algún f_0 conveniente. Notemos como tenemos una suma sucesiva de funcionales. Podemos conectar esta noción con el modelo aditivo paso por paso. Pero primero consideraremos una aproximación numérica del problema. Podemos tomar el conjunto de evaluaciones de la función en el conjunto de datos como representación de este:

$$\tilde{f} = (f(x_0), \dots, f(x_N)).$$

Esto juega el rol de parámetros de nuestro modelo. La ventaja es que el gradiente también se vuelve un vector de N dimensiones:

$$\nabla L(\tilde{f})_i = \left(\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right).$$

Es importante que nuestra función de costo sea diferenciable para aplicar este método. Cuando tenemos una forma cerrada para la derivada podemos simplemente evaluar en el punto de dato en cuestión.

Tenemos entonces una manera de actualizar las evaluaciones de nuestra función, sin embargo esto no nos da una expresión que podamos actualizar a un nuevo input arbitrario. Es por esto que usaremos estos N puntos para ajustar nuestro modelo débil. Para la función inicial, basta con ajustar un modelo débil con los datos originales como lo haríamos usualmente. El método se resume en el algoritmo 5 (Murphy, 2022).

Algoritmo 5 GradientBoosting

```
1: function GRADIENTBOOSTING( $D, M$ )
2:   Ajustar un modelo débil en los datos  $\mathcal{D}$ , i.e., fijar  $\phi_0(x) = \arg \min_{\phi} \sum_{i=1}^N L(y_i, \phi(x_i))$ 
3:   for  $m = 1, \dots, M$  do
4:     Calcular

$$r_{im} = - \left[ \frac{\partial L(y_i, \phi(x_i))}{\partial \phi(x_i)} \right]_{\phi(x_i)=\phi_{m-1}(x_i)}$$

5:     Entrenar un modelo débil  $\phi_m$  minimizando  $\sum_{i=1}^N (r_{im} - \phi(x_i))^2$ 
6:     Calcular  $\rho_m = \arg \min_{\rho} \sum_{i=1}^n L(y_i, \phi_{m-1}(x_i) + \rho \phi_m(x_i))$ 
7:     Actualizar  $\phi_m(x) = \phi_{m-1}(x) + \rho \phi_m(x_i)$ 
return  $\phi(x) = \phi_M(x)$ 
```

Si nuestra elección de pérdida es el error cuadrático, entonces basta notar que

$$r_i = \frac{\partial L(y_i, \phi(x_i))}{\partial \phi(x_i)} = y_i - \phi(x_i)$$

para obtener nuevamente el ajuste de nuestro modelo al residuo del modelo anterior. Esto es consistente con usar el modelamiento aditivo por etapas y se le denomina L2boosting, correspondiente a una de las muchas variantes de *boosting* (Hastie y cols., 2001).

Algoritmo 6 GradientTreeBoosting

```
1: function GRADIENTTREEBOOSTING( $D, M$ )
2:   Ajustar un árbol  $\phi_0$  en los datos  $\mathcal{D}$ .
3:   for  $m = 1, \dots, M$  do
4:     Calcular  $r_{im} = - \left[ \frac{\partial L(y_i, \phi(x_i))}{\partial \phi(x_i)} \right]_{\phi(x_i)=\phi_{m-1}(x_i)}.$ 
5:     Ajustar un árbol  $\phi_m$  en  $\{r_{im}\}_{i=1}^N$ .
6:     Fijar  $R_{jm}$  como los conjuntos correspondientes a los nodos, enumerados de 1 hasta  $J_m$ .
7:     for  $j \in 1, \dots, J_m$  do
8:        $\gamma_{jm} = \arg \min_{\gamma} \sum_{x_j \in R_{jm}} L(y_i, \phi_{m-1}(x_i) + \gamma)$ 
9:       Calcular  $\rho_m = \arg \min_{\rho} \sum_{i=1}^n L(y_i, \phi_{m-1}(x_i) + \rho \phi_m(x_i))$ 
10:      Actualizar  $\phi_m(x) = \phi_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}_{R_{jm}}(x)$ 
return  $\phi(x) = \phi_M(x)$ 
```

Como aplicación particular de la idea de *Gradient boosting* observemos el algoritmo 6, llamado *Gradient tree boosting*. La principal diferencia con usar *Gradient boosting* con árboles como modelo de base es que ajustamos el factor multiplicativo en cada hoja del árbol y lo usamos para reemplazar derechamente la estimación para ese subconjunto en cuestión. Esto hace que se ponderen más el aporte de aquellos nodos importantes para mejorar la predicción del ensamblaje. Este modelo fue considerado por Leo Breinman como el mejor modelo *off-the-shelf* en el caso de clasificación, i.e., un modelo confiable para variadas tareas de clasificación, usualmente logrando muy buenas métricas.

6.4.5. Aspectos prácticos

Boosting es un método ampliamente utilizado y adaptable. Su distintas variantes han tenido variadas aplicaciones. Un ejemplo clásico es el algoritmo de detección de rostros de Viola-Davis (2001), que usa

una variante de AdaBoost. A diferencia de métodos usuales que usan redes neuronales, la implementación de Viola-Davis tiene muchos menos parámetros y con rápida inferencia (muchas cámaras portátiles incorporan el algoritmo).

Un punto en contra de los modelos tipo *boosting* es el hecho de necesitar ajustar un estimador para realizar el paso siguiente. Esta naturaleza secuencial del aprendizaje limita su escalabilidad para problemas grandes. Gran parte del éxito de las redes neuronales hoy en día se debe a que buena parte de su entrenamiento se puede realizar de manera paralelizada en GPUs. Esto no es posible para modelos aditivos donde el aprendizaje se hace por etapas, pero han existido variantes de *boosting* que si parallelizan una parte del entrenamiento (como *XGBoost*).

6.4.6. Interpretabilidad de modelos basados en árboles

Otra desventaja del método es limitada capacidad de interpretación, sobretodo respecto al uso de árboles. Podemos recurrir a métodos para estimar la importancia relativa de las distintas variables. En el caso de un árbol podemos usar

$$I^2 = \frac{1}{M} \sum_{m=1}^M I_l^2(T_m) \quad (6.24)$$

para estimar la importancia de la variable l . Acá T_m es el árbol m -ésimo de la suma (por ejemplo usando *boosting* o *bagging*). A I_l^2 se le denomina relevancia cuadrática y está dada por:

$$I_l^2(T) = \sum_{t=1}^{J-1} \hat{i}_t^2 \mathbf{1}_{\{v(t)=l\}}^{(t)}. \quad (6.25)$$

Acá la suma se hace sobre los nodos internos de un árbol. En cada una de ellas, una de las variables $X_{v(t)}$ es usada para cortar el conjunto en dos regiones. Por otro lado, \hat{i}_t^2 denota la mejora estimada en términos del error cuadrático con respecto a no haber realizado el corte.

6.4.7. Más pérdidas y variaciones

Como vimos en secciones anteriores, es posible usar otras funciones de pérdida L tanto para modelamiento aditivo por etapas como para *Gradient boosting*. Por ejemplo, usar:

$$L(y, \phi) = |y - \phi(x)|$$

hace que debamos usar el signo de $y - \phi(x)$ como derivada en *Gradient boosting*. Esto significa que nos movemos en la dirección del valor real y .

También podemos considerar la pérdida *Logloss* o entropía cruzada para el caso clasificación binaria:

$$L(y, \phi) = \log(1 - e^{-y\phi(x)}).$$

Esta pérdida tiene el mismo minimizador de población que la pérdida exponencial, con lo cual usar ambas es equivalente en el caso límite de tener un dataset infinito. Pese a esto, usar la *Logloss* tiene la ventaja de tener una interpretación probabilística usando:

$$p(y = 1|x) = \frac{1}{1 + e^{-2\phi(x)}}.$$

Además, los errores son penalizados severamente con la pérdida exponencial. Por el contrario la *Logloss* castiga linealmente los errores. Usarla deriva en el algoritmo *LogitBoost* (Hastie y cols., 2001).

Más variaciones pueden surgir del uso de estrategias para evitar el sobreajuste, entre las cuales encontramos:

- Detención temprana: una alternativa a fijar un número de estimadores M pequeño es tener un conjunto de validación que permita evaluar cuando es apropiado parar de agregar modelos.
- *Schrinkage*: ponderar cada actualización por algún factor pequeño.
- Podar estimadores, esto es, evaluar los elementos de la suma y eliminarlos si su error es alto.
- *Stochastic gradient boosting*: escoger de manera aleatoria un mini-batch con el cual entrenar cada modelo débil. Esto guarda similitud con el método de *bagging*.

Se han desarrollado además numerosas variantes que intentan corregir algunas desventajas de los métodos de *boosting*. Los ejemplos más usados son *CatBoost* (capaz de manejar variables categóricas), *Histogram-based Gradient Boosting* (simplifica las variables agrupándolas como histogramas para simplificar los cortes) y *Extreme Gradient Boosting* (construcción de árboles de forma paralela, aproximación de segundo orden del gradiente para acelerar cálculos y regularización), también conocido como *XGBoost*.

7. Aprendizaje no supervisado

7.1. Reducción de dimensionalidad

El problema de reducción de dimensionalidad consiste con construir una representación de dimensión estrictamente menor que los datos originales con la finalidad de interpretar de mejor forma la información contenida en nuestros datos así como también disminuir el costo computacional en el entrenamiento.

7.1.1. Análisis de componentes principales (PCA)

Consideremos ahora un conjunto de observaciones de $\{\mathbf{x}_i\}_{i=1}^N \subset \mathbb{R}^M$, donde denotamos $\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{iM}]^\top$. Podemos entender que el elemento x_{ij} corresponde al valor del atributo j para la observación i .

Notemos que cada observación puede descomponerse en la base canónica $\{\mathbf{e}_i\}_{i=1}^M$ de \mathbb{R}^M de la forma

$$\mathbf{x}_i = x_{i1}\mathbf{e}_1 + x_{i2}\mathbf{e}_2 + \dots + x_{iM}\mathbf{e}_M \quad (7.1)$$

Notemos que es posible representar cada vector \mathbf{x}_i mediante una cantidad $M' < M$ de términos, truncando la representación anterior, es decir,

$$\mathbf{x}_i \approx \sum_{j=1}^{M'} x_{i\sigma(j)}\mathbf{e}_{\sigma(j)} \quad (7.2)$$

donde $\sigma : \{1, 2, \dots, M\} \mapsto \{1, 2, \dots, M\}$ es una permutación que prioriza las coordenadas más representativas de los datos. Dichas aproximaciones de las observaciones $\{\mathbf{x}_i\}_{i=1}^N$ son una versión de baja dimensión, entonces, naturalmente nos podemos hacer la siguiente pregunta: dada una dimensión $M' < M$ ¿es efectivamente un subconjunto de los vectores canónicos la mejor base para descomponer las observaciones? ¿cómo encontramos la *mejor* base?

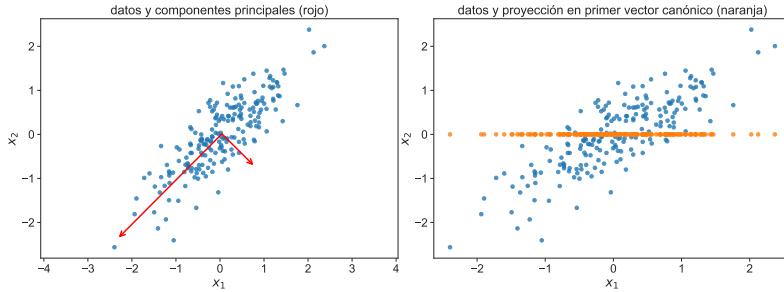


Fig. 37. Base ortogonal de \mathbb{R}^2 formada por las componentes principales de los datos. Se observa en este caso que ninguna de las componentes principales corresponde a un vector de la base canónica. El eje que cruza los cuadrantes 1 y 3 corresponde a \mathbf{c}_1 y el otro, a \mathbf{c}_2 .

La respuesta a la primera pregunta es, en la mayoría de los casos, negativa. Esto es porque los elementos de la base canónica, por sí solos, llevan poca información estructural que puede ser encontrada en los vectores observados. En particular, consideremos el caso en donde solo se dispone de dos observaciones $\{\mathbf{x}_1, \mathbf{x}_2\}$, si $M' = 2$, entonces una descomposición que garantiza error nulo es simplemente elegir \mathbf{x}_1 y \mathbf{x}_2 como bases de la nueva descomposición, donde los coeficientes estarían dados por $[1, 0]^\top$ y $[0, 1]^\top$.

Para encontrar la *mejor* base, lo primero que se requiere es definir qué se entiende por *mejor*. Nos enfocaremos en determinar una base cuyos componentes **ordenados** $\mathbf{c}_1, \mathbf{c}_2, \dots$ capturan las M' direcciones ortogonales de máxima variabilidad de nuestros datos. De esta forma, dado que $\langle \mathbf{c}, \mathbf{x} \rangle$ representa la proyección ortogonal de \mathbf{x} sobre \mathbf{c} , el primer elemento de la nueva base estará dado por

$$\mathbf{c}_1 = \arg \max_{\|\mathbf{c}\|=1} \langle \mathbf{c}, \mathbf{x} \rangle \quad (7.3)$$

Este criterio es conocido como **análisis de componentes principales (PCA)**. Notemos que la restricción $\|\mathbf{c}_1\| = 1$ es necesaria ya que $\langle \lambda \mathbf{c}_1, \mathbf{x} \rangle = \lambda \langle \mathbf{c}_1, \mathbf{x} \rangle$ por lo que $\langle \mathbf{c}_1, \mathbf{x} \rangle$ puede crecer indefinidamente si no se fija una restricción sobre la norma de \mathbf{c} . Por esta razón, s.p.g. podemos fijar la norma de \mathbf{c}_1 en 1 y buscar una base ortonormal. Además, es importante estandarizar los datos:

- Características de media nula: la matriz X con $(X)_{ij} = (\mathbf{x}_i)_j$ debe tener columnas con media 0. Esto se consigue restando la media de la columna a cada una de las entradas de dicha columna. El objetivo de este ajuste es poder centrar los datos.
- Varianzas marginales unitarias: si una dimensión tiene una varianza marginal mayor que el resto, esta será más importante en la determinación de la dirección de máxima varianza solo por su magnitud y no por la relación entre variables. La normalización se consigue dividiendo cada entrada de la columna por la desviación estándar de la columna.

Como en general no contamos con la distribución de las observaciones $p(\mathbf{x})$, podemos considerar una aproximación muestral de la varianza en la ecuación (7.3) y resolver

$$\mathbf{c}_1 = \arg \max_{\|\mathbf{c}\|=1} \sum_{i=1}^N \langle \mathbf{c}, \mathbf{x}_i \rangle^2. \quad (7.4)$$

Podemos ahora usar la siguiente notación

$$X = \begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_N^\top \end{bmatrix} = \begin{bmatrix} x_{11} & \dots & x_{1M} \\ \vdots & \ddots & \vdots \\ x_{N1} & \dots & x_{NM} \end{bmatrix}$$

y reescribir la ecuación (7.4) como

$$\mathbf{c}_1 = \arg \max_{\|\mathbf{c}\|=1} \|X\mathbf{c}\|^2 = \arg \max_{\|\mathbf{c}\|=1} \mathbf{c}^\top X^\top X \mathbf{c} = \arg \max_{\mathbf{c}} \frac{\mathbf{c}^\top X^\top X \mathbf{c}}{\mathbf{c}^\top \mathbf{c}} \quad (7.5)$$

Por otra parte, se tiene la siguiente propiedad:

Proposición 7.0.1 (minimización del cociente de Rayleigh). *Sea $M \in \mathcal{M}_{nn}(\mathbb{R})$ matriz cuadrada simétrica, entonces, para el cociente de Rayleigh*

$$R(M, x) := \frac{x^\top M x}{x^\top x} \quad (7.6)$$

Su valor mínimo corresponde al menor valor propio de M , y es alcanzado en su vector propio asociado.

Demostración. Dado que $R(M, cx) = R(M, x)$ para todo $c \neq 0$, basta verlo para $\|x\| = 1$, por lo que se puede considerar como una restricción adicional. Por teorema de Lagrange:

$$L(x, \lambda) = x^\top M x - \lambda(\|x\| - 1) \implies \frac{\partial L}{\partial x} = 2x^\top M - 2\lambda x^\top = 0 \implies Mx = \lambda x \quad (7.7)$$

Por lo tanto, los puntos críticos de lagrangiano son los vectores propios v_i de M . Por otra parte:

$$R(M, v_i) = \frac{v_i^\top M v_i}{v_i^\top v_i} = \frac{v_i^\top \lambda_i v_i}{v_i^\top v_i} = \lambda_i \in \mathbb{R} \text{ ya que } M \text{ es simétrica.} \quad (7.8)$$

Por lo tanto, el valor mínimo de $R(M, x)$ es λ_{\min} y es alcanzado en el vector propio asociado v_{\min} . Por el mismo argumento, el valor máximo es λ_{\max} y es alcanzado en v_{\max} . ■

De esta forma, dado que $X^\top X$ es simétrica, su cociente de Rayleigh es maximizado en el vector propio asociado al valor propio máximo de $X^\top X$. Consecuentemente, la proyección de una observación \mathbf{x}_i en la dirección de máxima varianza, o bien la *primera componente principal*, está dada por

$$\mathbf{x}_i^{(1)} = \langle \mathbf{x}_i, \mathbf{c}_1 \rangle \quad (7.9)$$

donde \mathbf{c}_1 es el vector propio asociado al mayor valor propio de la matriz de covarianza muestral XX^\top .

El cálculo de las siguientes componentes se realiza de forma iterativa sobre los residuos del conjunto de observaciones con respecto a las componentes anteriores. De esta forma, PCA encuentra una nueva base ortonormal tal que las componentes maximicen la variabilidad donde en algunos casos se puede perder intepretabilidad de las nuevas características generadas, pues son combinaciones lineales de las características originales de los datos. A pesar de esto, utilizando las primeras 2 o 3 componentes PCA se pueden visualizar datos de alta dimensionabilidad de forma ilustrativa.

7.1.2. Kernel PCA

El método Kernel PCA es similar a PCA, pero esta vez se utiliza el truco del kernel para proyectar los datos. En ese sentido, en vez de calcular la matriz de covarianza empírica $X^\top X$, se utiliza la matriz de Gram dada por un kernel K donde

$$K_{ij} = K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$

Luego, se realiza PCA utilizando dicha matriz.

En la Figura 38 se puede observar un ejemplo en que el resultado de linear PCA no es suficiente, puesto que el problema es simétrico, mientras que KPCA realiza una correcta separación de ambos clusters.

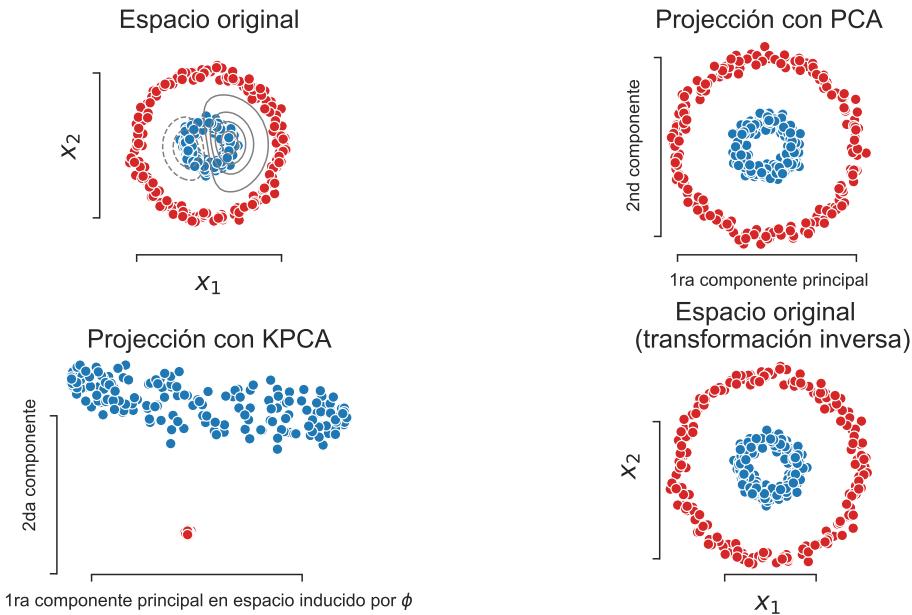


Fig. 38. Ejemplo de KPCA sobre un conjunto de datos que no es linealmente separable.

7.1.3. Probabilistic PCA

PCA probabilístico (PPCA) tiene su inspiración en que PCA se puede expresar como la solución vía máxima verosimilitud de un modelo probabilístico de variable latente. De este modo, PPCA propone un

método iterativo para obtener la solución evaluando solo cierto número de componentes, sin necesidad de calcular la matriz de covarianza empírica.

El modelo probabilístico para PCA en el que se inspira PPCA es el siguiente:

Sean $(x_i)_{i=1}^N \subset \mathbb{R}^M$ los elementos observados, inputs o variables y $z \in \mathbb{R}^l$ una variable latente explícita correspondiente al espacio de las componentes principales. Bajo la hipótesis de un modelo de observación lineal:

$$x = Wz + \mu + \epsilon \quad (7.10)$$

Donde $\epsilon \sim \mathcal{N}(0, \sigma^2)$ es un sumando de ruido y $W \in \mathbb{R}^{M \times l}$, $\mu \in \mathbb{R}^M$ y σ^2 son parámetros a determinar, se define el siguiente prior para z :

$$p(z) = \mathcal{N}(0, \mathbb{I}) \quad (7.11)$$

De este modo, la distribución condicional de x dado z también es gaussiana:

$$p(x|z) = \mathcal{N}(Wz + \mu, \sigma^2 \mathbb{I}) \quad (7.12)$$

Notemos que no se pierde generalidad tomar el prior para z con media cero y varianza unitaria, puesto que si se toma otro prior más general, se produce el mismo modelo.

Dado que tenemos un modelo paramétrico probabilístico, podemos estimar los parámetros con máxima verosimilitud. Dado los datos $\mathcal{D} = \{x_i\}_{i=1}^N$, la log-verosimilitud está dada por:

$$\log p(\mathcal{D}|W, \mu, \sigma^2) = \sum_{i=1}^N \log p(x_i|W, \mu, \sigma^2) \quad (7.13)$$

$$= \frac{NM}{2} \log(2\pi) - \frac{N}{2} \log|C| - \frac{1}{2} \sum_{i=1}^N (x_i - \mu)^T C^{-1} (x_i - \mu), \quad (7.14)$$

con $C = WW^T + \sigma^2 I$.

Usando la condición de primer orden obtenemos

$$\mu = \bar{x} = \frac{1}{N} \sum_{i=1}^N x_i \quad (7.15)$$

De esta manera tenemos la función de log-verosimilitud completa:

$$\log p(X, Z|W, \mu, \sigma^2) = \sum_{i=1}^N \{\log p(x_i|z_i) + \log p(z_i)\} \quad (7.16)$$

y evaluando en $\mu = \bar{x}$

$$\begin{aligned} \mathbb{E}[p(X, Z|W, \mu, \sigma^2)] = -\sum_{i=1}^N & \left\{ \frac{l}{2} \log(2\pi\sigma^2) \right. \\ & + \frac{1}{2} \text{Tr}(\mathbb{E}[z_i z_i^T]) \\ & \frac{1}{2\sigma^2} \|x_i - \mu\|^2 - \frac{1}{\sigma^2} \mathbb{E}[z_i]^T W^T (x_i - \mu) \\ & \left. \frac{1}{2\sigma^2} \text{Tr}(\mathbb{E}[z_i z_i^T] W^T W) \right\} \end{aligned} \quad (7.17)$$

7.1.4. Discriminante lineal de Fisher

Para evitar los artefactos (sesgos) introducidos por clases asimétricas en el uso de mínimos cuadrados para clasificación, es posible interpretar el problema de clasificación como uno de *reducción de dimensionalidad*, en donde la reducción consiste representar nuestros datos en solo una dimensión, la cual representa su (grado de pertenencia a una) clase. Con este objetivo en mente, consideremos el problema de clasificación binaria de $x \in \mathbb{R}^M$, donde proyectamos x en un espacio **unidimensional** con respecto a un vector $a \in \mathbb{R}^M$ de acuerdo a:

$$y = a^\top x, \quad (7.18)$$

donde podemos definir un umbral b para asignar x a \mathcal{C}_1 si $y + b \geq 0$ y x a \mathcal{C}_2 en caso contrario. Notemos que de esta forma recuperamos el modelo lineal para clasificación.

