



TRABAJO FIN DE GRADO
DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y
MATEMÁTICAS

Análisis de los algoritmos de Boosting en entornos con ruido de clase.

Rafael Nogales Vaquero

Autor

Rafael Nogales Vaquero (alumno)

Directores

Francisco Herrera Triguero (tutor1)

Julián Luengo Martín (tutor2)



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

Granada, 7 Agosto de 2018

Análisis de los algoritmos de Boosting en entornos con ruido de clase.

Autor

Rafael Nogales Vaquero (alumno)

Directores

Francisco Herrera Triguero (tutor1)

Salvador García López (tutor2)

Análisis de los algoritmos de Boosting en entornos con ruido de clase.

Rafael Nogales Vaquero

Palabras clave: Machine Learning, Boosting, Ruido de clase, Función de pérdida, Problemas de Clasificación

Resumen

En este TFG se propone cambiar la función de pérdida de los algoritmos de tipo boosting con el objetivo de hacerlos más robustos a conjuntos de datos de entrenamiento parcialmente mal clasificados.

Analysis of Boosting algorithms in environments with class noise.

Rafael Nogales Vaquero

Keywords: Machine Learning, Boosting, Class noise, Loss function, Classification problems

Abstract

In this TFG it is proposed to change the loss function of the boosting algorithms in order to make them more robust to partially misclassified training data sets.

Yo, **Rafael Nogales Vaquero**, alumno de la titulación DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 76669550Q, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Rafael Nogales Vaquero

Granada a 7 de Agosto de 2018

D. **Francisco Herrera Triguero** , Profesor del Área de XXXX del Departamento YYYY de la Universidad de Granada.

D. **Salvador García López** , Profesor del Área de XXXX del Departamento YYYY de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Título del proyecto, Subtítulo del proyecto***, ha sido realizado bajo su supervisión por **Rafael Nogales Vaquero** , y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a X de mes de 201 .

Los directores:

Francisco Herrera Triguero

Salvador García López

Agradecimientos

A mi familia, amigos y profesores

Capítulo 1

Introducción

1.1. Contexto

El objetivo final de este trabajo es desarrollar un mecanismo que haga que el algoritmo XGBoost (el algoritmo de Boosting más utilizado a nivel global) sea más robusto al ruido de clase.

Este algoritmo es utilizado en problemas de ambitos muy diferentes siempre que haya que trabajar con datos tabulados suele dar un gran resultado. Por desgracia hay muchos problemas con datos tabulados que son susceptibles de tener datos mal clasificados y esto es un gran problema para los algoritmos de boosting clásicos como Adaboost.

El problema principal radica en que para ajustarse rápidamente a los datos los algoritmos de boosting suelen utilizar funciones de pérdida exponenciales que dan una importancia enorme a los datos ruidosos.

Lo cual nos deja un amplio terreno sobre el que trabajar.

1.2. Solución

Debido a que el nuestro mayor problema es el sobreajuste debido a la función de pérdida exponencial ya que da un peso cada vez mayor a los datos difíciles de clasificar y los ruidosos siempre lo son. Es por esto que la estrategia a seguir para mitigar este problema va a consistir en utilizar otras funciones de pérdida más robustas al ruido. Previamente repasaremos todos los conceptos matemáticos relacionados con el boosting, las funciones de pérdida y el aprendizaje automático en general.

1.3. Herramientas

Clasificaremos nuestras herramientas en dos tipos: Herramientas matemáticas y herramientas software.

Para la parte matemática nuestras herramientas serán el cálculo diferencial, la estadística y el álgebra lineal.

En la parte informática utilizaremos principalmente R, Python y algunas de sus librerías: pandas, scikit-learn, xgboost, matplotlib...

1.4. Fuentes

Capítulo 2

Objetivos

2.1. Entender cómo funcionan los algoritmos de boosting

Lorem Ipsum es simplemente el texto de relleno de las imprentas y archivos de texto. Lorem Ipsum ha sido el texto de relleno estándar de las industrias desde el año 1500, cuando un impresor (N. del T. persona que se dedica a la imprenta) desconocido usó una galería de textos y los mezcló de tal manera que logró hacer un libro de textos especimen. No sólo sobrevivió 500 años, sino que también ingresó como texto de relleno en documentos electrónicos, quedando esencialmente igual al original. Fue popularizado en los 60s con la creación de las hojas "Letraset", las cuales contenían pasajes de Lorem Ipsum, y más recientemente con software de autoedición, como por ejemplo Aldus PageMaker, el cual incluye versiones de Lorem Ipsum.

2.2. Analizar cómo el ruido les afecta

Lorem Ipsum es simplemente el texto de relleno de las imprentas y archivos de texto. Lorem Ipsum ha sido el texto de relleno estándar de las industrias desde el año 1500, cuando un impresor (N. del T. persona que se dedica a la imprenta) desconocido usó una galería de textos y los mezcló de tal manera que logró hacer un libro de textos especimen. No sólo sobrevivió 500 años, sino que también ingresó como texto de relleno en documentos electrónicos, quedando esencialmente igual al original. Fue popularizado en los 60s con la creación de las hojas "Letraset", las cuales contenían pasajes de Lorem Ipsum, y más recientemente con software de autoedición, como por ejemplo Aldus PageMaker, el cual incluye versiones de Lorem Ipsum.