En general, al proyectar un objeto M -dimensional en un espacio 1-dimensional, se pierde gran parte de la información, lo cual genera el hecho de que clases claramente separadas en el espacio M -dimensional puedan traslaparse al ser proyectadas a 1 dimensión cuando la elección del vector $a \in \mathbb{R}^M$ no es la mejor. Sin embargo, es posible ajustar el vector a con la finalidad de obtener una proyección de x que maximice el grado de separación entre clases.

Con el objetivo de encontrar el vector a que cumple con este requerimiento en base a un conjunto de datos \mathcal{D} , primero definamos las cardinalidades de clases mediante $N_1 = |\{x \in \mathcal{D} : x \in \mathcal{C}_1\}|$ y $N_2 = |\{x \in \mathcal{D} : x \in \mathcal{C}_2\}|$, lo cual permite calcular los promedios muestrales (centros de masa) de cada clase mediante:

$$\mu_1 = \frac{1}{N_1} \sum_{n \in \mathcal{C}_1} x_n \quad \mu_2 = \frac{1}{N_2} \sum_{n \in \mathcal{C}_2} x_n \quad (7.19)$$

La medida más simple de separación entre las proyecciones de las clases sobre a es la distancia entre las medias de sus proyecciones:

$$m_1 - m_2 = a^\top (\mu_1 - \mu_2) \quad (7.20)$$

donde $m_k = a^\top \mu_k$ corresponde al promedio de los elementos de la clase \mathcal{C}_k proyectado sobre el vector a (centro de masa sobre la recta). Consecuentemente, el vector a que maximiza la distancia entre la proyección de clases es el que maximiza la expresión anterior. Sin embargo, esta expresión puede ser arbitrariamente grande si escalamos a , por lo que se fijará $\|a\|_2 = 1$. Además, por la desigualdad de Cauchy-Schwarz, $|a^\top (\mu_1 - \mu_2)| \leq \|a\| \|\mu_1 - \mu_2\|$ con igualdad si y solo si los vectores son paralelos. De este modo, se llega a que $a \propto (\mu_1 - \mu_2)$.

Una desventaja de este enfoque, en el cual se ha ignorado la dispersión de las clases y solo se ha considerado su media, es que pueden existir 2 clases bien separadas en el espacio M -dimensional, pero que al proyectar los datos sobre la recta que une sus promedios, las proyecciones de cada clase se traslanen.

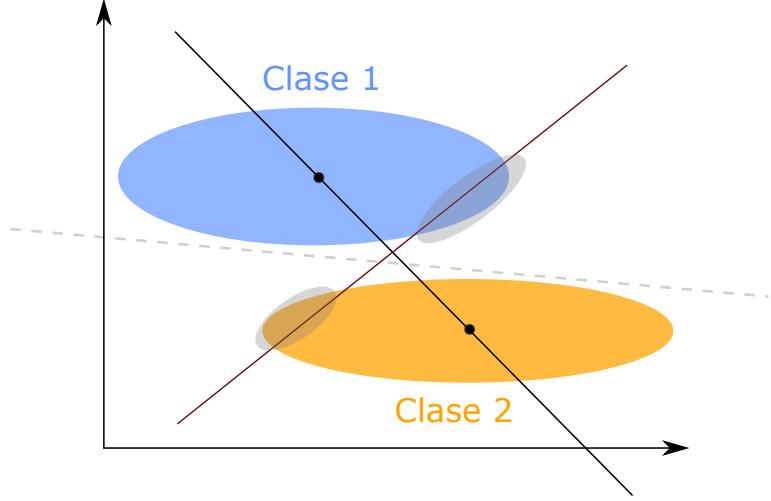


Fig. 39. Superposición de las proyecciones al considerar únicamente la recta que une las medias de clase. La recta roja determina la región de decisión y la recta segmentada muestra un posible hiperplano separador.

Para resolver este problema, Fisher propuso maximizar no solo distancia entre las (medias de las) clases proyectadas, sino que adicionalmente minimizar la dispersión de los elementos de una misma clase, con el objetivo de disminuir el traslape entre las proyecciones de las clases. Como medida de dispersión, definimos la varianza muestral proyectada de los elementos de la clase \mathcal{C}_k mediante

$$s_k^2 = \sum_{n \in \mathcal{C}_k} (a^\top (x_n - \mu_k))^2 \quad (7.21)$$

$$= \sum_{n \in \mathcal{C}_k} (y_n - m_k)^2, \quad (7.22)$$

Donde el factor de corrección $\frac{1}{N_k - 1}$ fue omitido ya que de lo contrario, todas las clases pesarían lo mismo sin importar la cantidad de elementos de la clase. Lo anterior nos permite definir la siguiente función objetivo

$$J(a) = \frac{m_1 - m_2}{s_1^2 + s_2^2}, \quad (7.23)$$

donde explícitamente vemos la discrepancia “inter” clases en el numerador y la discrepancia “intra” clases en el denominador. Adicionalmente, podemos expresar este costo directamente como función del vector de proyección a :

$$J(a) = \frac{a^\top S_B a}{a^\top S_W a}, \quad (7.24)$$

donde la matriz de covarianza entre clases S_B y matriz total de covarianza dentro de clases S_W están respectivamente dadas por

$$S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^\top \quad (7.25)$$

$$S_W = \sum_{n \in \mathcal{C}_1} (x_n - \mu_1)(x_n - \mu_1)^\top + \sum_{n \in \mathcal{C}_2} (x_n - \mu_2)(x_n - \mu_2)^\top. \quad (7.26)$$

Aplicando la condición de primer orden para $J(a)$, obtenemos que el vector a óptimo debe cumplir

$$(a^\top S_B a)S_W a = (a^\top S_W a)S_B a. \quad (7.27)$$

Sin embargo, notemos que la norma del vector a es irrelevante, solo interesa su orientación, con lo que ignorando los escalares $(a^\top S_B a)$ y $(a^\top S_W a)$ tenemos que la relación de optimalidad es $S_W a \propto S_B a$.

Además, por la definición de S_B , sabemos que $S_B a \propto (\mu_1 - \mu_2)$, con lo que la relación de optimalidad se convierte en es $S_W a \propto (\mu_1 - \mu_2)$. Consecuentemente, el vector optimo a en el criterio de Fisher debe cumplir

$$a \propto S_W^{-1}(\mu_1 - \mu_2). \quad (7.28)$$

La Figura 37 muestra el discriminador lineal que solo considera los promedios a la izquierda y la corrección de Fisher a la derecha. Observemos cómo el incluir una medida de la dispersión de los datos es clave para lograr un mejor discriminador.

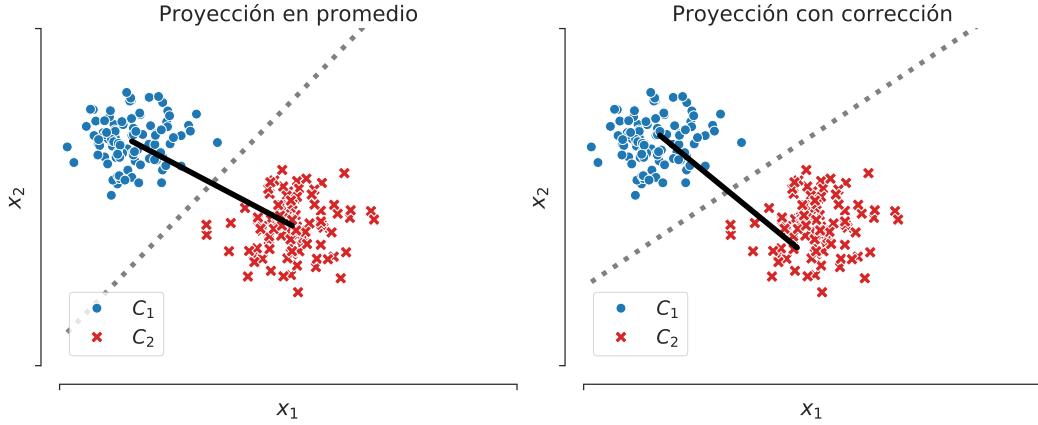


Fig. 40. Discriminador lineal considerando solo la media entre clases (izquierdo) y su extensión mediante la corrección de Fisher que incorpora la varianza muestral de los datos.

7.1.5. Compressed Sensing

Compressed Sensing es una técnica de reducción de dimensionalidad basada en la hipótesis previa de que el vector original de los datos es *sparse* en alguna base.

Como motivación, sea $x \in \mathbb{R}^d$ con a lo más s coordenadas distintas de 0. Es decir:

$$\|x\|_0 = |\{i : x_i \neq 0\}| \leq s.$$

En esta situación, x puede ser comprimido si es representado por s pares de la forma (*índice, valor*). Además, esta compresión no tiene pérdidas sobre la información de x , pues este vector se puede reconstruir exactamente en base a estos pares.

Ahora, dada U una matriz ortonormal fija, supongamos que $x = U\alpha$, con α un vector sparse, i.e $\|\alpha\|_0 \leq s$. Entonces, estamos suponiendo que existe otra base de \mathbb{R}^d bajo la cual x es un vector *sparse*. Si bien esta suposición puede parecer ingenua en un inicio, muchos vectores resultan cumplir esta propiedad. Ahora, en este caso ¿Es posible comprimir x usando sólo s pares como los mencionados anteriormente?

Una forma simple de hacer lo anterior es multiplicar x por U^T , y luego representar el vector resultante α por los pares deseados. Sin embargo, este proceso requiere primero guardar x , y luego multiplicar por U^T , lo que naturalmente levanta la pregunta de si es realmente necesario guardar una gran cantidad de información que luego será desechada. ¿Será posible medir lo que no será desecharlo y trabajar con esta parte de x directamente?

La respuesta es si, y está basada en un resultado clave: Una transformación lineal x puede comprimir x sin pérdida de información. Para profundizar esto, se requiere la siguiente definición:

Definición 7.1. Una matriz $U \in \mathbb{R}^{n \times d}$ es (ϵ, s) - RIP (*Restricted Isoperimetric Property*) si para todo $x \neq 0$, tal que $\|x\|_0 \leq s$, se cumple:

$$\left| \frac{\|Ux\|_2^2}{\|x\|_2^2} - 1 \right| \leq \epsilon$$

Teniendo en cuenta lo anterior, el siguiente Teorema es la razón por la cuál esta técnica tiene sentido.

Teorema 7.1. Sea $\epsilon < 1$ y $U \in \mathbb{R}^{n \times d}$ una matriz $(\epsilon, 2s)$ -RIP. Sea $x \neq 0$ un vector en \mathbb{R}^d tal que $\|x\|_0 \leq s$, e $y = Ux$ la compresión de x . Entonces el vector reconstruido:

$$\bar{x} \in \arg \min_{v: Wv=y}$$

cumple $\bar{x} = x$.

Luego, basándonos el Teorema anterior, es posible comprimir x sin pérdida alguna, siempre y cuando logremos encontrar la matriz U adecuada.

7.1.6. PCA vs. Compressed Sensing

Teniendo en cuenta los dos métodos de reducción de dimensionalidad aquí mencionados, es natural preguntarse cuál de ellos se debe usar en diferentes situaciones.

En primer lugar, es importante recordar que si se cumplen las suposiciones de cada técnica, es posible reconstruir perfectamente la información de dimensión reducida.

La pregunta clave es qué es lo que se quiere comprimir. Si son vectores que cumplen con tener una gran cantidad de 0, compressed sensing será probablemente una mejor opción que PCA. Un ejemplo podría ser trabajar con vectores canónicos en \mathbb{R}^d . Por otra parte, si no se tiene mucha información con respecto a lo que se quiere comprimir, es mejor ocupar PCA pues es un método más efectivo cuando los datos tienen ruido. Por otra parte, si tenemos N datos que están exactamente en un subespacio de dimensión n (por ejemplo un hiperplano), PCA es la alternativa recomendada pues nada asegura que se cumplan las condiciones para hacer compressed sensing.

7.1.7. Otros métodos de reducción de dimensionalidad

- **Escalamiento Multidimensional**

Se busca reducir la dimensión intentando preservar la distancia entre las instancias.

En términos prácticos, se comienza con $x_1, \dots, x_n \in \mathbb{R}^N$, y d_{ij} la distancia entre x_i y x_j . Frecuentemente esta distancia es la distancia euclídea, sin embargo se pueden elegir también medidas de disimilitud entre los inputs.

El escalamiento multidimensional busca encontrar $z_1, \dots, z_k \in \mathbb{R}^p$ que minimicen la función de stress:

$$S_M(z_1, \dots, z_k) = \sum_{i \neq j} (d_{ij} - \|z_i - z_j\|)^2.$$

Este escalamiento se conoce como de mínimos cuadrados o de Kruskal–Shephard. Es importante notar que la forma de la función de stress sugiere que un algoritmo de descenso de gradiente es una buena herramienta para minimizarla.

- **t-Distributed Stochastic Neighbor Embedding (t-SNE)**

Se busca reducir la dimensionalidad buscando dejar instancias similares cerca y distancias disimilares lejos. Se suele ocupar para visualización, sobre todo para visualizar clusters en data de altas dimensiones.

Este método consta, a grandes rasgos, de dos pasos:

1. En primer lugar, se genera una distribución sobre parejas de inputs, de forma tal que parejas de puntos que se encuentren cerca tengan alta probabilidad.
2. Se usa la divergencia de Kullback-Liebler para intentar reproducir la distribución del espacio de alta dimensión en el espacio de baja dimensión.

- **Random Projections**

Otra forma de reducir la dimensionalidad del conjunto de datos es usando proyecciones aleatorias. *A priori* esta idea pareciera no tener sentido, sin embargo el Teorema de Johnson-Lindestrauss indica justo lo contrario: Es altamente probable que al usar una proyección aleatoria se preserve la distancia entre puntos cercanos.

Teorema 7.2 (Teorema de Johnson-Lindestrauss). *Sea V un conjunto finito de vectores en \mathbb{R}^d . Sea $\delta \in (0, 1)$ y n un entero tal que*

$$\epsilon = \sqrt{\frac{6 \log(2|V|\delta)}{n}} \leq 3.$$

Entonces con probabilidad al menos $1 - \delta$, si se elige aleatoriamente una matriz U con cada coordenada distribuida como una normal $\mathcal{N}(0, \frac{1}{n})$ se cumple:

$$\sup_{x \in V} \left| \frac{\|Ux\|_2^2}{\|x\|_2} - 1 \right| < \epsilon$$

Observación 7.1. Este método suele ser útil cuando se intenta pasar de dimensiones "muy altas" a dimensiones "medianas", pero no así para pasar a dimensiones "bajas".

7.2. Clustering

Otro de los problemas principales del aprendizaje no supervisado es poder agrupar las entradas sin necesidad de que estas tengan etiquetas (como ocurre en los métodos de clasificación). La dificultad principal de esta tarea es que es necesario definir un concepto de similitud entre las distintas entradas para poder particionar los datos en grupos (clusters) de características similares.

7.2.1. Hierarchical clustering (HCA)

Corresponde al algoritmo de clustering más simple pero a su vez, es uno de los más caros computacionalmente. El objetivo es identificar k clusters en los datos.

El algoritmo aglomerativo de clustering es el siguiente:

1. Para comenzar, se considerarán n clusters distintos. Asignar a cada elemento un cluster único.
2. Buscar el par de clusters más similar (bajo algún criterio) y combinarlos en un solo cluster.
3. Repetir el paso anterior hasta tener k clusters.

Para dos clusters $A, B \subset \mathcal{D}$, los criterios de similitud más frecuentes son los siguientes:

- **Single-linkage clustering:** $D_s(A, B) := \min\{d(a, b) : a \in A, b \in B\}$.
- **Complete-linkage clustering:** $D_s(A, B) := \max\{d(a, b) : a \in A, b \in B\}$.
- **Average-linkage clustering:** $D_a(A, B) := \frac{1}{|A| \cdot |B|} \sum_{a \in A, b \in B} d(a, b)$.

Donde $d : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}_+$ es una métrica en \mathcal{D} . Elecciones distintas del criterio de similitud y/o métrica (generalmente euclíadiana) pueden llevar a agrupaciones distintas.

7.2.2. kmeans

Dado un entero $k \in \mathbb{N}$ y un conjunto de observaciones $X = \{x_i\}_{i=1}^N$ con $x_i \in \mathbb{R}^D$ queremos separar los datos en k grupos, donde cada grupo se le asigna un centroide μ_k y cada elemento x_i se le asigna el grupo que tenga el centroide más cercano.

Sea r_{ik} la asignación, esta estará definida por:

$$r_{ik} = \begin{cases} 1 & \text{si } k = \operatorname{argmin} \|x_i - x_k\| \\ 0 & \text{si no.} \end{cases} \quad (7.29)$$

Es decir, para encontrar los centroides se debe minimizar el funcional:

$$J = \sum_{i=1}^N \sum_{k=1}^K r_{ik} \|x_i - x_k\|^2 \quad (7.30)$$

Para minimizar esta función utilizaremos un enfoque llamado *Expectation-Maximization*. Este es un método iterativo y como tal, tiene problemas con mínimo locales, pero para solucionar esto, basta inicializar el algoritmo muchas veces.

El algoritmo más frecuente usado para k-means es el algoritmo de Lloyd:

- **E-step:** En este paso, se calculan (actualizan) las asignaciones r_{ik} , dejando fijos μ_k . Lo que corresponde a asignar el dato x_i al centroide más cercano.
- **M-step:** El siguiente paso corresponde a actualizar los centroides μ_k dejando fijo las asignaciones r_{ik} .

Como J es cuadrática en μ_k , entonces podemos utilizar la condición de primer orden:

$$\mu_k = \frac{\sum_{i=1}^N r_{ik} x_i}{\sum_{i=1}^N r_{ik}} \quad (7.31)$$

Lo que corresponde a asignar el centro del cluster al promedio de todas las muestras asignadas al antiguo cluster.

El algoritmo termina cuando los centroides ya no cambian.

Para los centroides iniciales, se tienen dos posibles inicializaciones:

- **Método de Forgy:** se eligen de forma aleatoria k puntos de la muestra como centroides iniciales.

- **Random partition:** se eligen asignaciones aleatorias para los elementos. De este modo, los centroides iniciales serán los centroides obtenidos al realizar M-step.

El método de Forgy es preferido cuando se realiza k-means mediante el algoritmo de Lloyd.

Ejemplo: En la figura 41 se observa un ejemplo de clustering utilizando kmeans. Los clusters creados por kmeans son circulares, puesto que se utiliza distancia euclídea hacia el centro del cluster.

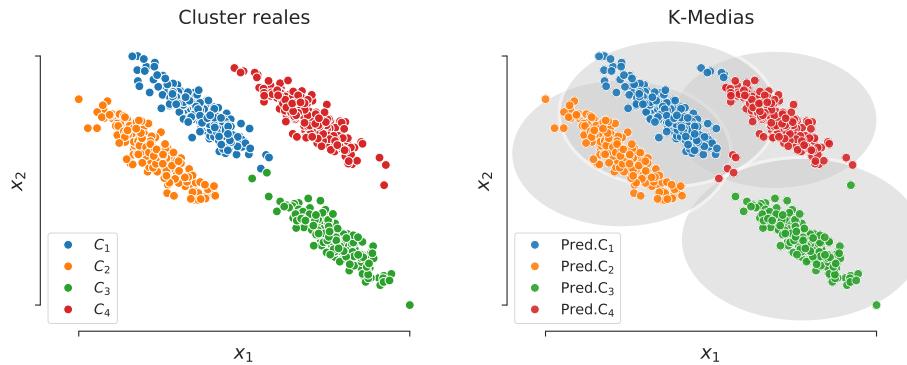


Fig. 41. (Izquierda) Datos reales con sus etiquetas correctas. (Derecha) Clusters encontrados por k-means.

Por otra parte, la asignación de cluster mediante el centroide más cercano provoca que cada par de centroides divida al espacio ambiente en dos semiespacios mediante el hiperplano simétrico que pasa entre ambos puntos. Luego, dado que la intersección finita de semiespacios genera un poliedro, se tiene que la partición generada por kmeans forma un diagrama de Voronoi.

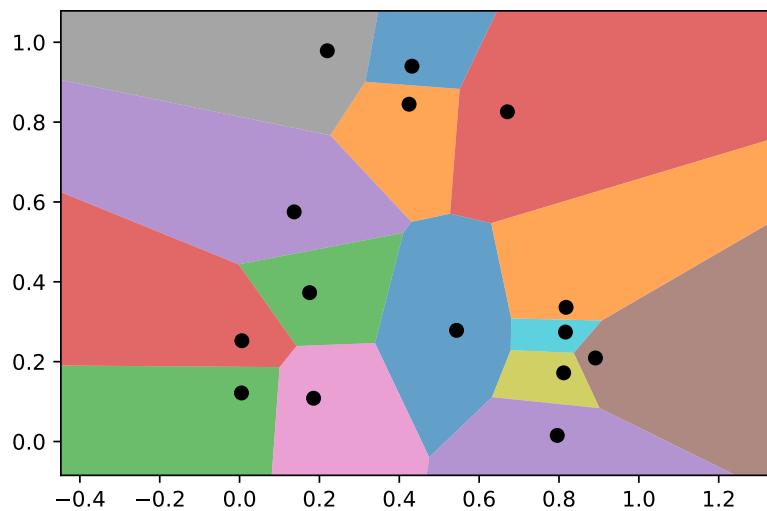


Fig. 42. Partición de \mathbb{R}^2 inducida por los centroides.

7.2.3. Modelo de mezcla de gaussianas (GMM)

La mezcla de gaussianas (GMM) es un caso general de k -means, en donde los clústeres pueden tener una forma anisotrópica modelada por una Gaussiana con covarianza no necesariamente proporcional a la identidad. Además, si bien su solución puede ser muy similar a k -means los supuestos subyacentes al

modelo CMM son diferentes y obedecen a un enfoque de modelo generativo.

Una distribución de mezcla de gaussianas consiste en una combinación convexa de distribuciones gaussianas

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \mu_k, \Sigma_k) \quad (7.32)$$

Donde una muestra x es generada mediante dos etapas: primero se elige un cluster al azar y luego, se genera una muestra aleatoria dentro del cluster. Nos referiremos a los parámetros de este modelo como

- π_k : coeficiente de mezcla del cluster k (probabilidad de venir del cluster k).
- μ_k : media del cluster k .
- Σ_k : matriz de covarianza del cluster k .

Para ver que la formulación anterior es la indicada para la generación de \mathbf{x} , se puede asignar una variable latente $\mathbf{z}(\mathbf{x}) = \mathbf{z} = [z_1, z_2, \dots, z_K]^\top \in \{0, 1\}^K$, que describa la asignación de \mathbf{x} a cada cluster, es decir, $z_k = 1$ si la observación \mathbf{x} pertenece al cluster k , y $z_k = 0$ si no. Bajo esta notación, podemos denotar la distribución marginal del cluster k (es decir, de que una observación sea generada por la k -ésima componente) como

$$p(z_k = 1) = \pi_k \quad (7.33)$$

Además, para una muestra \mathbf{x} proveniente del cluster k se tiene que

$$p(\mathbf{x}|z_k = 1) = \mathcal{N}(\mathbf{x} | \mu_k, \Sigma_k) \quad (7.34)$$

Por lo tanto, usando probabilidades totales:

$$p(\mathbf{x}) = \sum_{k=1}^N p(z_k = 1)p(\mathbf{x}|z_k = 1) = \sum_{k=1}^N \pi_k \mathcal{N}(\mathbf{x} | \mu_k, \Sigma_k) \quad (7.35)$$

Lo cual prueba que el modelo sugerido al comienzo es el indicado para una mezcla de gaussianas.

Recordemos que los parámetros del modelo de mezcla de gaussianas son los coeficientes de mezcla, las medias y varianzas de cada componente. Si disponemos de un conjunto de observaciones $\{\mathbf{x}_i\}_{i=1}^N$, entonces podemos encontrar dichos parámetros mediante máxima verosimilitud. Observe que la log-verosimilitud está dada por

$$\log p(\mathbf{x}_1, \dots, \mathbf{x}_N) = \sum_{n=1}^N \log p(\mathbf{x}_n) = \sum_{n=1}^N \log \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \mu_k, \Sigma_k) \quad (7.36)$$

Hay una serie de complicaciones relacionadas a la búsqueda de estos parámetros mediante la maximización de la ecuación (7.36). Primero, están las soluciones dadas por singularidades cuando una muestra es exactamente igual a una de las medias, en cuyo caso un término de la log-verosimilitud es proporcional a $1/\sigma_k$, con lo que la maximización de σ_k resulta en una log-verosimilitud infinita. En segundo lugar tenemos las redundancias de soluciones: para cada máximo local (o solución en general) de la log-verosimilitud existen $k!$ soluciones equivalentes con la misma verosimilitud dadas por las permutaciones de las etiquetas de los clusters. Finalmente, optimizar la log-verosimilitud de la mezcla de gaussianas es desafiante porque la sumatoria aparece *dentro* del logaritmo, consecuentemente, el logaritmo no actúa directamente en la gaussiana reduciendo el funcional de optimización a una solución en forma cerrada. Por esta razón, es necesario considerar métodos basados en gradiente.

Entrenamiento de una GGM Veamos las condiciones de primer orden sobre la log-verosimilitud para encontrar los parámetros. Denotando $\gamma(z_k(\mathbf{x}_i)) = p(z_k(\mathbf{x}_i) = 1 | \mathbf{x}_i)$, se tiene el siguiente resultado:

Proposición 7.2.1. *Para el modelo GGM, los parámetros óptimos son*

$$\mu_k = \frac{1}{R_k} \sum_{i=1}^N \gamma(z_k(\mathbf{x}_i)) \mathbf{x}_i \quad (7.37)$$

$$\Sigma_k = \frac{1}{R_k} \sum_{i=1}^N \gamma(z_k(\mathbf{x}_i)) (\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^\top \quad (7.38)$$

$$\pi_k = \frac{R_k}{R}, \quad (7.39)$$

donde $R_k = \sum_{i=n}^N \gamma(z_k(\mathbf{x}_i))$ y $R = \sum_{k=1}^K R_k$.

Observe que esto no constituye una solución en forma cerrada para los parámetros, pues la posterior $\gamma(z_k(\mathbf{x}_i))$ depende de todos los parámetros, en efecto, gracias al teorema de Bayes:

$$p(z_k(\mathbf{x}) = 1 | \mathbf{x}) = \frac{p(\mathbf{x}|z_k(\mathbf{x}) = 1)p(z_k(\mathbf{x}) = 1)}{p(\mathbf{x})} = \frac{\pi_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)}{\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)} \quad (7.40)$$

Sin embargo, podemos considerar un procedimiento iterativo en donde calculamos las posteriores $\gamma(z_k(\mathbf{x}_i))$ (llamado paso E), para luego calcular los parámetros óptimos de acuerdo a las ecuaciones anteriores (llamado paso M).

La Figura 43 muestra un ejemplo de clustering utilizando GMM. En este caso, se puede observar directamente como GMM es una generalización de k -means, en donde ahora los clusters tienen forma de gaussiana anisotrópica.

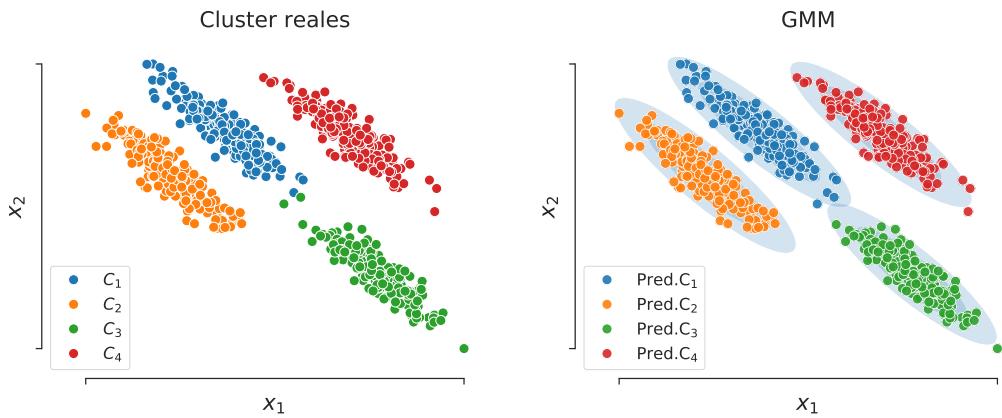


Fig. 43. (Izquierdo) Datos reales con sus etiquetas correctas. (Derecha) Clusters encontrados por GMM.

Expectation-maximization

En el caso general, tenemos *observables* \mathbf{x} , variables latentes \mathbf{z} y parámetros θ , y un modelo descrito por una distribución marginal que puede ser expresada mediante

$$\log p(\mathbf{x}|\theta) = \log \int p(\mathbf{x}, \mathbf{z}|\theta) d\mathbf{z} \quad (7.41)$$

donde el logaritmo de sumas es complicado de optimizar, incluso cuando la distribución conjunta $p(\mathbf{x}, \mathbf{z}|\theta)$ está en la familia exponencial.

Asumamos por un momento que tenemos valores para la variable latente \mathbf{z} , si este fuese el caso, podríamos buscar los parámetros mediante la optimización de la log-verosimilitud completa, $\log p(\mathbf{x}, \mathbf{z}|\theta)$, la cual como en GMM puede tener una forma más simple de optimizar debido a que no hay una suma dentro del logaritmo.

Sin embargo, en la práctica no tenemos acceso al valor de las variables latentes \mathbf{z}_n sino que únicamente podemos acceder a una estimación de estas mediante la distribución posterior $p(\mathbf{z}|\mathbf{x}, \theta)$. Entonces, estrictamente hablando, la cantidad que nos gustaría maximizar $\log p(\mathbf{x}, \mathbf{z}|\theta)$ es aleatoria, por lo que podemos maximizar su esperanza con respecto a las observaciones (etapa de maximización). Luego, con los valores obtenidos para los parámetros podemos recalcular la distribución posterior $p(\mathbf{z}|\mathbf{x}, \theta)$ (etapa de esperanza) y seguir este proceso iterativamente. La motivación de este procedimiento, llamado *expectation-maximization*, es que en primer lugar, con la cantidad $p(\mathbf{z}|\mathbf{x}, \theta)$ fija, el cálculo de los nuevos parámetros mediante máxima verosimilitud indiscutiblemente aumenta la verosimilitud, luego estos mejores parámetros dan consecuentemente una mejor estimación de la distribución posterior $p(\mathbf{z}|\mathbf{x}, \theta)$, con lo cual la actualización de los parámetros usando esta mejorada aproximación de la posterior debe ser incluso mejor.

7.2.4. Density-based spatial clustering of applications with noise (DBSCAN)

Es un algoritmo de clustering propuesto por Martin Ester et al. el cual ha tenido mucha popularidad puesto que no requiere definir una cantidad inicial de clusters. Los hiper-parámetros de entrada del modelo son 2:

- Mínimo número de punto: $minPts$.
- Radio o vecindad: ϵ .

El algoritmo se basa en la idea de que dado 2 puntos x_i, x_j dentro de un mismo cluster, se dice que x_i es *alcanzable por* x_j , si siempre se puede llegar de x_i a x_j avanzando de punto en punto, donde la distancia entre cada uno es a lo más ϵ y además un punto intermedio es un punto núcleo. Con estos el algoritmo define tres tipos de puntos:

- **Puntos núcleo:** Son puntos x_i tales que en una vecindad ϵ tienen al menos $minPts$ vecinos.
- **Puntos borde:** Constituyen el *borde externo* de los cluster.
- **Outliers:** Puntos que no son alcanzables por ningún punto.

Notemos que con lo anterior, se desprende que todo cluster debe tener al menos un punto núcleo. El algoritmo para encontrar los clusters es el siguiente:

Algoritmo 7 Pseudo código de DBSCAN

```
1: function DBSCAN( $D, \text{eps}, \text{MinPts}$ )
2:    $C \leftarrow 0$ 
3:   for cada punto  $P$  no visitado en  $D$  do
4:     marcar  $P$  como visitado
5:     if sizeOf(PuntosVecinos)  $\leq \text{MinPts}$  then
6:       marcar  $P$  como RUIDO
7:     else
8:        $C \leftarrow C + 1$ 
9:       expandirCluster( $P, \text{vecinos}, C, \text{eps}, \text{MinPts}$ )
```

Algoritmo 8 Función para expandir cluster.

```
1: function EXPANDIRCLUSTER( $P, \text{vecinosPts}, C, \text{eps}, \text{MinPts}$ )
2:   agregar  $P$  al cluster  $C$ 
3:   for cada punto  $P'$  en vecinosPts do
4:     if  $P'$  no fue visitado then
5:       marcar  $P'$  como visitado
6:       vecinosPts'  $\leftarrow$  regionDeConsulta( $P', \text{eps}$ )
7:       if sizeof(vecinosPts')  $\geq \text{MinPts}$  then
8:         vecinosPts  $\leftarrow$  vecinosPts  $\cup$  vecinosPts'
9:       if  $P'$  no tiene cluster asignado then
10:         $P'$  se le asigna el cluster  $C$ 
```

Algoritmo 9 Retorna los puntos de la vecindad de búsqueda para un punto.

```
1: function REGIONDECONSULTA( $P, \text{eps}$ )
   return Todos los puntos junto a  $P'$  que están a  $\text{eps}$  de distancia (incluyendo  $P$ )
```

La figura 44 muestra un ejemplo de clustering utilizando DBSCAN. A la derecha se muestran en negro los puntos que son clasificados como ruido o *outliers* por el algoritmo. Por otro lado, los puntos núcleos son graficados como un punto grande, mientras que los puntos borde se grafican con un marcador pequeño.

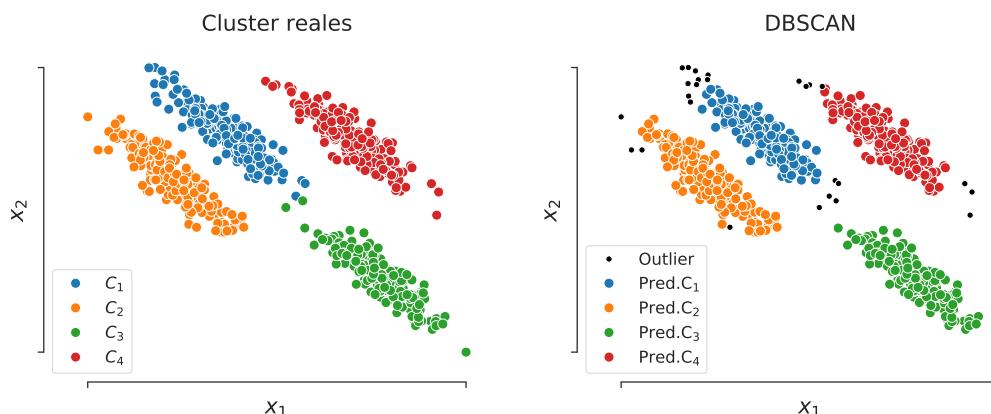


Fig. 44. Datos reales con sus etiquetas correctas (izquierda) y clusters encontrados por DBSCAN (derecha).

7.2.5. Aprendizaje Semi-Supervisado

Como el lector debe haber notado a este punto, todos los modelos anteriores son útiles siempre y cuando las características que se den como input al modelo permitan resumir y estudiar los datos de buena manera. Sin embargo, esta tarea no es nada fácil: Muchas veces los *features* que permiten llegar a un modelo son desconocidos o muy difíciles de encontrar.

Una propuesta para solucionar la problemática anterior es usar métodos de aprendizaje no supervisado para obtener *features*, y usarlos como input de un modelo supervisado. En otras palabras, se ocupa un modelo no supervisado para pre-procesar los datos. Es así que surge así el término de aprendizaje semi-supervisado.

Existen casos en la bibliografía donde modelos supervisados generalizan mejor cuando hay preprocesamiento mediante métodos no supervisados.

En particular, es muy común ocupar K-Means como preprocesamiento. Este proceso se sintetiza en los siguientes pasos:

1. Encontrar un número de clusters óptimos para el conjunto de entrenamiento.
2. Calcular la distancia a cada cluster para todo el conjunto de entrenamiento.
3. Usar las distancias a cada cluster como input a un modelo supervisado (por ejemplo, una regresión logística).

7.2.6. Otros Algoritmos de Clustering

Los algoritmos de clustering que fueron presentados son sólo los más comunes. Sin embargo, existen otros algoritmos que pueden ser útiles en la práctica. En esta sección mencionaremos brevemente algunos de ellos.

- **BIRCH:**

El algoritmo BIRCH (*Balanced Iterative Reducing and Clustering using Hierarchies*) fue diseñado específicamente para datasets muy grandes. Es más rápido que K-Means con batches, y con resultados parecidos con una cantidad de features baja (< 20). Su innovación es en el uso de memoria para asignar un dato a un cluster en particular.

- **Spectral Clustering:** Este algoritmo crea una matriz de similitud entre los datos, y reduce su dimensionalidad. Luego de esto, se ocupa otro algoritmo de clustering (por ejemplo, K-Means) en este espacio de dimensión baja.

Este algoritmo puede encontrar relaciones muy complejas en los datos, sin embargo no escala bien a grandes cantidades de datos, ni tampoco se comporta bien cuando los clusters están desbalanceados.

8. Redes Neuronales

8.1. Introducción y arquitectura

8.1.1. Conceptos básicos

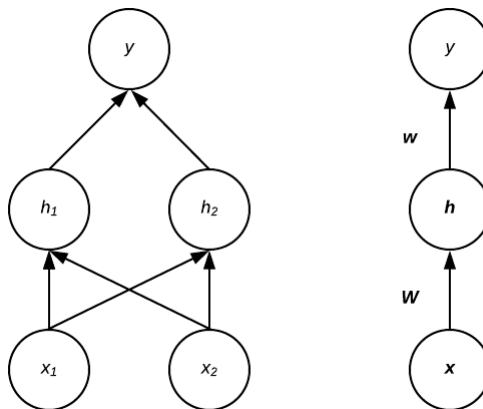
Una red neuronal es un modelo de aprendizaje de máquinas que, en sus inicios, estaba inspirado en el funcionamiento de las neuronas en nuestro cerebro. A medida que se ha desarrollado la teoría en torno a las redes neuronales, este modelo se ha ido alejando progresivamente de su semejante biológico. Es más, investigadores argumentan que se debería dejar de lado el concepto de neurona pues es demasiado restrictivo en cuanto a los sistemas que podemos crear.

Las redes neuronales son modelos que sirven tanto para clasificación como para regresión. Para ambos objetivos, su funcionamiento es el mismo.

La base de las redes neuronales reside en la conexión de múltiples unidades, llamadas neuronas, cuyo objetivo es aproximar una función f^* , más específicamente, definir un mapping $y = f(x; \theta)$ aprendiendo los parámetros θ que resultan en la mejor aproximación posible. Su gran ventaja radica en la posibilidad de entrenarlas con datos crudos, es decir, no es necesario extraer características relevantes de la data y en vez, se permite que el algoritmo aprenda cuales son.

Los modelos esenciales de redes neuronales se conocen como **feedforward neural networks**, o **multilayer perceptrons** (MLPs) y se conocen como redes ya que son típicamente el resultado de composiciones sucesivas de varios tipos de funciones $f^{(1)}, f^{(2)}, f^{(3)}, \dots, f^{(l)}$, se dice que $f^{(i)}$ es la i -ésima capa de la red, mientras que l , la cantidad de capas, se le conoce como **profundidad**. A las capas intermedias se les conoce como **capas ocultas** y el uso de redes neuronales con una múltiples capas se le conoce como **deep learning**, aunque este término se usa para problemas más específicos de aprendizaje de máquinas usando redes neuronales profundas (ver sección 3.5).

Fig. 45. Ejemplo de una red neuronal de 1 capa: (*izquierda*) Se muestra la red con inputs x_1 y x_2 , que luego pasan a la capa oculta para producir el output y . (*derecha*) Misma red con una representación vectorial de las capas. La matriz \mathbf{W} describe el mapping de \mathbf{x} a \mathbf{h} , y la matriz w el mapping de \mathbf{h} a y .



Como se puede apreciar en la figura, los coeficientes \mathbf{W} y w se usan para producir el output para la siguiente capa (denominados **pesos**), por lo que la primera operación de esta red será entregar h_1 y h_2 mediante la transformación $\mathbf{W}^T \mathbf{x} + b^{(1)}$. Luego, esta red aplica $w^T \mathbf{h} + b^{(2)}$ para producir el output y . Los coeficientes $b^{(1)}$ y $b^{(2)}$ se conocen como términos de **bias** (sesgo). Todos los parámetros de la red neuronal, los pesos y *bias*, serán agrupados en el término θ .

8.1.2. El perceptrón y funciones de activación

El *perceptrón* corresponde a la forma más básica de una red neuronal. Este recibe un input numérico $x = (x_i)_{i=1}^n \in \mathbb{R}^n$ y computa la suma ponderada $u = x_1w_1 + x_2w_2 + \dots + x_nw_n + b$ donde $W = (w_i)_{i=1}^n \in \mathbb{R}^n$ corresponden a los **pesos** (weights) y $b \in \mathbb{R}$ el **sesgo** (bias). A continuación, se aplica una función de activación f y se entrega un output $h = f(u)$.

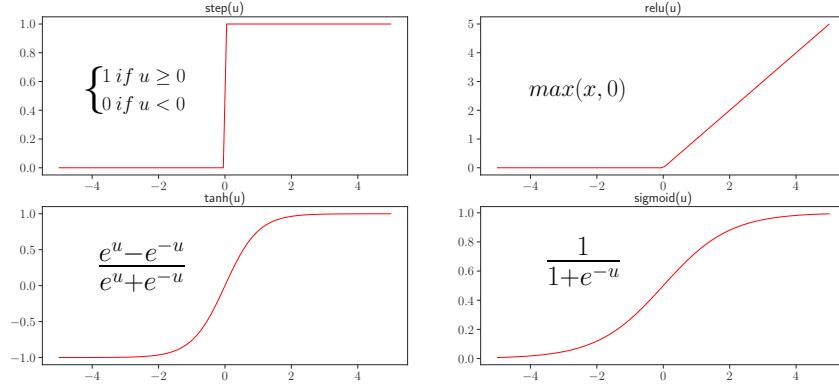


Fig. 46. Algunos ejemplos de funciones de activación

Esta idea de tomar combinaciones lineales del input, y luego aplicar una función no lineal se puede extender para interconectar perceptrones. Esto es lo que se conoce como *multilayer perceptrons*. Las funciones de activación son en general transformaciones no lineales que buscan dar la flexibilidad necesaria al modelo para aprender funciones extremadamente complejas. Esto se sustenta en el siguiente teorema

Teorema 8.1 (UAT - Ancho arbitrario (Hornik et al., 1989; Cybenko, 1989)). *Sea f^* una función objetivo Borel Medible (por ejemplo $f^* : [0, 1]^k \rightarrow [0, 1]$ continua con $k \in \mathbb{N}$) y sea f una función de activación no polinomial acotada, entonces para todo $\epsilon > 0$ existen W, b definidas como antes y U la matriz de pesos de la última capa tales que*

$$F(x) = f(xW + b)U, \quad |F(x) - f^*(x)| < \epsilon \quad \forall x \in \text{Dom}(f^*)$$

Es decir, para una red de una sola capa escondida con la suficiente cantidad de unidades, es posible aproximar una función objetivo razonable tan finamente como se desee. Notar que lo anterior es un resultado que considera el ancho (cantidad de perceptrones en la capa) y que este podría ser arbitrariamente grande. Además, la red podría tener una baja capacidad de generalización dependiendo de la complejidad de los datos con los que se pretende trabajar. Es por esto que surge la necesidad de arquitecturas más complejas, es decir, con mayor cantidad de capas (mayor profundidad) pero menor cantidad de perceptrones y por suerte, UAT para profundidad arbitraria también es válido.

Veamos el siguiente ejemplo:

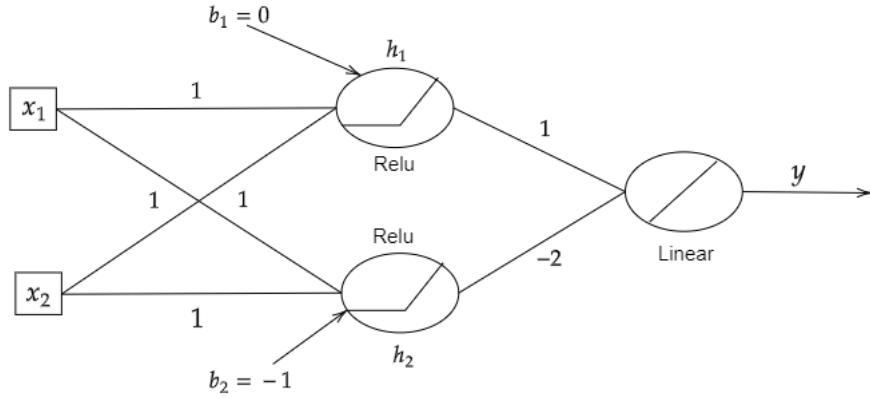


Fig. 47. Red Neuronal de 2 perceptrones para XOR

En su forma matricial

$$(h_1, h_2) = \text{Relu} \left((x_1 \ x_2) \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} + (0 \ -1) \right) \quad h = \text{Relu}(xW + b)$$

$$y = (h_1 \ h_2) \begin{pmatrix} 1 \\ -2 \end{pmatrix} + (0) \quad y = hU + c$$

Donde U y c corresponden al peso y bias de la última capa respectivamente, es una red capaz de computar el operador XOR.

Es importante notar que UAT indica que es posible aproximar una función objetivo razonable, pero no entrega una receta para ello. Es por ello que definir la arquitectura de una red neuronal no es un problema trivial. Si bien existen heurísticas para definir estas arquitecturas para distintos problemas, no existe (hasta el momento) una receta que permita encontrar la arquitectura correcta.

8.1.3. Arquitectura de una red neuronal

Como se vió anteriormente, no basta con una red arbitrariamente ancha para lograr el objetivo de aproximar f^* con una alta capacidad de generalización. La **arquitectura** de una red neuronal se refiere a la totalidad de su estructura, es decir, la cantidad de capas, perceptrones por capa, conexión entre unidades, funciones de activación, etc...

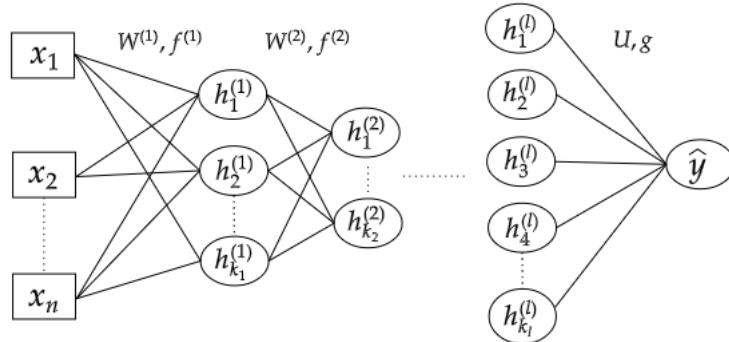


Fig. 48. Una red de profundidad l

Vamos a utilizar la siguiente notación a partir de este punto

$$h^{(k)} = f^{(k)}(h^{(k-1)}W^{(k)} + b^{(k)}) \quad \forall k \in \{1, \dots, l\}, \quad h^{(0)} = x \quad (8.1)$$

$$\hat{y} = g(h^{(l)}U + c) \quad (8.2)$$

Donde g es la función aplicada en la capa de output y es la que define la **unidad de output**.