2.3. Proponer una mejora para hacerlos más robustos al ruido

Lorem Ipsum es simplemente el texto de relleno de las imprentas y archivos de texto. Lorem Ipsum ha sido el texto de relleno estándar de las industrias desde el año 1500, cuando un impresor (N. del T. persona que se dedica a la imprenta) desconocido usó una galería de textos y los mezcló de tal manera que logró hacer un libro de textos especimen. No sólo sobrevivió 500 años, sino que tambien ingresó como texto de relleno en documentos electrónicos, quedando esencialmente igual al original. Fue popularizado en los 60s con la creación de las hojas "Letraset", las cuales contenian pasajes de Lorem Ipsum, y más recientemente con software de autoedición, como por ejemplo Aldus PageMaker, el cual incluye versiones de Lorem Ipsum.

Capítulo 3

Aspecto Matemático

El Machine Learning tiene una gran base matemática puesto que su fundamento se basa en comprender una distribución de probabilidad desconocida que genera lo que observamos en la realidad. Nuestro trabajo consistirá en construir una función que se aproxime a la real y de la que podamos inferir resultados aplicables en la realidad.

3.1. Empirical risk minimization

El principio ERM (Empirical Risk Minimization) es un principio en la teoría estadística del aprendizaje que se usa para acotar de forma teórica el rendimiento de una familia de algoritmos de aprendizaje.

Consideremos la siguiente situación, que es muy general en la mayoría de problemas de aprendizaje supervisado. Tenemos dos espacios de objetos X e Y y queremos aprender una función $h : X \mapsto Y$ (a menudo llamada *hipótesis*) cuya salida, dado un $x \in X$ es un objeto $y \in Y$. Para hacerlo disponemos de un *conjunto de entranamiento* de m muestras: $(x_1, y_1), \dots, (x_m, y_m)$ donde $x_i \in X$ es una entrada y $y_i \in Y$ es su correspondiente salida, la cual nosotros pensamos que viene dada por la función de hipótesis $h(x_i) = y_i$.

Para formalizar esta idea, supongamos que existe una función de distribución conjunta $P(x, y)$ sobre X e Y y que el conjunto de entremamiento consiste en m muestras $(x_1, y_1), \dots, (x_m, y_m)$ independientes e idénticamente distribuidas de $P(x, y)$. Es importante notar que esta suposición nos permite modelar la incertidumbre en las predicciones (por ejemplo, ruido en los datos) ya que y no es una función determinística de x sino una variable aleatoria cuya función de distribución es $P(y|x)$ para un x fijo.

Supongamos además que tenemos una función real no negativa $L(\hat{y}, y)$ a la que llamaremos *Función de Pérdida* que mide cómo de diferente es la

predicción \hat{y} (una hipótesis) es de la realidad y . El *riesgo* asociado a una hipótesis $h(x)$ se define como la esperanza matemática de la función de pérdida:

$$R(h) = \mathbb{E}[L(h(x), y)] = \int L(h(x), y) dP(x, y)$$

Una función de pérdida muy utilizada a nivel teórico es la **0-1**:

$$L(\hat{y}, y) = I(\hat{y} \neq y)$$

Dónde $I(\dots)$ es la función indicadora.

El fin último de un algoritmo de aprendizaje es encontrar una función de hipótesis h^* en una familia de funciones \mathcal{H} para la cual el operador riesgo es minimal:

$$h^* = \arg \min_{h \in \mathcal{H}} R(h)$$

Todo lo hablado es muy bonito teóricamente, pero en general, el riesgo $R(h)$ no puede calcularse de ningún modo ya que la función de distribución $P(x, y)$ es desconocida para el algoritmo de aprendizaje. Sin embargo, podemos calcular una aproximación, llamada *riesgo empírico*, calculando la media de la función de pérdida en el conjunto de entrenamiento:

$$R_{\text{emp}}(h) = \frac{1}{m} \sum_{i=1}^m L(h(x_i), y_i)$$

El principio de minimización del riesgo empírico establece que el algoritmo de aprendizaje debe escoger la función de hipótesis \hat{h} que minimize el riesgo empírico:

$$\hat{h} = \arg \min_{h \in \mathcal{H}} R_{\text{emp}}(h).$$

Es decir el algoritmo de aprendizaje definido por el principio ERM consiste en resolver el problema de optimización recién planteado. Para lo cual se van a utilizar las técnicas clásicas de resolución de problemas de optimización y se utilizarán propiedades matemáticas de las funciones de pérdida tales como la convexidad y la diferenciabilidad cuando procedan.

3.2. Calculo Diferencial Multivariante

3.2.1. Optimización convexa

El algoritmo de gradiente descendente es un algoritmo para minimizar de forma aproximada funciones convexas. Sea $f : \Omega \subset \mathbb{R}^m \rightarrow \mathbb{R}$ una función

convexa, diferenciable y que alcanza su mínimo en Ω . Sabemos que el gradiente de f (∇f) es un vector que indica la dirección de máximo crecimiento de f en cada punto.

Por tanto, dado $x \in \Omega$ tenemos que $-\nabla f(x)$ es el vector que indica la dirección en la que f se minimiza más rápidamente desde x .