8.1.4. Función de costos y unidades de output

Una de las principales diferencias entre los modelos lineales antes vistos y una red neuronal, es que el uso de ciertas funciones de activación hacen que la función de costos no sea convexa. Esto hace que el entrenamiento realizado en base a descenso de gradiente no entregue garantías de que se alcanzará el óptimo global, o una buena solución en términos generales. Esto pues el algoritmo podría estancarse en un óptimo local que entregue resultados pobres.

Si bien ocupar mínimos cuadrados como función de costos para este tipo de modelos es posible, se suelen ocupar otras funciones de costo.

En la mayoría de los casos, el modelo paramétrico define una distribución $p(y|\mathbf{x}; \boldsymbol{\theta})$, por lo que los parámetros del modelo se estimarán usando máxima verosimilitud, así, se optimizará la log-verosimilitud negativa. Así, en estos casos la **función de costos** a utilizar puede ser la llamada **Cross-Entropy Loss**:

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\mathbf{x}, \mathbf{y} \sim \hat{p}_{\text{data}}} (\log p_{\text{modelo}}(\mathbf{y}|\mathbf{x})) \quad (8.3)$$

Como se puede observar, la elección de la **unidad de output** definirá la forma que toma la función de costos, pero no será necesario definir una función específica para cada problema. En general se resolverá el problema por máxima verosimilitud. La elección en la unidad de output dependerá del tipo de problema que se quiera resolver, por ejemplo

1. Unidad de output lineal $\hat{y} = h^{(l)}U + c$

Útil para cuando se busca retornar la media de una distribución Gaussiana condicional $p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|\hat{\mathbf{y}}; \mathbf{I})$, el problema de maximizar la log-verosimilitud será equivalente a minimizar el error cuadrático medio por lo que resulta adecuado para un problema de regresión.

2. Unidad de output sigmoidal $\hat{y} = \text{sig}(h^{(l)}U + c)$

Este se utiliza para problemas de clasificación binaria, para resolver el problema de máxima verosimilitud, se modela utilizando una distribución Bernoulli en y condicional en x . La unidad de output modela entonces $P(y=1|\mathbf{x})$ o la probabilidad de pertenecer a la clase 1.

3. Unidad de output softmax $\hat{y} = \text{softmax}(h^{(l)}U + c)$

Es una generalización de la función sigmoidal definida como

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

y se utiliza para el caso de clasificación multiclas.

8.2. Entrenamiento de una red neuronal

8.2.1. Forward Propagation

Al usar una red neuronal *feedforward*, la información fluye a través de la red desde el ingreso de un input x hasta producir un output \hat{y} . Esto se conoce como **forward propagation** y es la que durante el entrenamiento calcula el costo $J(X, \theta)$. El algoritmo que la describe es el siguiente:

Algoritmo 10 Forward Propagation

Requerir: Profundidad de la red, l

Requerir: $\mathbf{W}^{(k)}, k \in \{1, \dots, l\}$, pesos de la red

Requerir: $\mathbf{b}^{(k)}, k \in \{1, \dots, l\}$, parámetros bias de la red

Requerir: \mathbf{x} , el input y \mathbf{y} el output

Requerir: U, c, g , matriz de peso, bias y función de output de la última capa respectivamente

```
1:  $\mathbf{h}^{(0)} \leftarrow \mathbf{x}$ 
2: for  $k = 1, \dots, l$ : do
3:    $\mathbf{u}^{(k)} \leftarrow \mathbf{h}^{(k-1)} \mathbf{W}^{(k)} + \mathbf{b}^{(k)}$ 
4:    $\mathbf{h}^{(k)} \leftarrow f^{(k)}(\mathbf{u}^{(k)})$ 
5:  $\hat{\mathbf{y}} \leftarrow g(\mathbf{h}^{(l)} U + c)$ 
6:  $J \leftarrow L(\hat{\mathbf{y}}, \mathbf{y})$ 
```

8.2.2. Backward Propagation - Preliminares

El algoritmo de **backpropagation** permite que la información del costo fluya en sentido inverso a través de la red para calcular el gradiente de la función de pérdida.

Aunque es posible obtener una expresión analítica para el gradiente, evaluar la expresión puede ser muy caro computacionalmente. Es por esto que las librerías que tienen implementadas redes neuronales utilizan diferenciación automática para poder calcularlo, logrando hacerlo de forma mucho más eficiente. Muchos autores indican que este último hecho es uno de los que permitió todo el avance que ha habido en este campo durante los últimos años.

Luego de obtener el gradiente, otro algoritmo como descenso de gradiente estocástico realizará el aprendizaje usando la expresión que fue calculada.

El algoritmo de backpropagation ha experimentado un resurgimiento en estas últimas décadas por su implementación en redes neuronales para tareas tales como reconocimiento de imágenes o procesamiento de lenguaje natural. Es considerado un algoritmo eficiente e implementaciones modernas de estas aprovechan el paralelismo de las GPU para mejorar su rendimiento.

Antes de comenzar a describir el algoritmo, debemos tener en cuenta que.

- Utilizaremos la notación para el paso intermedio antes de aplicar la función de activación como $u = hW + b$
- Es recurrente que en la etapa de entrenamiento de la red el conjunto de datos se entreguen en pequeños paquetes (**mini-batch**), es decir $(x^d)_{d=1}^N$ conjunto de N inputs, luego $u_{dj}^{(k)}$ corresponde al valor de u para el input d en el nodo j para la capa k . Todo esto con el fin de aprovechar lo más posible la optimización en operaciones de matrices que ofrecen las GPU, y de poder tener una convergencia más uniforme en el algoritmo de descenso de gradiente estocástico.
- Vamos a hacer la deducción del algoritmo de backpropagation para una función de error MSE en un problema de regresión.

$$J(X, \theta) = \frac{1}{2N} \sum_{d=1}^N (\hat{y}_d - y_d)^2$$

Nuestro objetivo será actualizar todos los valores $w_{ij}^{(k)}$ (peso del nodo i al nodo j en la capa k). Es decir, para utilizar el método del descenso de gradiente es necesario calcular $\frac{\partial J(X, \theta)}{\partial w_{ij}^{(k)}}$ notemos que

$$\frac{\partial J(X, \theta)}{\partial w_{ij}^{(k)}} = \frac{1}{N} \sum_{d=1}^N \frac{\partial}{\partial w_{ij}^{(k)}} \left(\frac{1}{2} (\hat{y}_d - y_d)^2 \right) = \frac{1}{N} \sum_{d=1}^N \frac{\partial J_d}{\partial w_{ij}^{(k)}}$$

Utilizando la regla de la cadena

$$\frac{\partial J_d}{\partial w_{ij}^{(k)}} = \frac{\partial J_d}{\partial u_{dj}^{(k)}} \frac{\partial u_{dj}^{(k)}}{\partial w_{ij}^{(k)}}$$

La expresión $\frac{\partial J_d}{\partial u_{dj}^{(k)}}$ corresponde a un término de **error** y lo denotaremos

$$\delta_{dj}^{(k)} \equiv \frac{\partial J_d}{\partial u_{dj}^{(k)}}$$

Mientras que para el otro término tenemos que

$$\frac{\partial u_{dj}^{(k)}}{\partial w_{ij}^{(k)}} = \frac{\partial}{\partial w_{ij}^{(k)}} \left(\sum_{a=1}^{k_k} w_{aj}^{(k)} h_{da}^{(k-1)} + b_j^{(k)} \right) = h_{di}^{(k-1)}$$

y así

$$\frac{\partial J_d}{\partial w_{ij}^{(k)}} = \delta_{dj}^{(k)} h_{di}^{(k-1)}$$

El gradiente total, será la suma de los N gradientes y que expresaremos en su forma matricial

$$\frac{\partial J}{\partial w_{ij}^{(k)}} = \sum_{d=1}^N \delta_{dj}^{(k)} h_{di}^{(k-1)} \Rightarrow \frac{\partial J}{\partial W^{(k)}} = (h^{(k-1)})^T @ \delta^{(k)} \quad (8.4)$$

Omitiremos la constante $1/N$ hasta el final del algoritmo.

8.2.3. Backward Propagation - Capas ocultas

Nuevamente, de la regla de la cadena

$$\delta_{dj}^{(k)} = \frac{\partial J_d}{\partial u_{dj}^{(k)}} = \sum_{a=1}^{k_{k+1}} \frac{\partial J_d}{\partial u_{da}^{(k+1)}} \frac{\partial u_{da}^{(k+1)}}{\partial u_{dj}^{(k)}} = \sum_{a=1}^{k_{k+1}} \delta_{da}^{(k+1)} \frac{\partial u_{da}^{(k+1)}}{\partial u_{dj}^{(k)}}$$

no es difícil ver que

$$\frac{\partial u_{da}^{(k+1)}}{\partial u_{dj}^{(k)}} = w_{ja}^{(k+1)} f'(u_{dj}^{(k)}) \Rightarrow \delta_{dj}^{(k)} = f'(u_{dj}^{(k)}) \sum_{a=1}^{k_{k+1}} w_{ja}^{(k+1)} \delta_{da}^{(k+1)}$$

Hemos encontrado una expresión para calcular el gradiente en una capa k en base al gradiente de la siguiente capa $k+1$ (De aquí el nombre **backward propagation**).

Lo anterior en su forma matricial queda descrito por

$$\delta^{(k)} = f'(u^{(k)}) * \left(\delta^{(k+1)} @ (W^{(k+1)})^T \right) \quad (8.5)$$

Lo único que queda para presentar el algoritmo final es calcular los gradientes en la última capa (output)

8.2.4. Backward Propagation - Capa de output

Estamos suponiendo un problema de regresión por lo que el output será de una sola salida y la función de error es MSE, entonces

$$\delta_{d1}^{(l)} = \frac{\partial J_d}{\partial u_{d1}^{(l)}} = (\hat{y}_d - y_d)(\hat{y}_d)'$$

Además, la función de activación en el output será lineal y por tanto $(\hat{y}_d)' = 1$, finalmente el término de normalización N se agrega en este paso. La forma matricial queda en

$$\delta^{(l)} = \frac{1}{N}(\hat{y} - y) \quad (8.6)$$

Además de lo anterior, se puede probar también que

$$\frac{\partial J}{\partial b^{(k)}} = \text{Sum}_1 \left(\delta^{(k)} \right), \quad \frac{\partial J}{\partial U} = (h^{(l)})^T @ \delta^{(l)}, \quad \frac{\partial J}{\partial c} = \text{Sum}_1 \left(\delta^{(l)} \right) \quad (8.7)$$

Donde Sum_1 es la suma sobre la primera dimensión de la matriz, es decir $\text{Sum}_1(A) = (\sum_i A_{ij})_j$. Con todo lo anterior ya estamos en condiciones para mostrar el algoritmo de backpropagation.

Algoritmo 11 Backward Propagation

Requerir: Learning rate λ .

- 1: Forward propagation, guardar (\hat{y}) , $(u^{(k)})$ y $(h^{(k)})$.
- 2: Evaluar el error de la última capa $\delta_1^{(l)}$ según la unidad de output
- 3: Actualizar $U \leftarrow U - \lambda \frac{\partial J}{\partial U}$
- 4: Actualizar $c \leftarrow c - \lambda \frac{\partial J}{\partial c}$
- 5: **for** $k = l, \dots, 1$: **do**
- 6: Calcular $\delta^{(k)}$ con las ecuaciones en las capas ocultas
- 7: Evaluar las derivadas parciales $\frac{\partial J}{\partial W^{(k)}}$ y $\frac{\partial J}{\partial b^{(k)}}$
- 8: Actualizar $W^{(k)} \leftarrow W^{(k)} - \lambda \frac{\partial J}{\partial W^{(k)}}$
- 9: Actualizar $b^{(k)} \leftarrow b^{(k)} - \lambda \frac{\partial J}{\partial b^{(k)}}$,

Observación 8.1. El algoritmo anterior debe ser aplicado para todos los batch del conjunto de entrenamiento, la cantidad de veces que se pase por todos los datos de entrenamiento, se conoce como **épocas**, en general un mayor número de épocas disminuye el error de entrenamiento, sin embargo, muchas épocas podrían reducir la capacidad de generalización debido al **overfitting**.

8.3. Regularización

Las redes neuronales y algoritmos de *deep learning* son aplicados a tareas extremadamente complejas como lo son el procesamiento de imágenes, audio, y texto. Controlar la complejidad de un modelo no solo se reduce a encontrar el tamaño y cantidad de parámetros adecuados, como se ha visto para otros modelos de aprendizaje de máquinas, sino que en la práctica el modelo con el mejor ajuste por lo general será un modelo grande (profundo) que ha sido regularizado apropiadamente.

El objetivo de las técnicas de regularización es el de reducir el *error de generalización*, es decir, el error esperado al clasificar datos nunca antes vistos pero manteniendo la capacidad del modelo (profundidad, cantidad de nodos, funciones de activación, etc...)

8.3.1. Regularización L^2

Una regularización que se basa en limitar la norma de los parámetros del modelo es la ya conocida **regularización L^2** (o **ridge regression**), mediante la cual se obtiene la función objetivo regularizada \tilde{J} :

$$\tilde{J}(\boldsymbol{\theta}; \mathbf{X}, \mathbf{y}) = J(\boldsymbol{\theta}; \mathbf{X}, \mathbf{y}) + \frac{\alpha}{2} \|\boldsymbol{\theta}\|_2^2 \quad (8.8)$$

en donde el hiperparámetro $\alpha \in [0, \infty]$ indica qué tanta importancia se le da al término de regularización sobre el objetivo, no habrá regularización cuando $\alpha = 0$ y se observará un mayor efecto regularizador a medida que α crece. Cabe destacar que típicamente en una regularización por la norma solo se regularizan los *pesos*, dejando los términos de *bias* sin regularizar. Esto ya que cada término de *bias* controla el comportamiento de solo una variable implicando que no se introduce mucha varianza al dejarlos sin regularizar, por otro lado, regularizar los *bias* puede inducir un alto nivel de *underfitting*.

No es difícil ver que esta regularización es equivalente a la actualización de parámetros según gradiente estocástico de la forma

$$W^{(k)} \leftarrow (1 - \beta)W^{(k)} - \lambda \frac{\partial J}{\partial W^{(k)}}, \quad \beta = \lambda\alpha$$

y por eso también es llamada *weight decay* y es la forma en que se implementa en la práctica.

8.3.2. Dropout

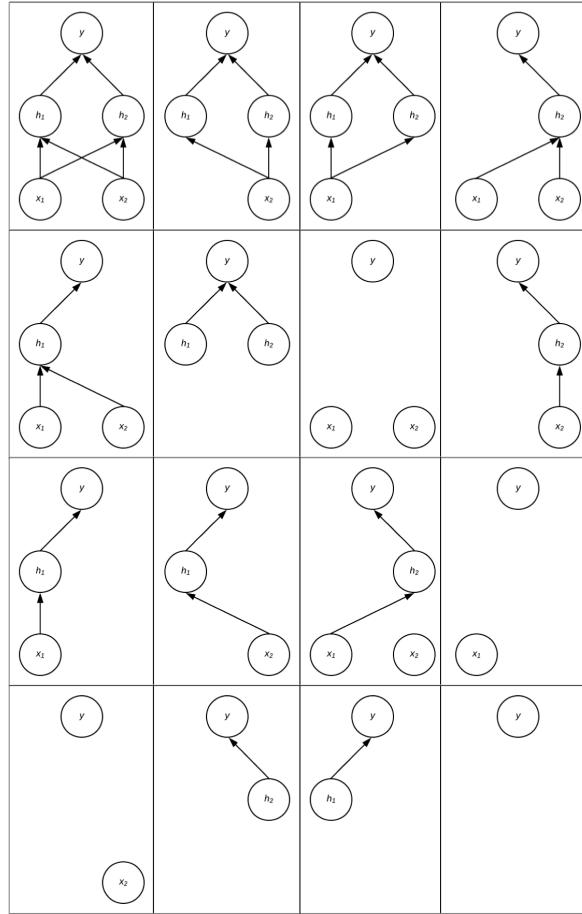
El método de **Bagging** consiste en entrenar múltiples modelos y evaluarlos en cada dato del set de testeо. Esto no es práctico para redes neuronales, ya que un solo modelo puede ser muy caro de entrenar y evaluar.

Sin embargo, el método de **Dropout** provee una aproximación barata computacionalmente para entrenar y evaluar *bagged* ensambles compuestos por una cantidad exponencial de redes neuronales.

Esta técnica de regularización consiste en entrenar en cada iteración una fracción de los pesos mediante una elección aleatoria. Su implementación es bastante simple y basta con definir para cada capa k la probabilidad $1 - p_k$ de 'apagar' una neurona en particular. Esto corresponde a la aplicación de una máscara binaria $M^{(k)}$ de la siguiente forma:

$$h^{(k)} = f^{(k)}(h^{(k-1)}W^{(k)} + b^{(k)}) * \frac{M^{(k)}}{p_k}, \quad M_i^{(k)} \sim \text{Bernoulli}(p_k) \quad \forall i, k \quad (8.9)$$

Fig. 49. Dropout entrena potencialmente todas las subredes que se puedan formar a partir de la red neuronal original (primer recuadro) al apagar el output que producen las distintas unidades



Notar que al aplicar el algoritmo de backpropagation, también es necesario 'apagar' las neuronas que no participaron del forward para evitar que sean entrenadas.

8.3.3. Otros métodos de regularización

Otras formas de regularización también buscan introducir alguna fuente de ruido (como en dropout) para que la red neuronal aprenda principalmente los parámetros más importantes, así logrando un bajo error de generalización. Una de estas técnicas es **dataset augmentation**, que consiste en generar nuevos datos de entrenamiento (inyectando ruido en el set de entrenamiento), creando datos \mathbf{x} falsos para los cuales se pueda tener una etiqueta y (por ejemplo, una imagen invertida de un gato sigue siendo un gato), o **entrenamiento adversarial**, en donde se perturban ejemplos para fortalecer a la red (por ejemplo, cambiar pixeles de una imagen que generen cambios imperceptibles para un humano pero que pueden afectar fuertemente la capacidad de predicción de un modelo). Otra técnica es **noise injection** en los pesos (Jim et al., 1996; Graves, 2011), lo cual se puede interpretar como una implementación estocástica de inferencia Bayesiana sobre los pesos, debido a que el aprendizaje consideraría que los pesos son inciertos y, por lo tanto, representables mediante una distribución de probabilidad.

También, por supuesto, **early stopping** es una técnica válida para regularizar redes neuronales al detener el entrenamiento cuando el error de generalización en el conjunto de validación comience a aumentar.

8.4. Algoritmos de optimización

Como ya se ha comentado, la optimización en redes neuronales busca resolver un problema particular: encontrar los parámetros θ que disminuyan significativamente $J(\theta)$, que depende de alguna medida de desempeño evaluada en la totalidad del set de entrenamiento, luego se evalúa el error en el set de validación para tener una idea del desempeño, finalmente se ven los resultados en el set de testeo.

Esto se reduce a minimizar la esperanza del error sobre la distribución generadora de los datos, p_{data} :

$$J^*(\theta) = \mathbb{E}_{(\mathbf{x}, y) \sim p_{\text{data}}} L(f(\mathbf{x}; \theta), y) \quad (8.10)$$

Reemplazamos la expresión anterior por un problema sustituto, que consiste en escribir la función de costos como un promedio sobre el set de entrenamiento, como se puede observar la diferencia entre las dos expresiones radica en el hecho que se considera que los datos fueron generados por distribuciones de probabilidad distintas (p_{data} en la primera expresión, \hat{p}_{data} en la segunda):

$$J(\theta) = \mathbb{E}_{(\mathbf{x}, y) \sim \hat{p}_{\text{data}}} L(f(\mathbf{x}; \theta), y) = \frac{1}{m} \sum_{i=1}^m L(f(x^{(i)}; \theta), y^{(i)}) \quad (8.11)$$

8.4.1. Minibatch

Los algoritmos de optimización para aprendizaje de máquinas típicamente actualizan los parámetros usando un valor esperado del costo, obtenido a través de un subset de los términos de la función de costos. La propiedad más usada respecto de la función objetivo es que

$$\nabla_{\theta} J(\theta) = -\mathbb{E}_{\mathbf{x}, y \sim \hat{p}_{\text{data}}} \nabla_{\theta} \log p_{\text{modelo}}(y | \mathbf{x})$$

Los algoritmos de optimización que usan el set de entrenamiento completo para actualizar los parámetros en cada iteración se conocen como **métodos de batch** o **determinísticos** y tienden a quedar atrapados en óptimos locales.

Los algoritmos que usan un solo ejemplo a la vez se conocen como **métodos estocásticos** y son tremadamente inefficientes para una cantidad grande de datos.

Por lo tanto, la mayoría de los algoritmos usados en redes neuronales pertenecen a una categoría intermedia, estos son los **métodos de minibatch** o **minibatch estocástico**, los cuales usan un subconjunto de tamaño reducido y calculan un promedio de gradientes para obtener el valor esperado.

8.4.2. Algoritmos con momentum

Aunque los métodos de descenso de gradiente estocástico sigue siendo un algoritmo popular, el aprendizaje a veces puede ser lento. Los algoritmos que incorporan momentum fueron diseñados para acelerar el aprendizaje, especialmente en presencia de altas curvaturas, cuando se tienen gradientes pequeños pero consistentes o gradientes ruidosos. El algoritmo **descenso de gradiente estocástico con momentum** acumula un decaimiento exponencial de media móvil de los gradientes pasados y continúa su movimiento en esta dirección. Un hiperparámetro β determina qué tan rápido las contribuciones de gradientes pasados decaen exponencialmente. Su implementación es como sigue

$$v_t = (1 - \beta) \left(\frac{\partial J}{\partial \theta_t} \right) + \beta v_{t-1}; \quad v_0 = 0$$

La actualización de parámetros viene dada por

$$\theta_{t+1} = \theta_t - \lambda v_t$$

8.4.3. Algoritmos con learning rate adaptativos

En la práctica el learning rate resulta ser uno de los hiperparámetros más difíciles de ajustar debido a su importante efecto en el desempeño del modelo. La función de costos suele ser altamente sensible (a crecer o decrecer) en algunas direcciones en el espacio de los parámetros e insensible en otras, por lo que hace sentido usar un learning rate distinto para cada parámetro y automáticamente adaptar este parámetro durante el aprendizaje. El algoritmo **AdaGrad** adapta el learning rate de todos los parámetros al escalarlos de manera inversamente proporcional a la raíz cuadrada de la suma de todos los cuadrados históricos del gradiente. Los parámetros con derivadas parciales más grandes tienen un rápido decrecimiento en su learning rate, mientras que los parámetros con derivadas parciales pequeñas decrecen en menor cantidad su learning rate. El efecto neto es mayor progreso en zonas más planas del espacio de los parámetros. Se implementa según:

$$v_t = v_{t-1} + \frac{\partial J}{\partial \theta_t} * \frac{\partial J}{\partial \theta_t}; \quad v_0 = 0$$

Con actualizaciones dadas por

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} * \frac{\partial J}{\partial \theta_t}$$

Donde ϵ es un parámetro para estabilidad numérica.

Otras generalizaciones populares son el algoritmo **RMSProp**, que modifica el algoritmo AdaGrad para tener un mejor desempeño en funciones no convexas al cambiar la acumulación del gradiente por una media móvil que decae exponencialmente.

$$v_t = \beta v_{t-1} + (1 - \beta) \frac{\partial J}{\partial \theta_t} * \frac{\partial J}{\partial \theta_t}; \quad v_0 = 0$$

y actualiza

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} * \frac{\partial J}{\partial \theta_t}$$

Finalmente el algoritmo **Adam**, es aquel que combina RMSProp con momentum (con algunas distinciones importantes). Hasta el momento no hay un algoritmo que tenga un desempeño superior al de los demás en distintos escenarios (Schaul et al., 2014), por lo que se recomienda usar el algoritmo de optimización con el que el usuario se sienta más cómodo al momento de ajustar los hiperparámetros.