Para ilustrarlo intuitivamente podemos pensar en $f(\Omega)$ como una superficie suave y en $(x, f(x))$ como un punto sobre ella. Entonces si dejásemos caer una gota de agua justo sobre este punto, la gota se deslizaría en la dirección de $\nabla f(x)$ y además lo haría a una velocidad proporcional a $|\nabla f(x)|$.

El algoritmo de gradiente descendente consiste en construir una sucesión de los puntos por los que esta gota pasaría si bajase en pequeños tramos rectos y no de forma continua.

Formalmente: Sea $f : \Omega \subset \mathbb{R}^m \rightarrow \mathbb{R}$ una función convexa de clase $C^1(\Omega)$ con mínimo en p . Y sea $\{p_t\}$ una sucesión de puntos de Ω tal que $\{p_t\}$ converge a p . Para esto se elige un punto p_0 cualquiera en Ω y construimos desde él el resto de la sucesión recurrentemente:

$$p_{t+1} = p_t - \alpha_t \nabla f(p_t)$$

Dónde el parámetro α_t se selecciona de tal manera que $p_{t+1} \in \Omega$ y $f(p_t) \geq f(p_{t+1})$.

Nota: En el contexto del aprendizaje automático a este parámetro α se le suele denotar como η y se le denomina *tasa de aprendizaje*, además se le suele dar un valor constante y no se le exige ninguna restricción, lo que puede conllevar a que el gradiente descendente no descienda.

3.3. Algoritmos de aprendizaje y gradiente descendente

En nuestro caso particular nos interesa minimizar la función de pérdida de un algoritmo de aprendizaje. Recordamos que el objetivo de un algoritmo de aprendizaje según el principio ERM es minimizar el riesgo empírico:

$$R_{\text{emp}}(h) = \frac{1}{m} \sum_{i=1}^m L(h(x_i), y_i)$$

Por lo tanto lo que buscamos es un clasificador \hat{H} que minimice el riesgo empírico

$$\hat{H} = \arg \min_h \frac{1}{m} \sum_{i=1}^m [L(h(x), y)]$$

El algoritmo de gradient boosting considera $y \in \mathbb{R}$ y busca una aproximación $\hat{H}(x)$ construida como suma de clasificadores débiles ponderados. Es decir como suma de funciones $h_i(x)$ pertenecientes a una familia de funciones \mathcal{H} .

$$\hat{H} = \sum_{i=1}^M w_i h_i(x) + \text{const}$$

De acuerdo con el principio de ERM el algoritmo de boosting debe encontrar una aproximación \hat{H} que minimiza el error medio de la función de pérdida en el conjunto de entrenamiento. El método elegido por el boosting consiste en establecer un modelo inicial constante H_0 y mejorarlo incrementalmente usando la metodología greedy:

$$H_0(x) = \arg \min_w \sum_{i=1}^n L(y_i, w)$$

$$H_m(x) = H_{m-1}(x) + \arg \min_{h_m \in \mathcal{H}} \sum_{i=1}^n L(y_i, H_{m-1}(x_i) + h_m(x_i))$$

dónde $h_m \in \mathcal{H}$ es un clasificador débil.

Desafortunadamente, elegir la mejor función h en cada paso para una función de pérdida L arbitraria no es computable en general. Sin embargo podemos restringirnos a una versión simplificada del problema. La idea es aplicar **gradiente descendente** para aproximar la solución del problema de minimización. Si consideramos el caso continuo, es decir, donde \mathcal{H} es el conjunto de las funciones diferenciables sobre \mathbb{R} , las funciones $\mathcal{C}^\infty(\mathbb{R})$ podemos actualizar los pesos del modelo de acuerdo a las siguientes ecuaciones:

$$H_m(x) = H_{m-1}(x) - w_m \sum_{i=1}^n \nabla H_{m-1} L(y_i, H_{m-1}(x_i))$$

,

$$w_m = \arg \min_w \sum_{i=1}^n L(y_i, H_{m-1}(x_i) - w \nabla H_{m-1} L(y_i, H_{m-1}(x_i)))$$

dónde la derivada se toma respecto a las funciones H_i con $i \in \{1, \dots, m\}$.

En el caso discreto, es decir cuando \mathcal{H} tiene un conjunto finito de funciones, elegimos h como la función más cercana al gradiente de L para el cual el peso w puede ser calculado minimizando las ecuaciones de arriba con algún algoritmo iterativo sencillo.

Algoritmo boosting:**Entrada₁:** El conjunto de entrenamiento $\{(x_i, y_i)\}_{i=1}^n$ **Entrada₂:** Una función de pérdida diferenciable $L(y, H(x))$ **Entrada₃:** El número de iteraciones M .