8.5. Obstáculos en el entrenamiento de redes neuronales

Como ya se mencionó, el entrenamiento de redes neuronales suele ser un problema no convexo. Esto sumado al hecho de que estos modelos suelen estar *sobre-parametrizados*, genera obstáculos muchas veces difíciles de sobrepassar en el entrenamiento de redes neuronales. En esta sección exploraremos algunos de ellos y sus posibles soluciones.

8.5.1. Valores Iniciales

La inicialización de los pesos de la red neuronales es muy importante para el entrenamiento, ya que comenzar desde el punto incorrecto puede significar que el entrenamiento no logre llegar a los mínimos deseados de la función de pérdida.

Como se puede observar en las distintas funciones de activación, tener pesos cercanos a 0 genera que este modelo actuar de forma bastante similar a un modelo lineal en el límite. Lo que se suele hacer es partir con pesos generados aleatoriamente cercanos a 0, y así con el entrenamiento converger de un modelo más parecido a uno lineal, a otro no lineal.

Es importante notar que inicializar los pesos en 0 genera que las derivadas parciales de la función de pérdida sean 0, y por lo tanto estos no se moverán durante el entrenamiento. Por otra parte, inicializar los pesos muy lejos del 0 suele llevar a malos resultados.

8.5.2. Overfitting

Dada la gran cantidad de parámetros que suelen tener las redes neuronales, es muy fácil hacer overfitting en el conjunto de entrenamiento. Tanto *early stopping* como *weight-decay* son buenos métodos para sobrepassar este obstáculo.

8.5.3. Cantidad de neuronas en las capas escondidas

Determinar la cantidad de neuronas en las capas escondidas define qué tipo de funciones podremos ajustar con el modelo. Sin embargo, no es trivial determinar esta cantidad.

En general, es mejor tener muchas neuronas a tener pocas. Con pocas neuronas, es posible que no se logre captar completamente la no-linealidad de un modelo. Por otra parte, con muchas neuronas, se esperaría que los pesos que sean innecesarios converjan a 0 en el entrenamiento.

8.6. Deep Learning y otros tipos de redes neuronales

El término **deep learning** se asocia a resolver problemas más intuitivos (y fáciles) para los humanos que hasta hace solo algunos años eran extremadamente difíciles para una máquina, como lo son el reconocimiento de objetos en imágenes (visión de computadores), la traducción de texto desde un lenguaje a otro (machine translation), reconocimiento de voz, entre otros, mediante redes neuronales de dos o más capas.

Los problemas más difíciles para los humanos ya se han estado resolviendo hace mucho tiempo antes del deep learning, como divisar una estrategia ganadora en ajedrez (Por ejemplo en ajedrez), aunque las redes neuronales profundas han seguido progresando en resolver este tipo de problemas (el ejemplo más famoso es el de Deep Mind).

Las arquitecturas principales que han permitido resolver estos problemas en los últimos años (sumado a los avances en poder de computación y cantidad de datos que existen hoy) se presentan en esta sección: las **redes convolucionales** para procesamiento de imágenes, y las **redes recurrentes** para modelar series de tiempo (e.g., texto, audio). También se presentan otras arquitecturas que son tema activo de investigación en deep learning: los **autoencoders** y las **redes generativas adversariales**.

8.6.1. Redes neuronales convolucionales

Las **redes neuronales convolucionales** (o CNNs por su sigla en inglés) son un tipo de redes neuronales que fueron diseñadas para procesar datos con una tipología tipo-*grid* (grilla). Una serie de tiempo que tiene observaciones en intervalos regulares de tiempo se puede pensar como un *grid* de 1 dimensión. Las imágenes se pueden pensar como *grids* de 2-D de píxeles. El nombre de esta arquitectura hace referencia a que usan una operación matemática conocida como convolución.

La operación de **convolución** se define como:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da \quad (8.12)$$

En el contexto de CNNs, el primer argumento a convolucionar, x , es el **input**, y el segundo argumento, w , se conoce como el **kernel**. El output, $s(t)$ se conoce como **feature map**. En aplicaciones de aprendizaje de máquinas, el input serán un arreglo multidimensional de datos, y el kernel un arreglo multidimensional

de parámetros que se buscarán aprender. Usualmente, al trabajar con datos en un computador el tiempo se considerará discreto, por lo que resulta conveniente definir la operación de convolución discreta:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \quad (8.13)$$

Se asumirá que las funciones son 0 en todo su dominio excepto en el set finito de puntos para el cual se guardan valores, permitiendo realizar estas sumatorias infinitas. Las librerías de redes neuronales implementan la función **cross-correlation** y la llaman convolución. Para una imagen I de 2 dimensiones y un kernel K de 2 dimensiones, esto es:

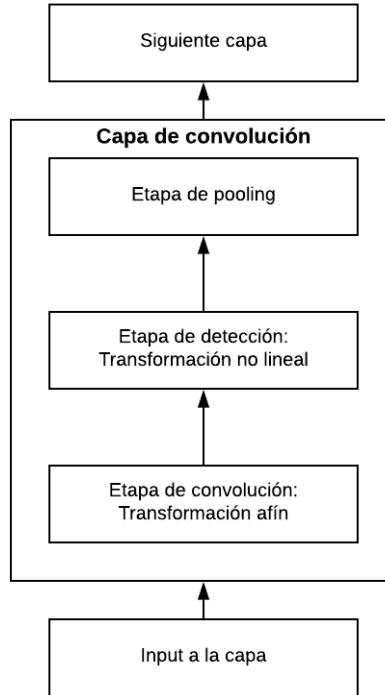
$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n) \quad (8.14)$$

Esto es equivalente a la operación de convolución discreta con la diferencia que el operador no es commutativo, $S(i, j) = (I * K)(i, j) \neq (K * I)(i, j)$, lo cual no es importante para implementaciones de redes neuronales.

La motivación por usar convoluciones surge de 3 importantes propiedades: interacciones *sparse*, *parameter sharing*, y representaciones equivariantes. **Interacciones sparse** se refiere a que hay menos conexiones que en una red *fully connected*, esto mediante el uso de kernels de menor dimensionalidad que el input, lo cual implica una eficiencia en términos de memoria por tener que almacenar menos parámetros, y eficiencia computacional por realizar menos operaciones. **Parameter sharing** se refiere a usar los mismo parámetros para distintas funciones dentro del modelo. Esto implica usar los pesos aprendidos en múltiples partes del input (como para reconocer bordes, por ejemplo). Que una función sea **equivariante** significa que si el input cambia, el output cambia de la misma forma. Esto permite que en imágenes la convolución cree un map 2-D dónde ciertos atributos aparecen en el input.

Una capa típica de una red convolucional consta de 3 etapas: primero, aplicar varias convoluciones para producir un set de activaciones lineales. Luego, aplicar una activación no lineal (**detector stage**). Finalmente, una función de *pooling* para modificar aún más el output de la capa (ver figura). Una función de **pooling** reemplaza el output de la red en alguna locación por estadísticos de los outputs cercanos.

Fig. 50. Capa de una red convolucional



Max pooling retorna el valor máximo de un output en una vecindad rectangular. Las operaciones de pooling permiten que la red sea invariantes a pequeñas transformaciones en el input. Pooling también es esencial para procesar inputs de tamaño variable (por ejemplo imágenes de distinto tamaño).

Otras diferencias con respecto a la operación de convolución en el contexto de redes neuronales son, por ejemplo, el aplicar múltiples convoluciones en paralelo, esto permite extraer distintos tipos de atributos en vez de 1 solo. Por otro lado, el **stride** hace referencia a cada cuántos pixeles se quieren convolucionar en cada dirección en el output. En la figura se muestra el ejemplo de una convolución con stride. Esta operación permite reducir nuevamente el costo computacional. Esto también implica que el output disminuye su tamaño en cada capa. El uso de **padding** puede revertir esto. **Padding** se refiere a agrandar el input con ceros para hacerlo más amplio. Una convolución en la que no se usa **zero-padding** se conoce como **valid**. Una convolución que mantiene el tamaño desde el input al output se conoce como **same** (ver figura). En la práctica, las capas de una red convolucional usan operaciones entre una convolución valid y same.

Fig. 51. Convolución con un stride igual a 2

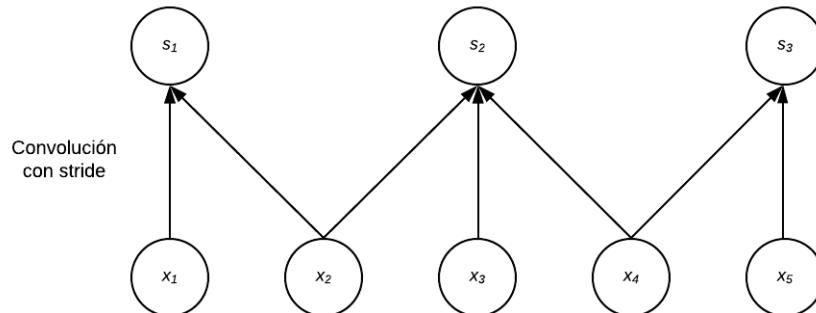
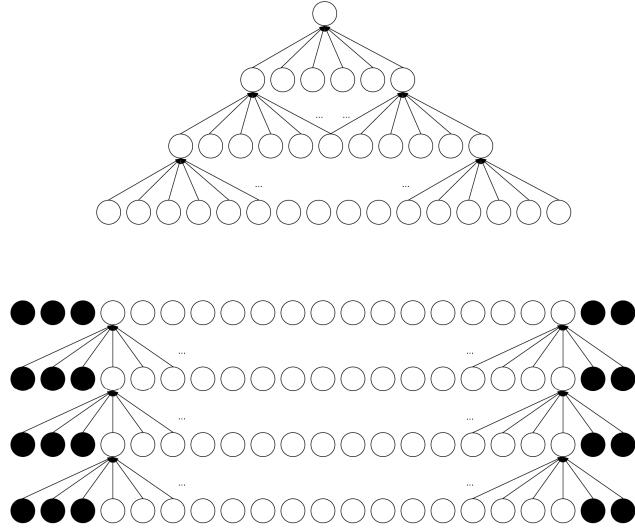


Fig. 52. Efecto de no usar zero-padding en una red convolucional (Arriba) y efecto de usar zero padding en una red convolucional (Abajo) en cuanto al tamaño de la red



8.6.2. Redes neuronales recurrentes

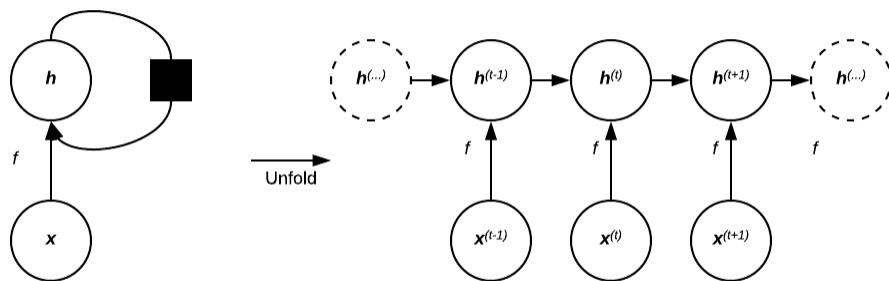
Las **redes neuronales recurrentes** o **RNNs** son una familia modelos de redes neuronales especializados para procesar datos secuenciales, $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}$. Las RNNs también comparten parámetros, pero en una forma muy distinta que las CNNs. En una RNN, cada miembro del output en una etapa es una función de cada miembro del output de la etapa anterior.

Se denota por $\mathbf{h}^{(t)}$ al estado de un sistema dinámico que involucra una recurrencia conducido por un input externo $\mathbf{x}^{(t)}$:

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}; \mathbf{x}^{(t)}, \boldsymbol{\theta}) \quad (8.15)$$

La figura siguiente muestra una red recurrente que procesa un input \mathbf{x} incorporándolo al estado \mathbf{h} que es traspasado a través del tiempo.

Fig. 53. Ejemplo de una red recurrente sin output



Las redes recurrentes se pueden construir de muchas formas distintas. Al igual que una red neuronal puede representar casi cualquier función, una red recurrente modela cualquier función que involucre una

recurrencia. Se puede representar el estado de una red recurrente luego de t pasos mediante una función $g^{(t)}$:

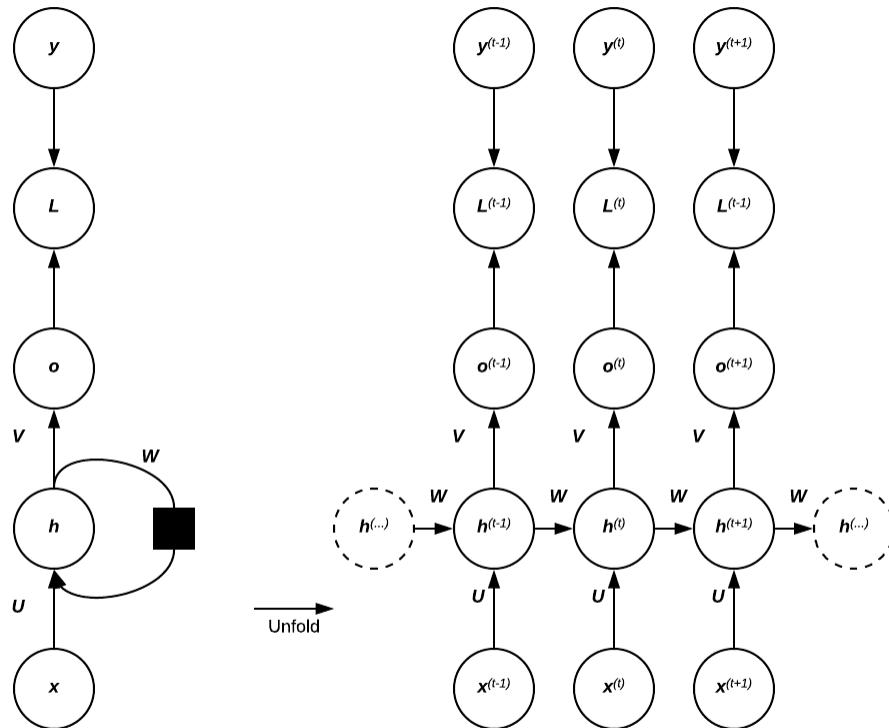
$$\mathbf{h}^{(t)} = g^{(t)}(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}, \dots, \mathbf{x}^{(1)}) = f(\mathbf{h}^{(t-1)}; \mathbf{x}^{(t)}, \boldsymbol{\theta}) \quad (8.16)$$

Existen varios tipos de RNNs que se han diseñado para distintos fines. Algunos ejemplos de estas son:

- Redes recurrentes que producen un output en cada instante de tiempo y tienen conexiones entre todas las unidades escondidas
- Redes recurrentes que producen un output en cada instante de tiempo y tienen conexiones entre el output a la unidad escondida del siguiente instante
- Redes recurrentes con conexiones entre las unidad escondidas, que procesan una secuencia entera antes de producir el output

La figura muestra un ejemplo de la arquitectura para el primer caso.

Fig. 54. Ejemplo de una red recurrente que produce un output en cada instante de tiempo, y que comparte el estado a través del tiempo



Esta red presentada puede ser usada para producir palabras en cada instante, y así producir oraciones que hagan sentido en una conversación. Como ejemplo de entrenamiento de una red con esta arquitectura, en donde en la última capa se decide mediante una función softmax la palabra más probable que deba seguir a la palabra anterior, se tienen las ecuaciones de *forward propagation* para cada instante de tiempo:

$$\begin{aligned}
\mathbf{a}^{(t)} &= \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t-1)} \\
\mathbf{h}^{(t)} &= \tanh(\mathbf{a}^{(t)}) \\
\mathbf{o}^{(t)} &= \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)} \\
\hat{\mathbf{y}}^{(t)} &= \text{softmax}(\mathbf{o}^{(t)})
\end{aligned} \tag{8.17}$$

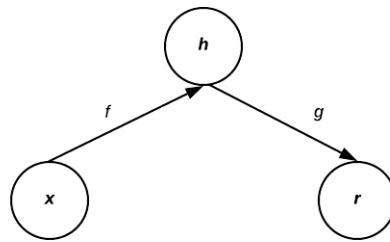
El algoritmo aplicado para obtener el gradiente en este tipo de arquitectura se conoce como **back-propagation through time**, y consiste en aplicar el algoritmo de *back-propagation* generalizado para el grafo computacional *unfolded* de la red, como los mostrados en las figuras de redes recurrentes.

Las redes recurrentes sufren de no poder recordar largas dependencias a través del tiempo, debido a que las recurrencias implican multiplicar una matriz de pesos múltiples veces a través de la red, provocando superficies planas o muy empinadas que resultan en que los algoritmos de aprendizaje por gradiente tengan problemas de **vanishing gradients** o **exploding gradients**, respectivamente. Arquitecturas que han logrado superar esto son las que incluyen compuertas (funciones sigmoidales) que deciden automáticamente qué olvidar y qué seguir propagando a través de la red. Estos son los modelos de **gated recurrent units (GRUs)** y **long-short term memory network (LSTM)**.

8.6.3. Autoencoders

Un **autoencoder** es una red neuronal que busca replicar el input hacia el output, osea, se busca que la información que entra a la red sea lo más parecida posible a la de salida, para lo cual cuenta con una capa interna $\mathbf{h} = f(\mathbf{x})$ que codifica el input (genera una representación de este) llamada **encoder** y una función que produce la reconstrucción $\mathbf{r} = g(\mathbf{h})$, el **decoder**. Un *autoencoder* buscará aprender los *encoder* y *decoder* tales que $g(f(\mathbf{x})) = \mathbf{x}$ para todo \mathbf{x} . Como el modelo está forzado a aprender los atributos más importantes para que pueda efectivamente reproducir el input en su output, este aprenderá en general propiedades útiles de los datos de entrenamiento. Los *autoencoders* modernos modelan mappings estocásticos $p_{\text{encoder}}(\mathbf{h}|\mathbf{x})$ y $p_{\text{decoder}}(\mathbf{x}|\mathbf{h})$, en vez de funciones determinísticas. En la figura se muestra la arquitectura de un *autoencoder*.

Fig. 55. Estructura de un *autoencoder* típico



Una manera de obtener atributos útiles de \mathbf{x} es forzando a que el *encoder* tenga una dimensionalidad menor que el input. Este tipo de *autoencoders* se denominan **undercomplete**. El aprendizaje se describe mediante la optimización de la función de pérdida, $L(\mathbf{x}, g(f(\mathbf{x})))$. Cuando el *decoder* es lineal y la función de pérdida es el error cuadrático medio, un *undercomplete autoencoder* aprende a generar el mismo subespacio que el algoritmo **principal component analysis (PCA)**, es decir, el *autoencoder* que fue entrenado para reproducir los datos de entrenamiento mediante una reducción de dimensionalidad y una reconstrucción aprendió como efecto colateral el subespacio principal. Es así como entonces, *autoencoders* con *encoders* y *decoders* no lineales pueden aprender representaciones no lineales más poderosas que PCA.

Un *autoencoder* con dimensión de su *encoder* igual a la del input se conoce como **overcomplete**. Estos *autoencoders*, al igual que los *undercomplete*, pueden fallar en aprender una representación útil del

input si tienen mucha capacidad, por lo que será importante también regularizar estas redes neuronales.

Otras aplicaciones de los autoencoders, aparte de aprender una reducción de dimensionalidad, es aprender representaciones útiles que sirvan para un posterior modelo de redes neuronales (o, más general, de aprendizaje de máquinas). Por ejemplo, en vez de usar **one-hot-vectors** para representar palabras (en donde se tiene un vector del largo de cierto vocabulario compuesto por ceros excepto para la palabra que se quiere representar, indicando un valor de 1 en esa posición), se pueden usar **embeddings**, que son representaciones del input a un espacio de valores reales. A diferencia de los *one-hot-vectors*, en un *embedding* la distancia entre las representaciones del texto sí tiene un significado, y este tipo de representaciones podría entregar mejores resultados en la tarea en que se esté usando.

8.6.4. Redes generativas adversariales

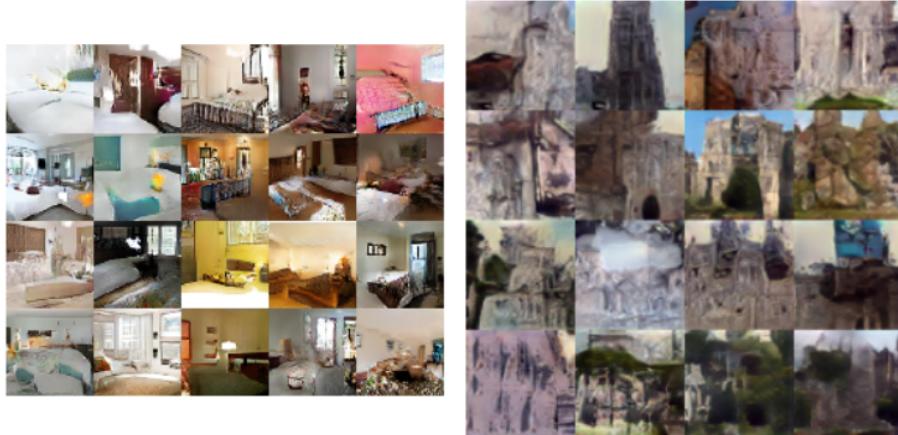
Una **red generativa adversarial** (o **GAN**) se basa en un escenario de teoría de juegos, en donde una **red generadora** debe competir con un adversario. La red generadora produce muestras $\mathbf{x} = g(\mathbf{z}; \boldsymbol{\theta}^{(g)})$, mientras que una **red discriminadora** trata de distinguir entre muestras obtenidas de los datos de entrenamiento y muestras generadas por la red generadora. El discriminador retorna una probabilidad, $d(\mathbf{x}; \boldsymbol{\theta}^{(d)})$, indicando la probabilidad de que \mathbf{x} sea un dato real y no uno simulado.

Para formular el aprendizaje, se describe un juego de suma cero en donde una función $v(\boldsymbol{\theta}^{(g)}, \boldsymbol{\theta}^{(d)})$ determina el pago del discriminador, y el generador recibe $-v(\boldsymbol{\theta}^{(g)}, \boldsymbol{\theta}^{(d)})$ como pago. Así, durante el entrenamiento cada jugador intenta maximizar su propio pago, para que en convergencia se tenga

$$g^* = \operatorname{argmin}_g \max_d v(g, d) \quad (8.18)$$

Esto motiva a que el discriminador aprenda a clasificar correctamente entre muestras reales y falsas y, simultáneamente, el generador intenta engañar al clasificador para que crea que las muestras generadas son reales. En convergencia, las muestras del generador son indistinguibles de los datos reales. Una motivación del uso de GANs es que cuando $\max_d v(g, d)$ es convexa en $\boldsymbol{\theta}^{(g)}$, el procedimiento asegura la convergencia.

Fig. 56. Imágenes generadas por una GAN entrenada con el set de datos LSUN. (Izquierda) Imágenes de dormitorios generadas por el modelo DCGAN (imagen de Radford et al., 2015). (Derecha) Imágenes de iglesias generadas por el modelo LAPGAN (imagen de Denton et al., 2015)



9. Procesos gaussianos

Como se vio en capítulos anteriores, el problema de regresión busca encontrar una función $y = f(x)$, dado un conjunto de pares de la forma $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$. Dentro de los métodos vistos para resolver el problema de regresión, se vio el de regresión lineal, lineal en los parámetros y no lineales. Una característica en común que tienen estos métodos es que el proceso de entrenamiento consiste en encontrar un número fijo de parámetros, que minimicen cierta función objetivo, donde la cantidad de parámetros y la forma del modelo es parte del diseño. A este tipo de modelos se les llama *modelos paramétricos*.