Inicializar el modelo con un valor constante:

$$H_0(x) = \arg \min_w \sum_{i=1}^n L(y_i, w).$$

for m en $\{1, \dots, M\}$ **do**

Calcular el error asociado a esa iteración:

$$r_{i,m} = - \left[\frac{\partial L(y_i, H(x_i))}{\partial H(x_i)} \right]_{H(x)=H_{m-1}(x)} \quad \text{for } i = 1, \dots, n.$$

Entrenar un clasificador base (por ejemplo un árbol)

 $h_m(x)$ usando el conjunto de aprendizaje $\{(x_i, r_{i,m})\}_{i=1}^n$. Calcular el peso w_m solucionando el siguiente problema de optimización:

$$w_m = \arg \min_w \sum_{i=1}^n L(y_i, H_{m-1}(x_i) + w h_m(x_i)).$$

Actualizar el modelo:

$$H_m(x) = H_{m-1}(x) + w_m h_m(x)$$

end for**Salida:** $H_M(x)$ **El parámetro M y el sobreajuste:**

Ajustarnos mucho al conjunto de entrenamiento puede degradar el modelo como ya sabemos (caemos en sobreajuste). Además de utilizar funciones de pérdida que incluyan formas de regularización podemos minimizar el efecto del sobreajuste eligiendo un parámetro M adecuado.

Un M pequeño infrajustará el modelo y por el contrario uno demasiado grande caerá en sobreajuste.

Capítulo 4

Aspecto Informático

“Aim for simplicity in Data Science. Real creativity won’t make things more complex. Instead, it will simplify them.”

Damian Duffy Mingle

4.1. Introducción al Machine Learning

Hasta ahora muchas compañías de todos los tipos y tamaños se han dedicado a obtener un potente sistema de información que les permite almacenar de forma ordenada (o no tanto) todos los datos que se producen en el día a día de la empresa.

Sin embargo esta estrategia de por sí no sirve de nada, ya que si almacenas datos y no los utilizas nunca estás desperdiciando tus recursos. La ciencia de datos es la disciplina que transforma la información en conocimiento. Este conocimiento puede ser utilizado para la toma de decisiones de forma manual o de forma automática. El machine learning es el subcampo de la ciencia de datos que se centra en hacer que las máquinas aprendan de los datos y puedan realizar tareas concretas sin haber sido programadas específicamente para ello.

Actualmente podemos encontrar aplicaciones del machine learning por todas partes:

- Buscadores web: (Clasificación de contenido)
- Redes sociales: (Segmentación de clientes en función de los gustos propios y de amigos)

- Comunicación: (Detección de spam y clasificación automática de la bandeja de entrada)
- Seguridad: (Detección de malware)
- Entretenimiento: (Shazam: Reconocimiento de canciones)
- Marketing: (Netflix: Sistemas de recomendación)
- Medicina: (Diagnostico de enfermedades)
- Arte: (Generar imágenes artificialmente)

El Machine Learning no es un concepto nuevo, sin embargo en los últimos años ha tenido un gran auge. Una de las razones principales de este auge es el aumento en la capacidad de procesamiento y la disminución de los costes del mismo, permitiendo así que pueda estar al alcance de todos.

Aquí podemos ver el interés en el machine learning a nivel global utilizando Google Trends:



Definición 4.1 Machine Learning

Campo de estudio que da a los ordenadores la habilidad de aprender sin la necesidad de ser explícitamente programados. Arthur Samuel, 1959

4.1.1. Tipos de problemas en Machine Learning

Hay tres tipos de problemas de machine learning: **Aprendizaje supervisado**, **Aprendizaje no supervisado** y **Aprendizaje por refuerzo**.

4.1.2. Aprendizaje Supervisado

El objetivo principal del aprendizaje supervisado es construir un modelo a partir de un conjunto etiquetado de *datos de entrenamiento* que nos permita hacer predicciones sobre los datos sin etiquetar que vengan en el futuro. Se llama aprendizaje supervisado porque la clave del paradigma radica en tener datos *etiquetados* a priori. Hay dos tipos de problemas que se resuelven mediante aprendizaje supervisado: **Clasificación** y **Regresión**.

4.1.3. Aprendizaje No Supervisado

En el aprendizaje supervisado conocemos la *respuesta correcta* de antemano cuando entrenamos nuestro modelo. En el aprendizaje no supervisado, por contra, estamos tratando con datos sin etiquetar o con datos con *estructura desconocida*. Usando aprendizaje no supervisado podemos explorar la estructura de nuestros datos y extraer información útil observando únicamente similitudes y diferencias entre los datos para poder agruparlos de forma que salgan a relucir posibles patrones ocultos en los datos. La técnica más popular dentro del aprendizaje no supervisado es el *clustering*. Este método se utiliza cuando se necesita clasificar las instancias de datos pero no se conocen previamente las categorías. Esta agrupación permite construir grupos (*cluster*) de forma automática para clasificar de una forma lógica aun sin conocer las clases de antemano.

Un ejemplo clásico es examinar los datos de ventas de una compañía para obtener grupos de clientes similares.

4.1.4. Aprendizaje por refuerzo

En el aprendizaje por refuerzo, el objetivo consiste en desarrollar un sistema (*un agente*) que mejore su rendimiento basandose en interacciones con el entorno. Como la forma de aprender del entorno consite en la nueva incorporación de datos acompañados de una medida de la *función de recompensa*. Podemos pensar en el aprendizaje por refuerzo como un tipo de aprendizaje supervisado. Sin embargo en el aprendizaje por refuerzo esta etiqueta que acompaña a los nuevos datos no es una clase, como ocurre en el aprendizaje supervisado, sino que es una medida de como de bien o mal se está comportando el agente (valor de la función de recompensa asociado a los nuevos datos). A traves de la interacción con el entorno un agente puede utilizar aprendizaje por refuerzo para aprender una serie de acciones que maximizen su recompensa mediante ensayo-error o mediante un plan deliberado.