En contraste, se encuentran los modelos *no paramétricos* los cuales no tienen un número fijo de parámetros, donde pueden llegar a ser en algunos casos infinito. Un ejemplo es el algoritmo de k -vecinos más cercanos (KNN), donde los puntos del conjunto de entrenamiento son usados para clasificar nuevas muestras. Otro ejemplo son las máquinas de soporte vectorial (SVM) donde al entrenar se obtienen los vectores de soporte.

Es importante hacer la distinción entre parámetros que se aprenden y los parámetros del modelo, muchas veces llamados hiperparámetros, donde estos últimos pueden ser fijos independiente si el método es paramétrico o no paramétrico. Esto se ve en el caso de SVM donde los parámetros serían los vectores de soporte, y los hiperparámetros serían el tipo de kernel, los parámetros de dicho kernel y los demás valores elegidos de antemano que definen el tipo de modelo.

En este capítulo introduciremos un método no paramétrico probabilístico de regresión no lineal, llamado procesos gaussianos (\mathcal{GP}). Este modelo en vez de encontrar un candidato único de la función a estimar, define una distribución sobre funciones $\mathbb{P}(f)$, donde f es una función de un espacio de entrada \mathcal{X} a los reales, $f : \mathcal{X} \rightarrow \mathbb{R}$. Esto tiene la virtud de permitir cuantificar la incertidumbre puntual que existe en la predicción de nuestro modelo, la cual servirá en forma de intervalos de confianza para la distribución gaussiana.

Partiremos definiendo un \mathcal{GP} como una distribución a priori sobre funciones, y mostraremos que la densidad posterior se puede encontrar de forma exacta y que esta también es un \mathcal{GP} , conservando sus propiedades.

Es importante notar que si \mathcal{X} tiene cardinalidad infinita (por ejemplo $\mathcal{X} = \mathbb{R}$), f puede ser visto como un vector infinito dimensional. Como en la práctica no podemos trabajar con un vector infinito dimensional, dados n puntos $\{x_i\}_{i=1}^n \subset \mathcal{X}$, podemos definir el vector n -dimensional de valores de la función evaluada en dichos puntos $f(\mathbf{x}) = (f(x_1), \dots, f(x_n))^\top$.

Definición 9.1 (proceso gaussiano). Un proceso gaussiano (\mathcal{GP}) es una colección de variables aleatorias, tal que para cualquier subconjunto finito de puntos, estos tienen una distribución conjuntamente gaussiana.

Al aplicar esta definición a nuestro caso anterior, $\mathbb{P}(f)$ será un \mathcal{GP} y para cualquier conjunto finito $\{x_i\}_{i=1}^n \subset \mathcal{X}$, la distribución de $\mathbb{P}(f(\mathbf{x}))$ es Gaussiana multivariada $f(\mathbf{x}) = (f(x_1), \dots, f(x_n))^\top$. En este caso las variables aleatorias representan el valor de la función $f(x_i)$ en la posición x_i .

Un \mathcal{GP} queda completamente caracterizado por su función de media $m(\cdot)$ y función de covarianza $K(\cdot, \cdot)$, de esta forma para cualquier conjunto finito podemos encontrar la distribución. Definimos estas funciones como

$$m(x) = \mathbb{E}\{f(x)\} \tag{9.1}$$

$$K(x, x') = \mathbb{E}\{(f(x) - m(x))(f(x') - m(x'))\} \tag{9.2}$$

Y de esta forma podemos escribir el proceso como:

$$f \sim \mathcal{GP}(m(\cdot), K(\cdot, \cdot)) \quad (9.3)$$

Donde para un conjunto finito tenemos que la marginal resulta de la forma:

$$f(\mathbf{x}) \sim \mathcal{N}(m(\mathbf{x}), K(\mathbf{x}, \mathbf{x})) \quad (9.4)$$

Hasta el momento hemos hablado del espacio de entrada \mathcal{X} como genérico, un caso común es definir los \mathcal{GP} sobre el tiempo (\mathbb{R}^+), es decir que los x_i son instantes de tiempo. Es de notar que este no es el único caso, y se podría definir sobre un espacio más general, por ejemplo \mathbb{R}^d .

Otro punto a notar es que como estamos hablando de una colección (no necesariamente finita) de variables aleatorias, es necesario que se cumpla la propiedad de marginalización (o llamada consistencia¹²). Esta propiedad se refiere a que si un \mathcal{GP} define una distribución multivariada para digamos dos variables $(y_1, y_2) \sim \mathcal{N}(\mu, \Sigma)$ entonces también debe definir $y_1 \sim \mathcal{N}(\mu_1, \Sigma_{11})$ donde μ_1 es la componente respectiva del vector μ y Σ_{11} la submatriz correspondiente de Σ . En otras palabras, el tomar un subconjunto más grande de puntos no cambia la distribución de un subconjunto más pequeño. Y podemos notar que esta condición se cumple si tomamos la función de covarianza definida anteriormente.

9.1. Muestreo de un prior \mathcal{GP}

Como fue mencionado, un \mathcal{GP} define un *prior* sobre funciones, por lo que, antes de ver ningún dato se podría obtener una muestra de este proceso dado una función de media y covarianza. Un supuesto común es asumir la función de media $m(\cdot) = 0$ por lo que solo nos queda definir una función de covarianza o kernel, un ejemplo de kernel es el *Exponencial Cuadrático* (Square Exponential), también conocido como RBF (Radial Basis function) o Kernel Gaussiano (en general se evita este nombre porque este kernel no tiene relación con la distribución de los datos).

$$K_{SE}(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right) \quad (9.5)$$

Donde en este caso los parámetros son interpretables (y como veremos más adelante pueden ser aprendidos a través de un conjunto de entrenamiento) donde σ^2 es la varianza de la función, notar que esta es la diagonal de la matriz covarianza. El parámetro ℓ es conocido como el *lengthscale* que determina que tan lejos tiene influencia un punto sobre otro, donde en general un punto no tendrá influencia más allá de ℓ unidades alrededor.

Como sabemos, las funciones definidas por el \mathcal{GP} son vectores infinito dimensionales, por lo que no podemos muestrear de toda la función, pero tomando una cantidad suficiente de puntos podemos graficar muestras de un \mathcal{GP} dada una función de covarianza. Tomando un \mathcal{GP} con media cero ($m(\cdot) = 0$) y kernel SE, muestras del proceso para distintos valores de ℓ obtenemos la Fig.57, donde el área sombreada corresponde al intervalo de confianza del 95 %. Se puede ver que el parámetro ℓ controla que tan erráticas son las funciones, donde a medida que va aumentando las muestras se vuelven funciones más suaves.

¹²Para más detalles puede ver el teorema de consistencia de Kolmogorov

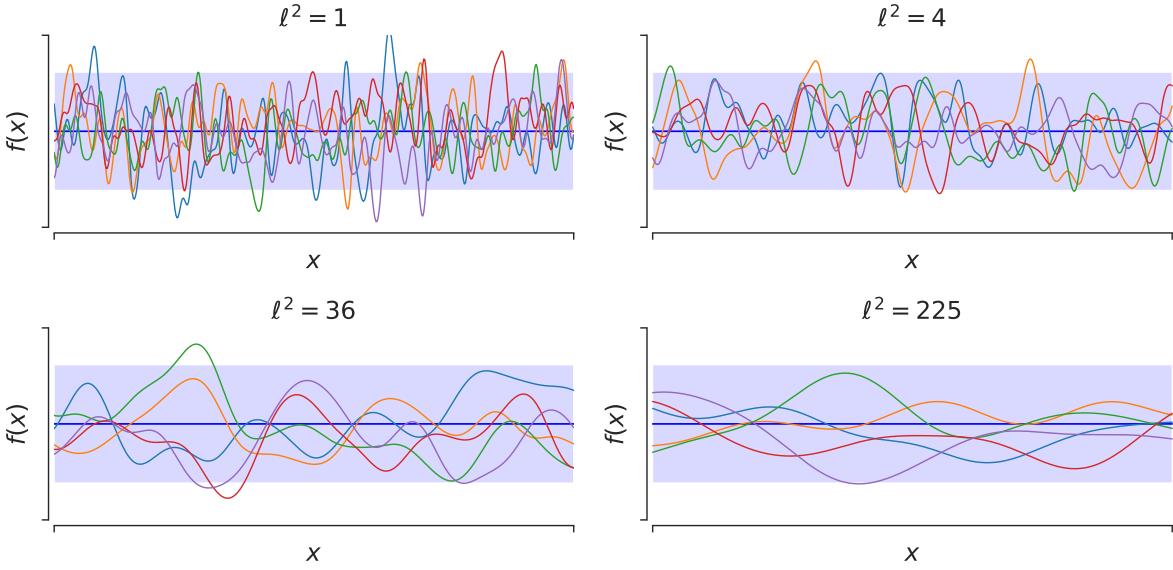


Fig. 57. Muestras de un prior \mathcal{GP} con kernel SE, para distintos *lengthscales* (ℓ) y función media $m(\cdot) = 0$, la parte sombreada corresponde al intervalo de confianza del 95 %. Se puede ver que a mayor ℓ las funciones se van volviendo más suaves.

9.2. Incorporando información

Ahora que ya podemos muestrear de nuestro prior, nos interesaría incorporar las observaciones que tenemos de la función a nuestro modelo. Para esto, se pueden considerar dos casos: observaciones con ruido y observaciones sin ruidos (o deterministas).

9.2.1. Evaluación sin ruido

En el caso de las observaciones sin ruido, tenemos observaciones de la forma $\{(x_i, f(x_i))\}_{i=1}^n$, donde tenemos el valor real de nuestra función en los puntos $[x_1, \dots, x_n] = X$. Luego, tenemos el par de entradas y observaciones $(X, f(X))$. Digamos que queremos realizar una predicción en el conjunto X_* de n_* puntos, luego la distribución conjunta es de la forma:

$$\begin{bmatrix} f(X) \\ f(X_*) \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} m(X) \\ m(X_*) \end{bmatrix}, \begin{bmatrix} K(X, X) & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix} \right) \quad (9.6)$$

Donde la submatriz $K(X, X_*)$ es de $n \times n_*$, $K(X, X)$ de $n \times n$ y así respectivamente para cada submatriz. Dada las observaciones y una función de covarianza, podemos evaluar la verosimilitud, que también es gaussiana, lo que nos lleva a un punto clave de los processos gaussianos:

Proposición 9.0.1. *Dado un prior \mathcal{GP} sobre $f(\cdot)$ y una verosimilitud Gaussiana, la posterior sobre $f(\cdot)$ es también un \mathcal{GP} . Además, se puede condicionar sobre las observaciones $(X, f(X))$ para obtener*

$$f(X_*)|f(X), X \sim \mathcal{N}(m_{X_*|X}, \Sigma_{X_*|X}) \quad (9.7)$$

Donde la media y covarianza son:

$$m_{X_*|X} = m(X_*) + K(X_*, X)K^{-1}(X, X)(f(X) - m(X)) \quad (9.8)$$

$$\Sigma_{X_*|X} = K(X_*, X_*) - K(X_*, X)K^{-1}(X, X)K(X, X_*) \quad (9.9)$$

Ahora, dado un conjunto de observaciones y dada una función de covarianza, podemos obtener la densidad posterior. Para mostrar esto tomemos el caso de hacer regresión para una función conocida,

para la cual tenemos observaciones sin ruido muestreadas no uniformemente, con estas observaciones queremos encontrar la función real de las que provienen; para esto usamos un prior \mathcal{GP} con función media nula y kernel SE (por el momento tendrá parámetros fijos), nos damos un rango donde queremos hacer predicción y condicionamos en las observaciones usando la Ec.(9.7). En este caso las observaciones corresponden al 15 % de los puntos generados por nuestra función sintética.

Esto se muestra en la Fig.58, donde podemos ver que la media de la posterior pasa por las observaciones sin incertidumbre asociada, es decir que para estos puntos se tiene una posterior degenerada pues no hay varianza. Se puede ver que a medida que la predicción se aleja de las observaciones el intervalo de confianza (al que lo podemos asociar con incertidumbre del modelo) va creciendo.

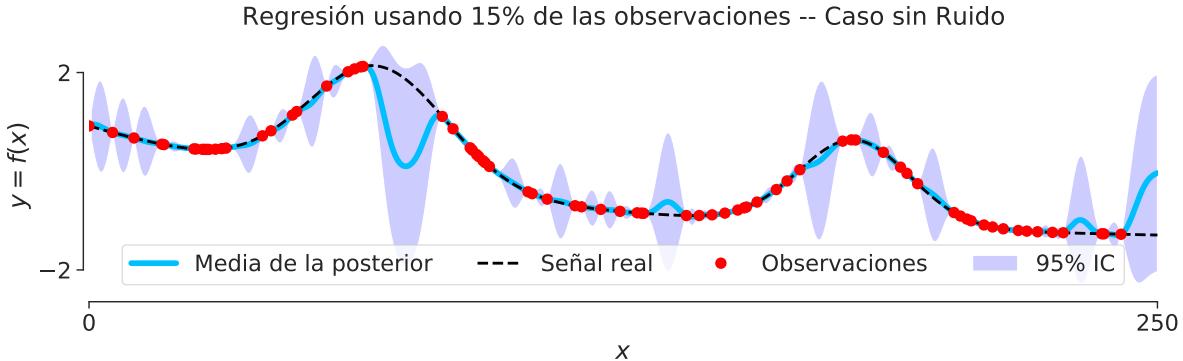


Fig. 58. Regresión con \mathcal{GP} para señal sintética usando el 15 % de los datos muestreados de forma no uniforme, utilizando un \mathcal{GP} de media nula y kernel SE.

9.2.2. Evaluación con ruido

En aplicaciones reales el caso de tener observaciones sin ruido es poco habitual, por lo que si queremos incorporar la incertidumbre real debemos tomar en cuenta errores de medición en nuestro modelo. Tomemos el caso de tener observaciones contaminadas con ruido i.i.d (independientes idénticamente distribuidas) donde las observaciones serán de la forma $y_i = f(x_i) + \eta$ donde $\eta \sim \mathcal{N}(0, \sigma_n^2)$, por lo que ahora nuestro conjunto de observaciones es de la forma (X, Y) donde $Y = f(X) + \eta$.

Lo que en nuestro modelo equivale a agregar un término a la función de covarianza

$$\text{cov}(Y) = K(X, X) + \sigma_n^2 \mathbb{I} \quad (9.10)$$

Donde si tenemos el mismo caso anterior, observaciones (X, Y) y queremos evaluar en X_* , la conjunta queda

$$\begin{bmatrix} Y \\ f(X_*) \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} m(X) \\ m(X_*) \end{bmatrix}, \begin{bmatrix} K(X, X) + \sigma_n^2 \mathbb{I} & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix} \right) \quad (9.11)$$

Notemos que el término de ruido solo es agregado al subbloque correspondiente a las observaciones, no se agrega el término en los otros subbloques pues buscamos hacer una predicción de la función latente $f(\cdot)$ y no una versión ruidosa de esta. Igual que en el caso sin ruido, podemos condicionar esta conjunta a las observaciones y obtenemos el siguiente resultado:

Proposición 9.0.2. *Para una evaluación con ruido se tiene que*

$$f(X_*)|Y, X \sim \mathcal{N}(m_{X_*|X}, \Sigma_{X_*|X}) \quad (9.12)$$

Donde la media y covarianza son:

$$m_{X_*|X} = m(X_*) + K(X_*, X)[K(X, X) + \sigma_n^2 \mathbb{I}]^{-1}(Y - m(X)) \quad (9.13)$$

$$\Sigma_{X_*|X} = K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 \mathbb{I}]^{-1}K(X, X_*) \quad (9.14)$$

Si tomamos el mismo ejemplo anterior, pero añadimos el ruido al modelo, obtenemos la predicción de la Fig.59. En este caso podemos ver que la media de la posterior no necesariamente coincide su valor con el de la observación, pues se toma en cuenta la incertidumbre en las observaciones mismas, también se ve que no se obtienen soluciones degeneradas incluso en zonas donde hay observaciones aglomeradas.

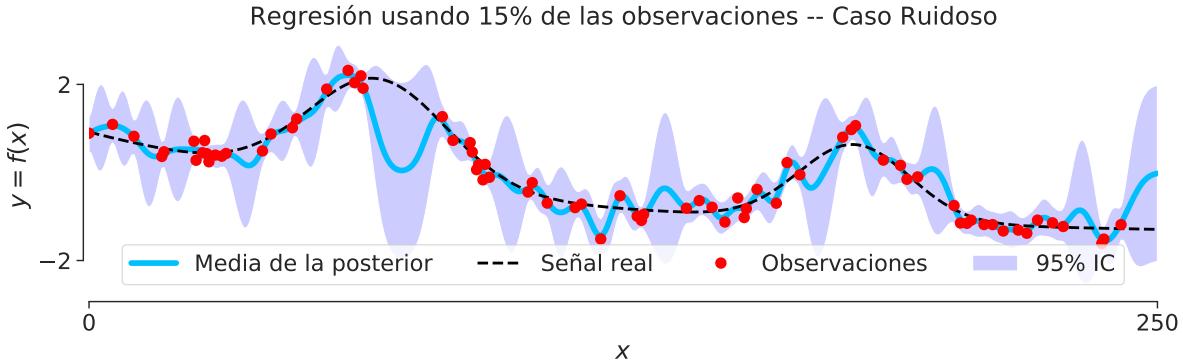


Fig. 59. Regresión con \mathcal{GP} para señal sintética usando el 15 % de los datos muestreados de forma no uniforme y contaminados con ruido Gaussiano, utilizando un \mathcal{GP} de media nula y kernel SE.

9.3. Entrenamiento y optimización de un \mathcal{GP}

hasta el momento nos hemos dado la función de covarianza y sus parámetros, por lo que aún no hemos hecho realizado el *aprendizaje* de nuestro \mathcal{GP} , que es lo que haremos a continuación.

9.3.1. ¿Qué es entrenar un \mathcal{GP} ?

Vimos que dada una función de covarianza podemos representar el proceso, y podemos encontrar analíticamente la densidad posterior de nuestra función $f(\cdot)$ condicionando a las observaciones. Pero la forma que tendrá la posterior y la función fuera de las observaciones dependerá fuertemente en nuestra función kernel escogida, en este sentido, para un kernel dado nos gustaría encontrar los parámetros de este que mejor representen nuestra función a estimar.

Nos referiremos a entrenar u optimizar un \mathcal{GP} cuando queremos obtener los hyperparámetros, es decir los parámetros del kernel (los denotamos θ) y la varianza del ruido (la denotamos σ_n^2) si es que aplica.

Para esto nos gustaría poder comparar funciones de covarianza, o mejor aún un funcional que podamos optimizar, afortunadamente podemos usar la *verosimilitud marginal*, obtenida marginalizando sobre la función $f(\cdot)$, donde dado un conjunto de entrenamiento $(X, Y) = \{(x_i, y_i)\}_{i=1}^n$, esta dada por

$$\mathbb{P}(Y|X, \theta, \sigma) = \int \mathbb{P}(Y|f, X, \theta, \sigma_n)p(f|X, \theta, \sigma)df \quad (9.15)$$

$$= \frac{1}{(2\pi|\mathbf{K}_y|)^{\frac{n}{2}}} \exp\left(-\frac{1}{2}(Y - \mathbf{m})^T \mathbf{K}_y^{-1} (Y - \mathbf{m})\right) \quad (9.16)$$

Donde $\mathbf{m} = m(X)$ y $\mathbf{K}_y = K_\theta(X, X) + \sigma_n^2 \mathbb{I}$, la matriz de covarianza dados los parámetros θ agregando el término de la diagonal correspondiente al ruido. De la misma forma que lo hacemos con otros modelos probabilísticos, en vez de maximizar la verosimilitud, es conveniente minimizar la log-verosimilitud negativa (NLL) dada por la expresión:

$$NLL = -\log \mathbb{P}(Y|X, \theta, \sigma_n) \quad (9.17)$$

$$NLL = \underbrace{\frac{1}{2} \log |\mathbf{K}_y|}_{\text{Penalización por complejidad}} + \underbrace{\frac{1}{2} (Y - \mathbf{m})^T \mathbf{K}_y^{-1} (Y - \mathbf{m})}_{\text{Data fit (Única parte que depende de } Y\text{)}} + \underbrace{\frac{n}{2} \log 2\pi}_{\text{Constante de normalización}} \quad (9.18)$$

De izquierda a derecha, el primer término tiene el rol de penalizar por la complejidad del modelo, y vemos que depende solo del kernel y las entradas; el segundo término cuantifica que tan bien se ajusta el modelo a los datos, y es la única componente que depende de las observaciones Y (ruidosas) de la función, el último término es una constante de normalización.

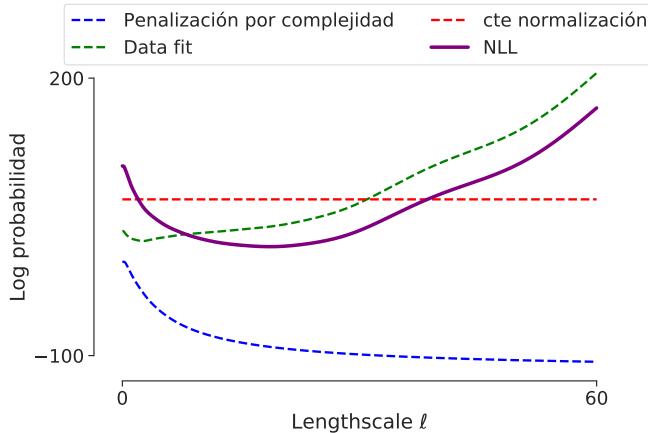


Fig. 60. Log verosimilitud marginal negativa (NLL) en función del *lengthscale* (ℓ) para señal sintética, se mantienen constantes los otros parámetros del \mathcal{GP} .

Siguiendo con los ejemplos anteriores, para el mismo conjunto de observaciones ruidosas, se calculan las tres componentes de la *NLL*, utilizando un kernel SE, para este caso, se dejan fijo tanto la varianza de la señal como la varianza del ruido y se varía el *lengthscale* ℓ del kernel, en la Fig.60.

Al ir variando ℓ se ve que la penalización por complejidad va disminuyendo, pues a mayor ℓ menos complejas son las funciones (recordar Fig.57, a mayor ℓ más suaves y regulares las muestras), también vemos que la componente del ajuste de datos comienza a decaer y luego se mantiene en incremento, pues a mayor *lengthscale* el modelo se vuelve menos flexible, por lo que no es capaz de ajustarse de manera correcta a los datos. De esta forma, el proceso de entrenamiento dará preferencia a funciones que se ajusten bien a los datos sin ser tan complejas, ahora viendo el valor total del NLL vemos que este alcanza su mínimo en ℓ en el rango de [10 – 30].

Ya tenemos una noción de que es entrenar un \mathcal{GP} , queremos minimizar la *NLL* (Ec.(9.18)) y encontrar los parámetros del kernel y del ruido (si aplica). Ahora discutiremos formas de optimizar este funcional.

9.3.2. ¿Cómo se entrena un \mathcal{GP} ?

Como contamos con una expresión cerrada para la NLL, podemos utilizar métodos clásicos de optimización, una opción es calcular el gradiente de esta función objetivo y aplicar algún método basado en gradiente, como L-BFGS; otra es utilizar el método de Powell que no requiere que la función sea diferenciable, por lo que no utiliza gradiente.

Siguiendo ejemplos anteriores, usando la misma señal sintética y las mismas observaciones ruidosas de la Fig.59, ahora optimizamos nuestro \mathcal{GP} utilizando L-BFGS, lo que nos entrega como resultado el mostrado en la Tabla.2 donde comparamos los parámetros del \mathcal{GP} sin entrenar usado en la Fig.59, podemos ver que no difiere mucho en cuanto a las varianzas, pues como es una señal sintética poseímos control sobre la varianza del ruido, el valor obtenido luego de optimizar es cercano al valor real (0.2), vemos que el *lengthscale* óptimo es concordante con lo analizado en la Fig.60.