Un ejemplo clásico de aprendizaje por refuerzo es un programa que jue-

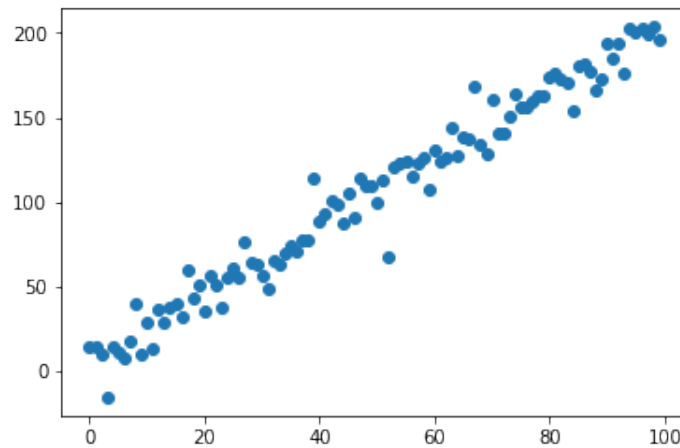
gue a algún juego como el ajedrez. En este caso, el agente decide la jugada basandose en el estado actual del tablero (entorno) y la recompensa puede determinarse como una función que evalúa cada situación y que lógicamente asigna la máxima puntuación a ganar y la mínima a perder.

4.2. Resolviendo problemas con Aprendizaje Supervisado

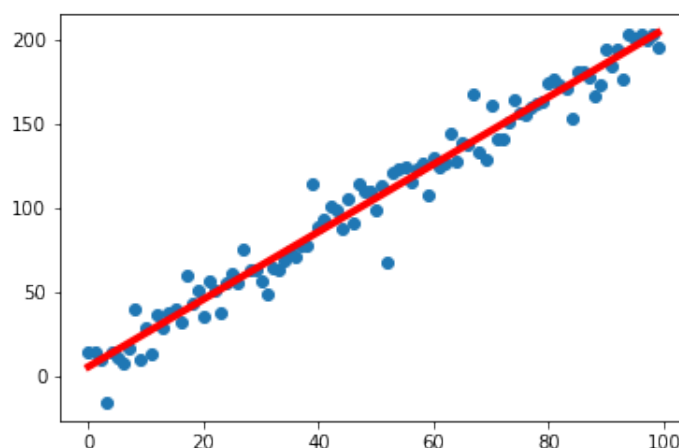
4.2.1. Método de Regresión

Este método se utiliza para predecir el valor de un atributo continuo. Consiste en encontrar la mejor ecuación que atraviese de forma óptima un conjunto de puntos (n-dimensiones). Esta mejor ecuación se va a buscar dentro de una familia de funciones, de esta forma hablaremos de *Regresión Lineal*, *Regresión Logística* etc... En estos métodos lo único que cambiamos es la familia de funciones, aunque también se pueden añadir mecanismos de regularización.

Supongamos que tenemos dos variables aleatorias X e Y :



Queremos buscar la mejor función lineal que exprese Y como función de X : Aplicando regresión lineal obtenemos una función lineal $LM(x) = w_0 + w_1x$ tal que $LM(x) \simeq Y(X)$

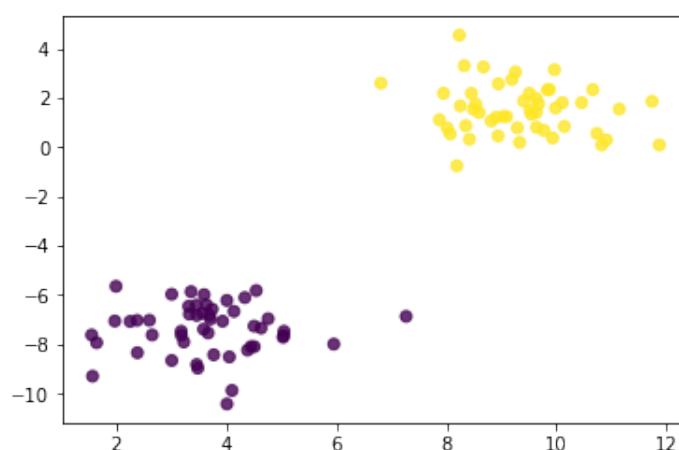


De esta forma ahora podríamos predecir el valor de nuevos datos colocándolos sobre la línea roja en la posición que les corresponda. Es decir: $y_{pred} = LM(X_{new})$