Por último, podemos ver la predicción usando este \mathcal{GP} optimizado en la Fig.61 donde vemos que se tiene una del proceso más concordante con la función real, esto se ve especialmente en la zona con pocas observaciones, entre 50 y 100 para x , donde el \mathcal{GP} sin entrenar presentaba mucha incertidumbre en esa zona, en cuanto ahora se tiene un intervalo de confianza más bien limitado pues la señal no era muy compleja.

| | σ_{ruido} | ℓ | $\sigma_{\text{señal}}$ | NLL |
|--------------|-------------------------|---------|-------------------------|----------------|
| Sin entrenar | 0.2 | 3.1622 | 1 | 55.3538 |
| Entrenado | 0.2067 | 18.7267 | 0.9956 | 17.6945 |

Tabla 2. Resultado optimización \mathcal{GP} y comparación con los parámetros sin entrenar.

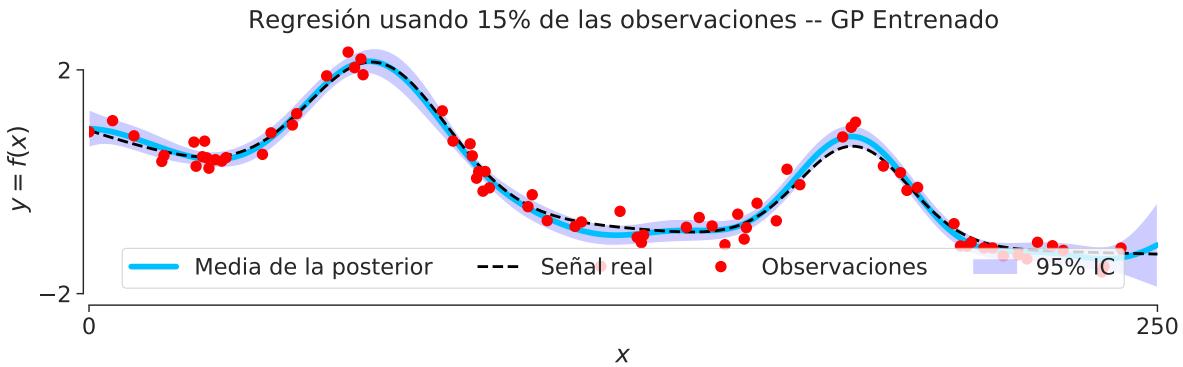


Fig. 61. Regresión con \mathcal{GP} para señal sintética usando el 15 % de los datos muestreados de forma no uniforme y contaminados con ruido Gaussiano, utilizando un \mathcal{GP} de media nula y kernel SE; Modelo optimizado utilizando L-BFGS.

La desventaja de utilizar métodos tradicionales de optimización es que estos entregan una solución puntual (en el sentido de un valor para cada parámetro, el \mathcal{GP} sigue siendo una distribución sobre funciones), en distintas aplicaciones puede que un valor fijo de parámetros no represente de manera idónea el proceso latente a estudiar. Una opción sería computar la densidad posterior de los parámetros del kernel condicionado a las observaciones. Lamentablemente en la mayoría de los casos no existe una expresión cerrada para esta posterior, por lo que debemos recurrir a métodos de *inferencia aproximada* como Markov Chain Monte Carlo (MCMC) o Inferencia Variacional.

9.3.3. Complejidad computacional

Es importante reconocer una de las principales desventajas de utilizar un \mathcal{GP} cuando se cuenta con una gran cantidad de datos, esto es, su costo computacional. Recordando, cuando queremos entrenar nuestro \mathcal{GP} vamos a minimizar la log verosimilitud marginal negativa (NLL), mostrada en la Ec.(9.18), donde al observar en segundo término vemos que es la operación más costosa siendo $\mathcal{O}(n^3)$ con respecto al número de puntos de entrenamiento n . Hay que tomar en cuenta que la evaluación de la NLL se debe hacer en cada iteración del método de optimización elegido.

En cuanto a la evaluación, esta está dada por la Ec.(9.12), donde en este caso vemos que es de orden cuadrático $\mathcal{O}(n^2)$ con respecto al número de puntos de test. Es de notar que esta vez no tomamos en cuenta la inversa de la matriz de Gram del conjunto de entrenamiento, pues puede ser precalculada y ser usada para múltiples conjuntos de test.

Lo anterior mencionado limita la cantidad de problemas que se pueden abordar usando estos procesos, sin embargo existen formas de obtener aproximaciones que siguen manteniendo la estructura y deseables propiedades de los \mathcal{GP} a un menor costo computacional, estos reducen el orden de $\mathcal{O}(n^3)$ a $\mathcal{O}(nm^2)$ en entrenamiento y $\mathcal{O}(m^2)$ en test, con $m < n$. Un ejemplo de estos son los *Sparse Gaussian Processes* que utilizan una aproximación de rango bajo para la matriz de covarianza, utilizando *pseudo-inputs* en vez del conjunto de entrenamiento completo.

9.4. Funciones de covarianza (kernels)

Una función de covarianza es una función de dos argumentos donde cualquier matriz que se obtiene como resultado en la evaluación de un conjunto de puntos (la llamaremos matriz de Gram) es semidefinida positiva.

Hasta el momento solo hemos utilizado un tipo de función de covarianza, el llamado kernel *Squared Exponential* (SE), conocido también como kernel RBF, dado por la Ec.(9.19). En esta sección mostraremos distintos tipos de funciones de covarianza y los distintos tipos de funciones que generan.

$$K_{SE}(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right) \quad (9.19)$$

Es importante denotar tipos de familias de funciones de covarianza, si la función solo depende de la diferencia, es decir $k(x, x') = k(x - x')$ se le llamará *kernel estacionario*, más aún, si depende solo de la norma de la diferencia $k(x, x') = k(|x - x'|)$ se le llamará *kernel isométrico*, un ejemplo de esto es el kernel SE.

Es de notar que kernels estacionarios hacen que la covarianza entre puntos sea invariante a traslaciones en el espacio de entradas. Una noción importante es que un kernel puede ser visto como una medida de similaridad entre puntos, y en el caso de kernels estacionarios, mientras más cercanos estén dos puntos más símiles serán.

9.4.1. Rational Quadratic (RQ)

Este kernel viene dado por la Ec.(9.20), este puede ser interpretado como una suma infinita de kernels SE con distintos *lengthscale*, donde α es un parámetro de variación de escala, notar que cuando $\alpha \rightarrow \infty$ el kernel tiende a uno SE. En la Fig.62 se muestra el kernel RQ, a la izquierda se ve el valor de la covarianza en función de la diferencia de los argumentos $x - x'$, a la derecha se muestran diferentes muestras de funciones con este kernel.

$$K_{RQ}(x, x') = \sigma^2 \left(1 + \frac{(x - x')^2}{2\alpha\ell^2}\right)^{-\alpha} \quad (9.20)$$

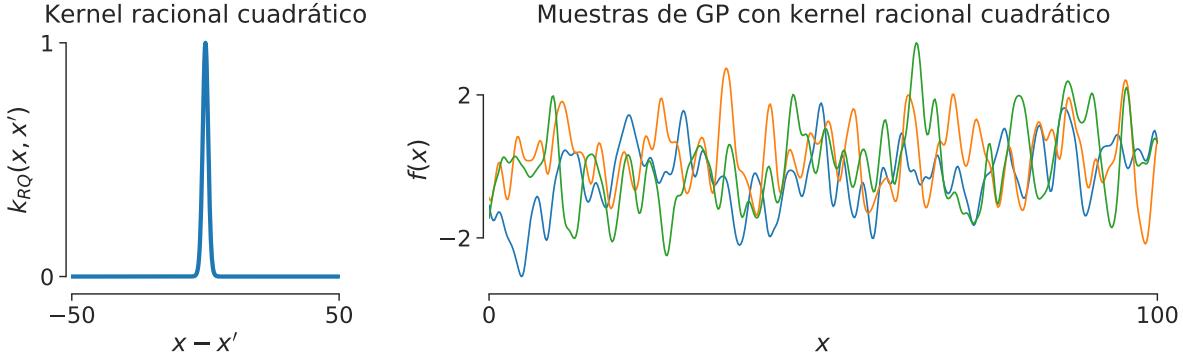


Fig. 62. Kernel *Rational Quadratic*, en la izquierda se muestra la covarianza en función de su argumento $\tau = x - x'$, a la derecha de un \mathcal{GP} usando un kernel RQ.

9.4.2. Kernel periódico

Como su nombre lo indica, este kernel, dado por la Ec.(9.21), permite modelar funciones periódicas, donde el parámetro p controla el periodo de la función. Una extensión de este el kernel localmente periódico, dado por la Ec.(9.22) que es un kernel SE ponderado por uno periódico, este da la libertad de tener funciones con una estructura periódica local.

$$K_P(x, x') = \sigma^2 \exp\left(-\frac{2 \sin^2(\pi|x - x'|/p)}{\ell^2}\right) \quad (9.21)$$

$$K_{LP}(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right) \exp\left(-\frac{2 \sin^2(\pi|x - x'|/p)}{\ell^2}\right) \quad (9.22)$$

En la Fig.63 se muestra el kernel periódico, a la izquierda se ve el valor de la covarianza en función de la diferencia de los argumentos $x - x'$, a la derecha se muestran diferentes muestras de funciones con este kernel.

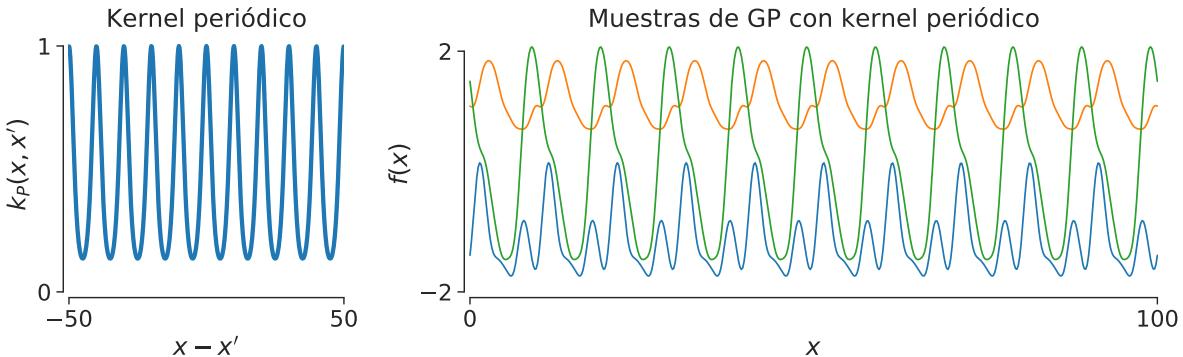


Fig. 63. Kernel periódico, en la izquierda se muestra la covarianza en función de su argumento $\tau = x - x'$, a la derecha de un \mathcal{GP} usando un kernel periódico.

9.4.3. Operaciones con kernels

A la hora de diseñar un \mathcal{GP} y elegir una función de covarianza, no se está completamente limitado a kernels conocidos, sino que también se pueden combinar para obtener distintas funciones de covarianza y representar mejor el proceso.

Tanto la suma y multiplicación de kernels da como resultado un kernel válido que puede ser utilizado, también la exponencial de un kernel es también es un kernel, es decir $\exp(k_1(\cdot, \cdot))$ con k_1 un kernel válido.

9.4.4. Representación espectral

Un teorema importante para las funciones de covarianza en procesos débilmente estacionarios es el teorema de Wiener-Khinchin, el cual dice que si para un proceso débilmente estacionario existe una función de covarianza $k(\tau)$ finita y definida para cualquier $\tau = x - x'$, entonces existe una función $S(\xi)$ tal que:

$$k(\tau) = \int S(\xi) e^{2\pi i \xi \cdot \tau} d\xi, \quad S(\xi) = \int k(\tau) e^{-2\pi i \xi \cdot \tau} d\tau \quad (9.23)$$

Donde i es la unidad imaginaria. $S(\xi)$ es conocida como la densidad espectral de potencia (PSD), en otras palabras, la función de covarianza $k(\tau)$ y la PSD $S(\xi)$ son duales de Fourier el uno del otro. Esto es extremadamente útil a la hora de diseñar funciones de covarianza pues este puede ser llevado a cabo en el dominio de la frecuencia y luego llevado a una función de covarianza usando la transformada inversa de Fourier.

9.5. Extensiones para un \mathcal{GP}

A continuación veremos un número de extensiones que se le pueden hacer a un \mathcal{GP} para abordar distintos tipos de problemas.

9.5.1. \mathcal{GP} de clasificación

Hasta el momento hemos visto como usar un \mathcal{GP} para regresión, pero este también puede usarse para clasificación, para esto simplemente “pasamos” nuestro \mathcal{GP} por una función logística, para así obtener un prior sobre $\sigma(f(x))$ donde σ es la función logística. Sin embargo esto trae consigo un problema, pues ahora la distribución posterior a las observaciones no se tiene de forma analítica como para el caso de regresión, esto lleva a que tengamos que recurrir a métodos aproximados de inferencia. Una solución simple es utilizar la aproximación de Laplace, pero si se quieren aproximaciones más fidedignas métodos más complejos pueden ser usados como *Expectation Maximization* y métodos MCMC. En la Fig.64 se muestra un ejemplo con datos sintéticos, a diferencia de la mayoría de los clasificadores vistos, este entrega naturalmente una densidad de probabilidad para la función de decisión.

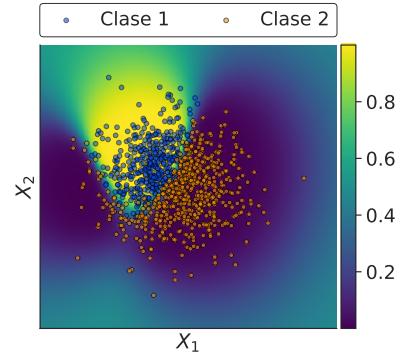


Fig. 64. \mathcal{GP} de clasificación utilizando datos sintéticos. Este clasificador entrega una densidad de probabilidad en vez de una sola función de decisión.

9.5.2. Selección automática de relevancia (ARD) (*Selección automática de features*)

Un \mathcal{GP} define una densidad de probabilidad sobre funciones, donde estas funciones son del tipo $f : \mathbb{R}^D \rightarrow \mathbb{R}$, con D es finito, este es nuestra dimensión de entrada o “características”. Haciendo un pequeño cambio en nuestra función kernel podemos hacer que esta automáticamente seleccione las entradas más relevantes con el problema, es decir realice una selección de características automática.

Si tomamos el kernel de la Ec.(9.24) vemos que es una multiplicación de kernels SE, donde se tiene un *lengthscale* por cada entrada ℓ_d , sabemos que mientras más grande es este ℓ_d menos flexible será el \mathcal{GP} respecto a cambios en ese eje, haciendo que las funciones del proceso dependan cada vez menos de la

componente d a medida que $\ell_d \rightarrow \infty$. De esta forma se puede controlar de forma automática la relevancia de cada eje del conjunto de entrada, pues los parámetros del kernel se obtienen en el entrenamiento. De esta forma estamos optimizando también en que grado afecta cada variable en nuestra predicción.

$$k(x, x') = \sigma^2 \exp\left(-\sum_{d=1}^D \frac{(x_d - x'_d)^2}{2\ell_d^2}\right) \quad (9.24)$$

9.5.3. Multi output \mathcal{GP}

Hasta el momento solo hemos hablado de \mathcal{GP} cuando nuestro proceso es solo una dimensión de salida. Se pueden extender los procesos Gaussianos a funciones de más de una salida o canal, donde ahora la función de covarianza $k(x, x')$ no entrega un escalar sino una matriz definida positiva, donde la diagonal corresponde a la covarianza del canal o autocovarianza y los elementos fuera de la diagonal corresponden a las covarianzas cruzadas o cross-covarianza. Este tipo de procesos Gaussianos aumentan considerablemente de complejidad al diseñar funciones de covarianza.

Dado un número m de canales, se tendrán m funciones de autocovarianza y ahora $m(m-1)/2$ funciones de covarianza y $k(x, x')$ será una matriz de $m \times m$. El desafío está en diseñar o escoger estas funciones de tal forma que para cualquier par de puntos x, x' la matriz $k(x, x')$ sea definida positiva.

Una opción simple es asumir que los canales son independientes entre sí, lo que equivale a entrenar independientemente m procesos Gaussianos, uno para cada canal, esto facilita el diseño de las funciones de covarianza pero hace que se pierdan relaciones entre los canales.

9.6. Diferentes interpretaciones de un \mathcal{GP}

9.6.1. De regresión lineal a \mathcal{GP}

Partiendo de la regresión lineal donde el modelo es:

$$y_i = wX_i + \epsilon_i \quad (9.25)$$

Donde $\epsilon_i \sim \mathcal{N}(0, \sigma_n^2)$, luego podíamos extender este modelo usando M funciones base ϕ_m y obtener:

$$y_i = \sum_{m=1}^M w_m \phi_m(x_i) + \epsilon_i \quad (9.26)$$

El paso siguiente es la regresión Bayesiana en base de funciones, donde agregamos un prior sobre los pesos $w_m \sim \mathcal{N}(0, \lambda_m^2)$, donde si obtenemos la covarianza de este proceso y marginalizamos por los pesos w_m obtenemos:

$$\text{Cov}(x, x') = k(x, x') = \sum_{m=1}^M \lambda_m^2 \phi_m(x)^T \phi_m(x') + \delta_{x,x'} \sigma_n^2 \quad (9.27)$$

Donde $\delta_{x,x'}$ el delta de Kronecker. Si nos damos cuenta esto define un \mathcal{GP} con la función covarianza con un número finito de funciones bases. En general los \mathcal{GP} corresponderán a covarianzas con infinitas funciones bases, como veremos a continuación.

Tomando un modelo sin ruido, con un número M de funciones base ϕ_m , donde sobre los pesos se define un prior i.i.d $w_m \sim \mathcal{N}(0, \sigma^2)$, tenemos:

$$f(x) = \sum_{m=1}^M w_m \phi_m(x) \quad (9.28)$$

Y tomando $\phi = \exp(-\frac{1}{2\ell^2}(x - c_i)^2)$ donde c_i son los centros de estas bases, y luego haciendo tender el número de funciones base M a infinito tenemos que la covarianza es:

$$\mathbb{E} \{ f(x)f(x') \} = \sigma^2 \sum_{m=1}^M \phi_m(x)\phi_m(x') \quad \text{tomando } M \rightarrow \infty \quad (9.29)$$

$$\mathbb{E} \{ f(x)f(x') \} \rightarrow \sigma^2 \int e^{-\frac{1}{2\ell^2}(x-c)^2} e^{-\frac{1}{2\ell^2}(x'-c)^2} dc \quad (9.30)$$

$$= \sigma^2 \sqrt{\pi\ell^2} e^{-\frac{1}{4\ell^2}(x-x')^2} \quad (9.31)$$

$$= k_{SE}(x, x') \quad (9.32)$$

Donde vemos que efectivamente el kernel SE es una función de covarianza para una composición infinita de funciones base.

9.6.2. Nota sobre RKHS

Dado un conjunto de entrenamiento $(X, Y) = \{x_i, y_i\}_{i=1}^n$ condicionando el proceso solo a una muestra de test $X_* = x_*$ y asumiendo una función media nula, de la Ec.(9.7) para un \mathcal{GP} la posterior de la media está dada por:

$$\bar{f}(x_*) = m_{x_*|X} = k(X, x_*)(k(X, X) + \sigma_n \mathbb{I})^{-1} Y \quad (9.33)$$

Y tomamos el vector $\boldsymbol{\alpha} = (k(X, X) + \sigma_n \mathbb{I})^{-1} Y$ obtenemos la expresión:

$$\bar{f}(x_*) = \sum_{i=1}^n \alpha_i k(x_i, x_*) \quad (9.34)$$

Donde, a pesar de que el proceso esta descrito por (posiblemente infinita) funciones base, aún así es la suma de finitos términos, cada uno centrado en un punto de entrenamiento, esto es debido al teorema del representante de los Espacios de Hilbert de Kernel Reproductor (RKHS). La intuición detrás de esto es que incluso si el \mathcal{GP} induce una distribución conjunta sobre todos los $y = f(x)$, una para cada x en el dominio, al hacer predicciones en el punto x_* solo nos interesa la distribución $(n+1)$ -dimensional definida por los puntos de entrenamiento más este punto de test.

10. Anexos

10.1. ¿Qué hizo efectivamente Bayes y por qué?

Thomas Bayes fue uno de los primeros *inconformistas anglicanos*.¹³ Su trabajo de 1763, titulado

An Essay Towards Solving a Problem in the Doctrine of Chances

esbozó por primera vez el resultado que hoy conocemos como el Teorema de Bayes. Este trabajo fue terminado por el amigo y colega de Bayes, Richard Price (1723 – 1791), el cual envió el artículo a la prestigiosa revista inglesa *Philosophical Transactions of the Royal Society* (PTRS), donde fue publicado póstumamente. Este artículo consideraba el caso de *invertir la distribución binomial*: en un experimento donde el resultado puede ser éxito (con probabilidad p) o fracaso (con probabilidad $1 - p$), el objetivo es encontrar la distribución del parámetro p dado que se han observado q éxitos luego de n experimentos. Para resolver este caso particular, Bayes descubrió que efectivamente la distribución posterior es proporcional a la verosimilitud. Sin embargo, a pesar de que Price completara el trabajo de Bayes luego de la repentina muerte de éste último, este trascendental descubrimiento quedó momentáneamente en el olvido. Fue finalmente Laplace quien en 1774 publicó la relación entre las distribuciones prior y posterior como la conocemos hoy, sin dejar de reconocer el trabajo de Bayes. En efecto, en su *Essai Philosophique dur les Probabilités* (1814), Laplace menciona que Bayes ya había llegado al mismo resultado de forma “refinada y muy ingeniosa, aunque un poco confusa”.

Existen dudas sobre la autoría de Bayes del artículo en cuestión y cuánto de éste efectivamente venía de las propias notas no publicadas de Bayes, y cuánto de la edición y cálculos de R. Price (Bellhouse, 2004). En efecto, al año siguiente del artículo original, Price publicó en 1764 otro artículo en PTRS llamado *A Demonstration of the Second Rule in the Essay toward the Solution of a Problem in the Doctrine of Chances*, esta vez de su propia autoría. Tanto Bayes como Price estaban muy lejos de ser matemáticos prolíficos de su época, de hecho, las publicaciones de ambos no están particularmente ligadas a las matemáticas sino que son de índole religiosa—por mucho que sus contribuciones hayan sido suficientemente conceptuales y generales para ser consideradas como avances en ambos campos. Una posibilidad para entender las motivaciones de Bayes (y Price) para el desarrollo de los artículos mencionados, pueden encontrarse en el artículo (Stigler, 2013) el cual postula que el verdadero título del artículo de Bayes aparentemente es:

A Method of Calculating the Exact Probability of All Conclusions founded on Induction.

Este título, según explica (Stigler, 2013), pudo haber sido perdido en la edición del artículo o probablemente el propio editor de PTRS lo modificó por encontrarlo un tanto *atrevido*. Si este fuese efectivamente el título, y no el menos informativo título original, se entiende que el artículo tiene por objetivo caracterizar el problema más general de *inducción*. Esto tiene un sentido particular dado el contexto filosófico y religioso en ese entonces, pues pocos años antes la obra de Hume ‘Of Miracles’ (1748) presentaba un argumento probabilístico para desestimar los milagros (como la resurrección). En pocas palabras, Hume postulaba que la considerable *improbabilidad* de un milagro sobrepasaba ampliamente la *probabilidad* de que el milagro fuese incorrectamente documentado. Este ensayo de Hume fue ampliamente leído, discutiendo y—evidentemente—atacado, en particular, tanto Bayes como Price, ambos inconformistas anglicanos, consideraron el ensayo de Hume como una agresión. Consecuentemente, Bayes consideró responder al argumento de Hume mediante la aplicación de probabilidades al problema de inducción, lo cual es confirmado por las notas no publicadas de Bayes con fecha anterior al 31 de diciembre de 1749 (Bellhouse, 2004), las que eventualmente llegaron al famoso artículo de 1763. Sin embargo, el trabajo de Bayes no estaba completo, en particular, no contaba una satisfactoria aproximación de la distribución posterior

¹³Este término se usa para denotar a los protestantes que no estaban de acuerdo con los métodos y la gobernanza eclesiástica establecida por la “Iglesia de Inglaterra” (The Church of England).

en el ejemplo de la inversión de la distribución binomial mencionado anteriormente. En este sentido, la motivación de Price para terminar este trabajo no era únicamente concluir el trabajo póstumo de su amigo, sino que también desarrollar una respuesta eficaz contra el argumento de Hume. En efecto, luego de una serie de trabajos relacionados, fue finalmente en 1767 que Price logró publicar su disertación *On the Importance of Christianity, its Evidences, and the Objections which have been made to it*, donde hace una crítica directa a “Of miracles”. Básicamente, Price sentencia que Hume habría subestimado el impacto que una (gran) cantidad de observadores **independientes** reportando la existencia del milagro puede tener. En dicho caso, el Teorema de Bayes mostraba que la multiplicación de evidencia, la cual puede ser falible de forma individual, podría sobreponer la improbabilidad de un evento (el milagro) y establecerlo como verdad (o en realidad muy probable).