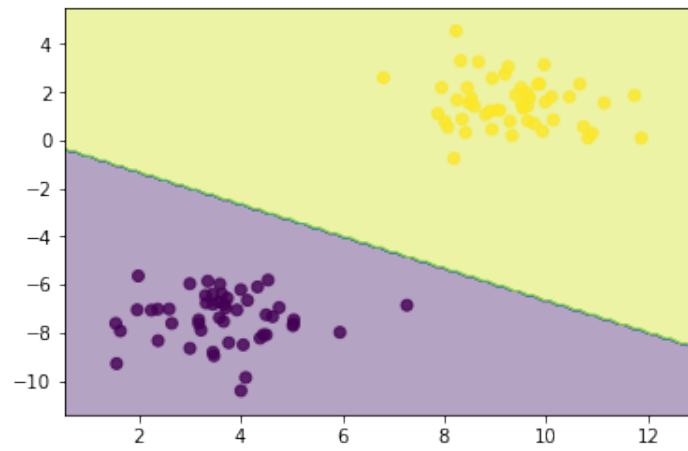
4.2.2. Método de Clasificación

Este método se utiliza para predecir a que clase pertenece una determinada instancia a partir de unas características determinadas. El tipo más simple de clasificación es la clasificación binaria mediante un clasificador lineal:

Ejemplo: Supongamos que tenemos datos pertenecientes a dos clases (la clase violeta y la clase amarilla):



Queremos buscar la mejor función lineal que separe ambas clases:



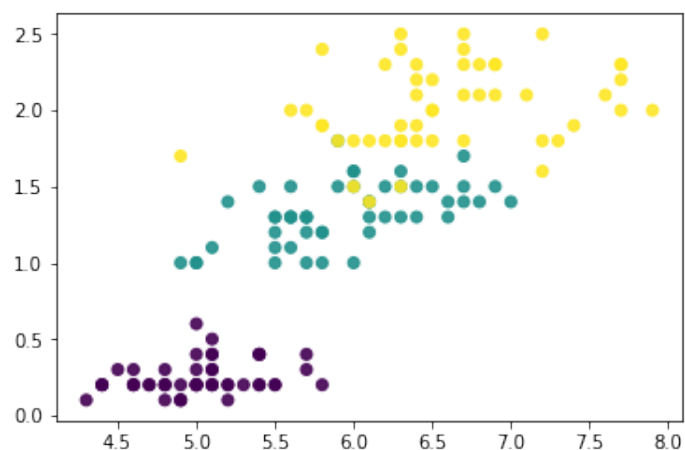
4.2.3. Iris: El problema de clasificación más clásico

En 1936 Ronald Fisher anotó la colección de datos *Iris*, en el que cuantificó varias mediciones taxonómicas de las flores de tres especies de Iris:



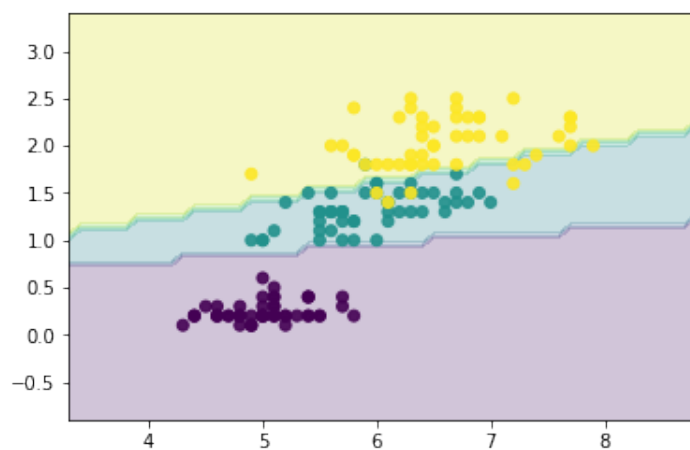
Figura 4.1: Especies de Iris.

Podemos establecer como variables la longitud del sépalo y la longitud del pétalo y obtenemos una representación en 2D del conjunto de Fisher:

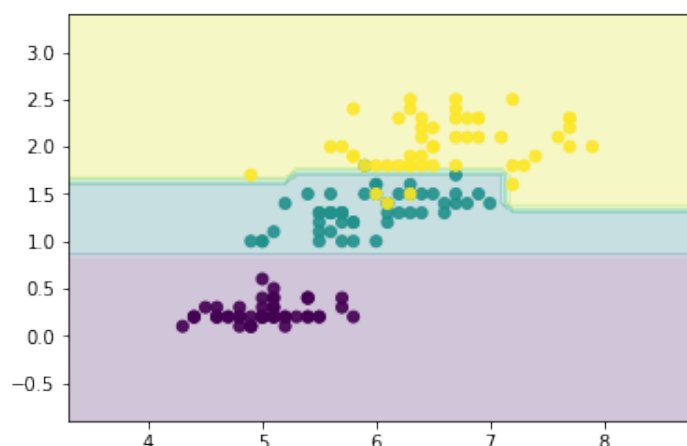


Y podemos establecer muchos tipos de clasificadores:

El clasificador lineal que utilizamos antes se vería de este modo:



Pero podemos utilizar también clasificadores de tipo Boosting, concretamente el Gradient Boosting Machine, del que hablaremos más adelante y que ocupará la mayor parte del trabajo:



4.3. El flujo de trabajo para clasificación

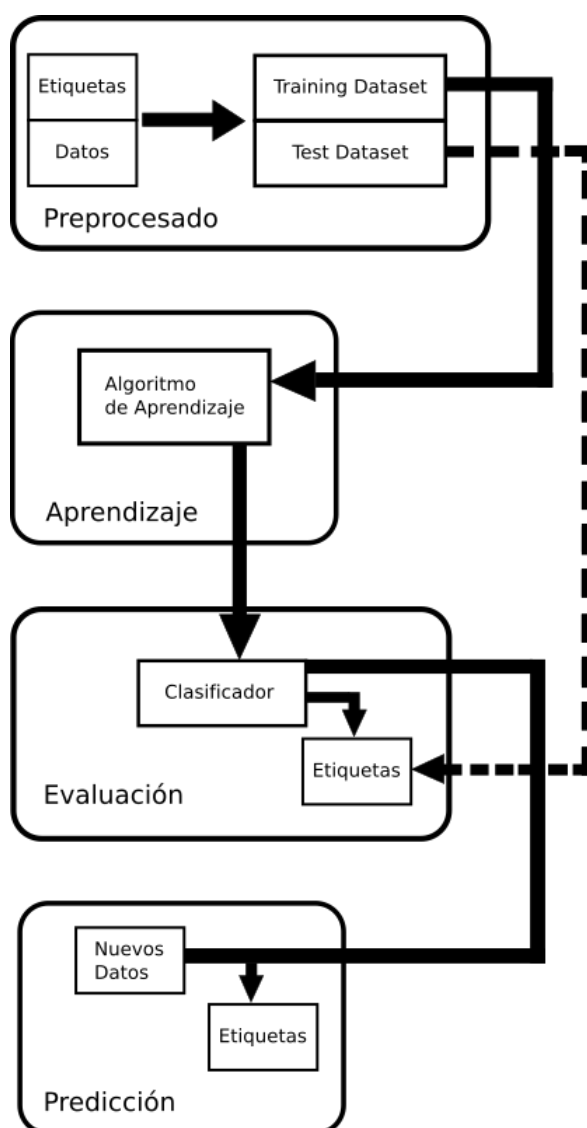
Todos los problemas de clasificación parten de un conjunto de datos etiquetados. Y el objetivo es construir una función que acepta nuevos datos con la misma estructura que en el conjunto inicial, pero sin etiqueta, y devuelve la etiqueta que considera que debe ser la correcta.

Lo primero, antes de nada es separar el dataset en datos de entrenamiento (training dataset) y en datos de test (test dataset). Los datos de test se guardarán hasta el final sin hacer nada sobre ellos y se trabajará únicamente con los datos de entrenamiento. Para construir un modelo de calidad, primero deben **preprocesarse** los datos de entrenamiento, es decir, normalizarlos, escalarlos, aplicar transformaciones, seleccionar qué variables son importantes y cuales no... Este es un subcampo bastante amplio llamado *Ingeniería de Características*. Normalmente el preprocesamiento es la parte que más tiempo requiere y a la que el científico de datos más esfuerzo debe dedicarle.

Una vez se han preprocesado los datos de entrenamiento se utilizarán para **ajustar un modelo**, es decir, se construirá un modelo mediante un algoritmo de aprendizaje.

Una vez el modelo se ha construido llega el turno de **evaluar la calidad del modelo**, lo mejor es utilizar cross validation con los datos de entrenamiento para evaluarlo y dejar los de test únicamente para evaluar los modelos que creemos que son buenos. Existen técnicas para ajustar los parámetros empíricamente como por ejemplo *gridsearch*.

Una vez tenemos un modelo de calidad podrá utilizarse para **predecir nuevos datos** y nuestro trabajo volverá a empezar para mejorar el modelo si así lo consideramos necesario.



4.4. Uso de Python para Machine Learning

En este caso vamos a utilizar Python, debido a que actualmente se está convirtiendo en el lenguaje más utilizado para ciencia de datos, además de estar ganando terreno a muchos otros lenguajes clásicos debido a la enorme cantidad de librerías que existen para Python. La mayor alternativa a Python es el lenguaje estadístico R. No obstante un científico de datos debería conocer ambos lenguajes ya que R es muy potente a la hora de hacer un buen análisis exploratorio de forma rápida y eficaz. En Python también es posible trabajar como con R de forma rápida e interactiva utilizando Jupyter Notebook.

Para lo que nos ocupa nos centraremos en utilizar las principales librerías de Python para Machine Learning:

- *numpy*: Es una librería de cálculo numérico para Python, se encarga de que las operaciones computacionalmente intensivas con matrices se hagan con rutinas C muy eficientes en lugar de directamente interpretadas con Python
- *pandas*: Es una librería para importar y exportar datos en Python, además con *pandas* es muy fácil manejar grandes tablas de forma cómoda ya que está construida sobre *numpy*
- *scipy*: Es otra colección de rutinas en Fortran y C para agilizar los cálculos científicos
- *scikit-learn*: Está contruida sobre *numpy* y *scipy* y tiene una gran cantidad de herramientas para poder completar el flujo de trabajo de un problema de machine learning sin necesidad de otras librerías en la mayoría de los casos sencillos
- *xgboost*: Es una librería que contiene la implementación del algoritmo XGBoost, el algoritmo de boosting más utilizado en la actualidad
- *matplotlib*: Es una librería para visualización de gráficos

4.4.1. Instalación de Python y de paquetes

Python está disponible para los tres grandes sistemas operativos (Linux, Microsoft Windows y Mac OS X). En la mayoría de distribuciones Linux modernas Python viene instalado por defecto en su versión moderna (Python3). En Mac no obstante aún viene por defecto la versión 2.7 (Python antiguo). Lo ideal en este caso es instalar Python3 con un alias (*python3*) y utilizarlo, pero sin desinstalar el Python del sistema, ya que el sistema operativo utiliza internamente la versión 2.7 para algunas tareas. En Windows

Python no viene instalado por defecto en ninguna versión por lo que hay que descargarlo de la web oficial: <https://www.python.org>

Todo el código que se ha utilizado en este proyecto está escrito para Python3 y debería funcionar en todas las versiones superiores a la 2.7.10

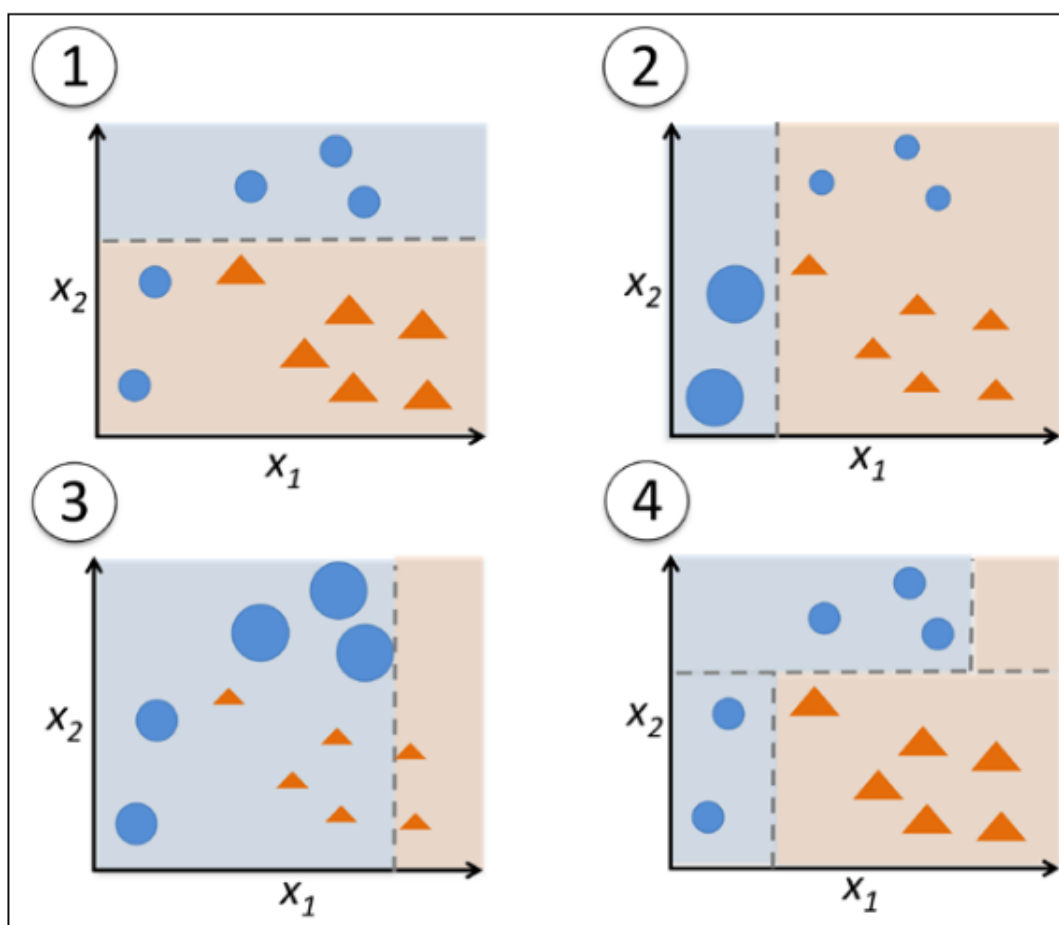
Para instalar paquetes en Python utilizaremos su gestor de paquetes *pip*.

```
pip install nombre_paquete
```

En nuestro caso vamos a instalar los paquetes básicos para probar el flujo de trabajo en machine learning. Más adelante profundizaremos en la utilización de los algoritmos de boosting.

4.5. Algoritmos de Boosting

El boosting es un tipo particular de ensemble, consiste en un conglomerado de clasificadores débiles (que son solo un poco mejor que la asignación aleatoria) que actúan de forma conjunta para construir un clasificador fuerte (de gran calidad). Un ejemplo típico de clasificado débil es un árbol de una sola hoja. En boosting se va construyendo el clasificador de forma iterativa, como ya mencionamos en la sección matemática. Las instancias que no han sido bien clasificadas en la iteración anterior adquieren una mayor importancia para la siguiente iteración. Podríamos decir que el algoritmo de boosting va concentrándose en acertar las instancias más difíciles conforme avanza en su proceso iterativo.



4.5.1. Adaboost

4.5.2. Extreme Gradient Boosting - XGBoost

4.6. Ruido de clase, Regularización, Underfing y
 Overfitting

4.7. Regularización en XGBoost

Capítulo 5

Conclusiones