Existen varias razones atribuibles a que el trabajo de Bayes haya sido ignorado parcialmente hasta Laplace, e.g., la incompletitud del trabajo del propio Bayes, el rol incierto que tuvo Price en el desarrollo de éste, y el nombre poco elocuente del artículo que probablemente desvió atención al trabajo. Todo esto nos hace cuestionarnos si es correcto referirnos a este trascendental Teorema en honor al Rev. Thomas Bayes, el cual lanzó el primer atisbo de este resultado y no a Laplace, el cual fue el primero en enunciar la forma general para la relación entre prior, posterior y modelo en la forma que hoy conocemos y usamos.

10.2. Álgebra lineal

10.2.1. Cálculo matricial

Para dos vectores $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} \in \mathbb{R}^m$, el operador $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ puede ser definido como una matriz de dos formas distintas dependiendo del orden en el que se completen las entradas de la matriz:

- Notación jacobiana (numerator-layout): la matriz se completa de acuerdo a \mathbf{y} y \mathbf{x}^\top , es decir:

$$\left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)_{ij} = \frac{\partial y_i}{\partial x_j} \implies \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \in \mathcal{M}_{mn}(\mathbb{R})$$

- Notación hessiana (denominator-layour): la matriz se completa de acuerdo a x e y^\top , es decir:

$$\left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)_{ij} = \frac{\partial y_j}{\partial x_i} \implies \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \in \mathcal{M}_{nm}(\mathbb{R})$$

La notación estandar corresponde a la notación jacobiana y es la que se sigue a lo largo del apunte. Bajo esta notación, la derivada de un campo vectorial (gradiente) corresponde a un vector vertical cuyas componentes son las derivadas de las componentes del campo.

Sean \mathbf{x} , $\mathbf{u} = \mathbf{u}(\mathbf{x})$, $\mathbf{v} = \mathbf{v}(\mathbf{x})$ y \mathbf{a} vectores, \mathbf{g} campo vectorial y A matriz de dimensiones apropiadas. Bajo la convención de la notación jacobiana, se tienen las siguientes fórmulas de derivación:

- Transformación lineal:

$$\frac{\partial(A\mathbf{x})}{\partial \mathbf{x}} = A \implies \begin{cases} \frac{\partial \mathbf{x}}{\partial \mathbf{x}} = I \\ \frac{\partial(\mathbf{x}^\top A)}{\partial \mathbf{x}} = A^\top \text{ ya que } \frac{\partial \mathbf{u}^\top}{\partial \mathbf{x}} = \left(\frac{\partial \mathbf{u}}{\partial \mathbf{x}} \right)^\top \end{cases} \quad (10.1)$$

- Producto punto:

$$\frac{\partial(\mathbf{u} \cdot \mathbf{v})}{\partial \mathbf{x}} = \mathbf{u}^\top \frac{\partial \mathbf{v}}{\partial \mathbf{x}} + \mathbf{v}^\top \frac{\partial \mathbf{u}}{\partial \mathbf{x}} \implies \begin{cases} \frac{\partial(\mathbf{a} \cdot \mathbf{x})}{\partial \mathbf{x}} = \mathbf{a}^\top \\ \frac{\partial(\mathbf{x} \cdot \mathbf{x})}{\partial \mathbf{x}} = \frac{\partial \|\mathbf{x}\|^2}{\partial \mathbf{x}} = 2\mathbf{x}^\top \end{cases} \quad (10.2)$$

- Forma cuadrática:

$$\frac{\partial(\mathbf{u}^\top A \mathbf{v})}{\partial \mathbf{x}} = \mathbf{u}^\top A \frac{\partial \mathbf{v}}{\partial \mathbf{x}} + \mathbf{v}^\top A^\top \frac{\partial \mathbf{u}}{\partial \mathbf{x}} \implies \begin{cases} \frac{\partial(\mathbf{x}^\top A \mathbf{x})}{\partial \mathbf{x}} = \mathbf{x}^\top (A + A^\top) = 2\mathbf{x}^\top A \text{ si } A \text{ es simétrica.} \\ \frac{\partial^2(\mathbf{x}^\top A \mathbf{x})}{\partial \mathbf{x} \partial \mathbf{x}^\top} = A + A^\top = 2A \text{ si } A \text{ es simétrica.} \end{cases} \quad (10.3)$$

- Regla de la cadena:

$$\frac{\partial(\mathbf{g}(\mathbf{u}))}{\partial \mathbf{x}} = \frac{\partial \mathbf{g}(\mathbf{u})}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}} \quad (10.4)$$

Por otra parte, si x es una variable escalar e $Y \in \mathcal{M}_{mn}(\mathbb{R})$ es una matriz dependiente de x , se define $\frac{\partial Y}{\partial x}$ como la matriz Y con el operador derivada aplicado a cada entrada, es decir:

$$\left(\frac{\partial Y}{\partial x} \right)_{ij} = \frac{\partial Y_{ij}}{\partial x} \implies \frac{\partial Y}{\partial x} \in \mathcal{M}_{mn}(\mathbb{R})$$

Bajo esta definición, si U, V son matrices dependientes de x de dimensiones apropiadas, entonces se tienen las siguientes reglas de derivación:

- Regla del producto:

$$\frac{\partial(UV)}{\partial x} = U \frac{\partial V}{\partial x} + \frac{\partial U}{\partial x} V \quad (10.5)$$

- Inversa (para U invertible):

$$\frac{\partial U^{-1}}{\partial x} = -U^{-1} \frac{\partial U}{\partial x} - U^{-1} \quad (10.6)$$

Por último, sea $X \in \mathcal{M}_{mn}(\mathbb{R})$ una matriz e y un escalar dependiente de X . Bajo la notación jacobiana, se tiene la siguiente definición:

$$\left(\frac{\partial y}{\partial X} \right)_{ij} = \frac{\partial y}{\partial X_{ji}} \implies \frac{\partial y}{\partial X} \in \mathcal{M}_{nm}(\mathbb{R})$$

Sean $u = u(X), v = v(X)$ escalares, g función real, H función matricial y A, B matrices. Se tienen las siguientes reglas de derivación:

- Producto de escalares:

$$\frac{\partial uv}{\partial X} = u \frac{\partial v}{\partial X} + v \frac{\partial u}{\partial X} \quad (10.7)$$

- Regla de la cadena:

$$\frac{\partial g(u)}{\partial X} = \frac{\partial g(u)}{\partial u} \frac{\partial u}{\partial X} \quad (10.8)$$

- Operador traza:

$$\frac{\partial \text{Tr}(H(X))}{\partial X} = H'(X), \text{ en particular} \quad \begin{cases} \frac{\partial \text{Tr}(AX)}{\partial X} = A \\ \frac{\partial \text{Tr}(x^\top AX)}{\partial X} = X^\top (A + A^\top) \\ \frac{\partial \text{Tr}(AXB)}{\partial X} = BA \\ \frac{\partial \text{Tr}(X^n)}{\partial X} = nX^{n-1} \end{cases} \quad (10.9)$$

Sean X matriz invertible, entonces

$$\frac{\partial |X|}{\partial X} = |X| X^{-T} \quad (10.10)$$

$$\frac{\partial x^\top X^{-1} x}{\partial X} = -X^{-\top} x x^\top X^{-\top} \quad (10.11)$$

10.2.2. Rango e inversa de Moore-Penrose

Definición 10.1 (rango). Para una matriz $M \in \mathcal{M}_{mn}(\mathbb{R})$ se define su rango como el número máximo de filas (equivalentemente, columnas) linealmente independientes que tiene la matriz.

Por lo tanto, una matriz cuadrada es invertible si y solo si tiene rango máximo (todas sus filas y columnas son l.i.). De este modo, se tiene la siguiente propiedad:

Proposición 10.0.1. *Sea $A \in \mathcal{M}_{mn}(\mathbb{R})$ entonces $A^\top A \in \mathcal{M}_{nn}(\mathbb{R})$ es invertible si y solo si A tiene todas sus columnas l.i.*

En efecto, dado que $r(M_1 M_2) \leq \min\{r(M_1), r(M_2)\}$ entonces, $A^\top A \in \mathcal{M}_{nn}(\mathbb{R})$ es invertible si y solo si $r(A^\top A) = n \iff r(A) = n$, por lo que A tiene n columnas (y filas) l.i.

Debido a esto es que para poder utilizar la fórmula de regresión lineal mediante MC o MCR necesariamente $\hat{X} \in \mathcal{M}_{N,M+1}(\mathbb{R})$ debe ser de rango $M + 1$ (máximo por columnas), lo cual es equivalente a pedir que las $M + 1$ observaciones sean l.i.

Definición 10.2 (pseudoinversa de Moore-Penrose). Sea $A \in \mathcal{M}_{mn}(\mathbb{R})$. Otra matriz $A^+ \in \mathcal{M}_{nm}(\mathbb{R})$ se dice que es la pseudoinversa de A si:

- (inversa débil) $AA^+A = A$ y $A^+AA^+ = A^+$
- (simetría) $(AA^+)^\top = AA^+$ Y $(A^+A)^\top = A^+A$

Se puede probar que la pseudoinversa siempre existe. Además, por el lema anterior se tiene que si A es de rango completo, entonces la pseudoinversa tiene forma cerrada:

- A tiene columnas l.i. $\implies A^+ = (A^\top A)^{-1}A^\top$ y es inversa por izquierda: $A^+A = I$.
- A tiene filas l.i. $\implies A^+ = A^\top(AA^\top)^{-1}$ y es inversa por derecha: $AA^+ = I$.

De este modo, en la regresión por MC, la solución $\theta = \hat{X}^+Y$ puede ser interpretada como el proceso de despejar θ en la hipótesis $Y = X\theta$.

10.2.3. Fórmula de Woodbury

Esta fórmula permite descomponer la inversa de una suma particular de matrices.

Teorema 10.1 (Fórmula de Woodbury). *Para matrices de dimensiones adecuadas, se tiene la siguiente fórmula de inversión:*

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \quad (10.12)$$

Esta descomposición puede ser demostrada directamente probando que al multiplicar $A + UCV$ por la igualdad de la derecha se obtiene la identidad.

10.3. Optimización

10.3.1. Teorema de Karush-Kuhn-Tucker

Sean $f : E \subset \mathbb{R}^n \rightarrow \mathbb{R}$, $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$ y $h : \mathbb{R}^n \rightarrow \mathbb{R}^p$ funciones diferenciables. Un problema de optimización con estructura de Karush-Kuhn-Tucker viene dado por:

$$\begin{aligned} (P) \min_{s.a.} & f(x) \\ & g(x) \leq 0 \\ & h(x) = 0 \\ & x \in E \end{aligned} \tag{10.13}$$

Donde $g(x) \leq 0 \iff g_i(x) \leq 0, \forall i \in \{1, \dots, m\}$.

Sea x_0 un punto factible de (P) tal que $\{\{\nabla g_i(x_0)\}_{i=1}^m, \{\nabla h_i(x_0)\}_{i=1}^p\}$ es linealmente independiente. Si x_0 es solución de (P) , entonces existen $\mu \in \mathbb{R}^m$ y $\lambda \in \mathbb{R}^p$ tal que

$$\begin{aligned} f(x) + \langle \lambda, g(x_0) \rangle + \langle \mu, h(x_0) \rangle &= 0 \\ u_i g_i(x_0) &= 0 \\ u_i &\geq 0 \end{aligned} \tag{10.14}$$

10.3.2. Método del gradiente clásico

El método del gradiente es un método numérico iterativo que busca aproximar la solución del problema irrestricto

$$(P) \min_{x \in \mathbb{R}^n} f(x) \tag{10.15}$$

Donde las iteraciones son de la forma $x_{k+1} = x_k + \beta_k d_k$, con d_k una dirección de descenso y β_k la magnitud del desplazamiento (paso). El método del gradiente considera d_k como la dirección de máximo descenso:

$$d_k = \arg \min_{\|d\|=1} \langle \nabla f(x_k), d \rangle \implies d_k = \frac{-\nabla f(x_k)}{\|\nabla f(x_k)\|} \tag{10.16}$$

Ya que el producto interno es máximo cuando los dos vectores son paralelos. Por otra parte, para β_k se considerará el desplazamiento óptimo, el cual está dado por un problema de optimización unidimensional:

$$\beta_k = \arg \min_{\beta \geq 0} f(x_k + \beta d_k) \tag{10.17}$$

De este modo, se tendrá un algoritmo iterativo que genera secuencias decrecientes en el valor de la función: $f(x_n) \geq f(x_m)$ para $m \geq n$.

Bajo ciertas condiciones sobre f (convexidad y gradiente lipschitz) y elecciones particulares de β_k , se puede probar que el algoritmo converge a un mínimo global.

El algoritmo se detiene cuando se alcanza una cierta tolerancia de error, la cual es obtenida cuando la norma del gradiente es suficientemente pequeña, lo que indica que no habrá mucha variación entre la iteración actual y la siguiente.

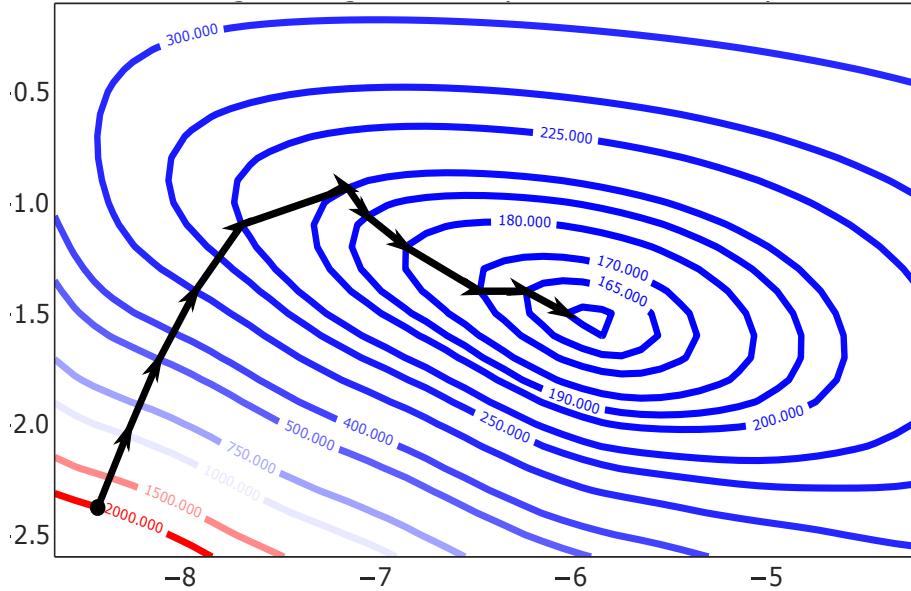


Fig. 65. Iteraciones del método del gradiente para una función de ejemplo.

10.3.3. Dualidad lagrangiana

Sean $f : E \subset \mathbb{R}^n \rightarrow \mathbb{R}$, $g : \mathbb{R}^n \rightarrow \mathbb{R}^m$ y $h : \mathbb{R}^n \rightarrow \mathbb{R}^p$ funciones diferenciables y (P) un problema de optimización con estructura de Karush-Kuhn-Tucker:

$$(P) \begin{aligned} & \min_{s.a} f(x) \\ & g(x) \leq 0 \\ & h(x) = 0 \\ & x \in E \end{aligned} \tag{10.18}$$

El lagrangiano de (P) corresponde a $L(x, \lambda, \mu) := f(x) + \langle \lambda, g(x) \rangle + \langle \mu, h(x) \rangle$, donde λ y μ se denominan multiplicadores de Lagrange o variables duales. Se define la función lagrangiana dual como:

$$\theta(\lambda, \mu) := \inf_{x \in E} L(x, \lambda, \mu) \tag{10.19}$$

De este modo, se tiene el problema lagrangiano dual:

$$(D) \max_{\lambda \geq 0} \theta(\lambda, \mu) \tag{10.20}$$

Teorema 10.2 (dualidad fuerte). *Sea $E \subset \mathbb{R}$ convexo no vacío, f, g convexas y h afín (es decir, de la forma $h(x) = Ax + b$). Supóngase que:*

- $\exists x \in E : g(x) < 0, h(x) = 0$.
- $0 \in \text{int}(Im(h))$.

Entonces, (P) y (D) tienen el mismo valor óptimo límite (como ínfimo y supremo) y si dicho valor es finito, entonces el supremo es alcanzado en el problema dual. Además, si en (P) el ínfimo es alcanzado en $\hat{x} \in E$ entonces $\langle \mu, g(\hat{x}) \rangle = 0$.

En el caso de un problema convexo con restricciones de desigualdad diferenciables, el ínfimo del lagrangiano es alcanzado donde se anula su gradiente, de este modo, la función lagrangiana dual tiene forma explícita y se tiene que:

$$\begin{aligned} (P) \min_{s.a.} f(x) &\implies (D) \max_{\lambda \geq 0} L(x, \lambda) = f(x) + \langle \lambda, g(x) \rangle \\ g(x) \leq 0 & \quad \nabla_x L(x, \lambda) = \nabla f(x) + \langle \lambda, \nabla g(x) \rangle = 0 \end{aligned} \tag{10.21}$$

Teorema 10.3 (holgura complementaria). *Bajo las hipótesis de dualidad fuerte, si el ínfimo de (P) es alcanzado en \hat{x} y el supremo de (D) es alcanzado en $(\hat{\lambda}, \hat{\mu})$ entonces $\lambda_i f_i(\hat{x}) = 0, \forall i \in \{1, \dots, m\}$.*

Referencias

- Bayes, T. (1763). An essay towards solving a problem in the doctrine of chances. *Phil. Trans. R. Soc.*, 53, 370–418. Descargado de <https://doi.org/10.1214/13-STS438> doi: 10.1098/rstl.1763.0053
- Bellhouse, D. R. (2004). The reverend thomas bayes, frs: A biography to celebrate the tercentenary of his birth. *Statistical Science*, 19(1), 3–32.
- Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and Trends® in Machine Learning*, 2(1), 1-127.
- Bengio, Y. (2016). *What's yoshua bengio's opinion on max welling's position paper äre ml and statistics complementary?* Descargado de <https://www.quora.com/Whats-Yoshua-Bengios-opinion-on-Max-Wellings-position-paper-Are-ML-and-Statistics-Complementary>
- Ben-Israel, A., y Greville, T. (2006). *Generalized inverses: Theory and applications*. Springer New York.
- Boser, B. E., Guyon, I. M., y Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. En *Proceedings of the fifth annual workshop on computational learning theory* (pp. 144–).
- Bostrom, N. (2014). *Superintelligence: Paths, dangers, strategies*. Oxford University Press.
- Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24, 123–140.
- Breiman, L. (2001). Random forests. *Machine learning*, 45, 5–32.
- Breiman, L., Friedman, J. H., Olshen, R. A., y Stone, C. J. (1984). *Classification and regression trees*. Wadsworth & Brooks.
- Davenport, T. H., y Patil, D. (2012). *Data scientist: The sexiest job of the 21st century*. Descargado de <https://hbr.org/2012/10/data-scientist-the-sexiest-job-of-the-21st-century>
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., y Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. En *Cvpr09*.
- Farley, B., y Clark, W. (1954). Simulation of self-organizing systems by digital computer. *IRE Transactions on Information Theory*, 4(4), 76-84.
- Fayyad, U., Piatetsky-Shapiro, G., y Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, 17(3).
- Ferguson, T. S. (1973). A bayesian analysis of some nonparametric problems. *Annals of Statistics*, 1(2), 209– 230.
- Freund, Y., y Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1), 119–139.
- Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol. Cybern.*, 36, 193–202.
- Gal, Y. (2015). *The science of deep learning*. Descargado de YarinGals' swebsite:http://mlg.eng.cam.ac.uk/yarin/blog_5058.html
- Geurts, P., Ernst, D., y Wehenkel, L. (2006). Extremely randomized trees. *Machine learning*, 63, 3–42.
- Ghahramani, Z. (2015). Probabilistic machine learning and artificial intelligence. *Nature*, 521, 452–459.
- Harari, Y. N. (2015). *Sapiens: A brief history of humankind*. Harper.
- Hastie, T., Tibshirani, R., y Friedman, J. (2001). *The elements of statistical learning*. Springer.
- Hjort, N., Holmes, C., Müller, P., y Walker, S. (2010). *Bayesian nonparametrics*. Cambridge University Press.
- Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the USA*, 79, 2554–2558.
- James, G., Witten, D., Hastie, T., y Tibshirani, R. (2014). *An introduction to statistical learning: With applications in r*. Springer.
- Jordan, M. I., Ghahramani, Z., Jaakkola, T., y Saul, L. (1999). An introduction to variational methods for graphical models. *Machine Learning*, 37, 183-233.
- LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., y Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4), 541-551.
- Lighthill, J. (1973). Artificial intelligence: A general survey. *Artificial Intelligence: a paper symposium*.

- Minsky, M. (1952). *A neural-analogue calculator based upon a probability model of reinforcement* (Inf. Téc.). Boston, MA: Harvard University Psychological Laboratories.
- Minsky, M., y Papert, S. (1969). *Perceptrons: an introduction to computational geometry*. MIT.
- Murphy, K. P. (2022). *Probabilistic machine learning: An introduction*. MIT Press. Descargado de probml.ai
- Neal, R. M. (1993). *Probabilistic inference using markov chain monte carlo methods* (Inf. Téc.). Toronto, Canada: University of Toronto, Department of Computer Science.
- Pierce, G. (1949). *The song of insects*. Harvard College Press.
- Rasmussen, C. E., and Williams, C. K. (2006). *Gaussian processes for machine learning*. The MIT Press.
- Ripley, B. D. (2008). Pattern recognition and neural networks. En (cap. 7: Tree-structured Classifiers).
- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386–408.
- Rumelhart, D. E., Hinton, G. E., y Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 23(6008), 533-536.
- Russell, S. J., y Norvig, P. (2009). *Artificial intelligence: A modern approach* (3.^a ed.). Pearson Education.
- Salakhutdinov, G. E. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313, 504-507.
- Schapire, R. E., y Freund, Y. (2012). *Boosting: Foundations and algorithms. adaptive computation and machine learning*. Mit Press London.
- Shannon, C. (1950). Programming a computer for playing chess. *Philosophical Magazine*, 41(324).
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., ... Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529, 484–503. Descargado de <http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html>
- Stigler, S. M. (2013, 08). The true title of bayes's essay. *Statist. Sci.*, 28(3), 283–288. Descargado de <https://doi.org/10.1214/13-STS438> doi: 10.1214/13-STS438
- Tibshirani, R. (1996). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1), 267-288.
- Tikhonov, A. N., y Arsenin, V. Y. (1977). *Solution of ill-posed problems*. Washington: Winston & Sons.
- Turing, A. (1950). Computing intelligence and machinery. *Mind*, 59(236), 433-460.
- Vapnik, V., and Chervonenkis, A. . (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16, 264-280.
- Welling, M. (2015). Are ml and statistics complementary? *Roundtable discussion at the 6th IMSISBA meeting on “Data Science in the next 50 years”*.
- Werbos, P. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences* (Tesis Doctoral no publicada). Harvard University.