REDES NEURONALES

APRENDIZAJE DE MAQUINA I - CEIA - FIUBA

Dr. Ing. Facundo Adrián Lucianna

REPASO CLASE ANTERIOR

- Metodos de ajustes de hiperparámetros
- Maximal Margin Classifier
- Clasificador de vector de soportes
- Support Vector Machines para clasificar
- Support Vector Machines para regresión

MÉTODOS DE AJUSTE DE LOS HIPERPARÁMETROS

Dos métodos de búsqueda típicos:

Búsqueda de grilla: Busca exhaustiva de toda las combinaciones de los parámetros. Es el mas completo pero el mas ineficiente.

Búsqueda aleatoria: Busca aleatoriamente tomando datos del espacio de combinaciones de los parámetros, bajo una distribución aleatoria dada. Termina dado una cierta cantidad arbitraria de iteraciones.

MAXIMAL MARGIN CLASSIFIER

Si nuestra data es linealmente separable, puede existir un numero infinito de hiperplanos que van a funcionar

Por lo que necesitamos algún criterio de selección.

El caso que aquí estamos viendo busca el hiperplano que más lejos se encuentra de los datos de entrenamiento.

Es decir, computamos la distancia mínima de cada observación de entrenamiento y obtenemos la distancia más chica de las distancias, que llamamos **margen**.

El objetivo es buscar el hiperplano que mas grande posee este margen. Y el algoritmo que hace esto es el **Maximal Margin Classifier**

Podemos pensar que el clasificador busca el máximo **grosor** de recta que puede pasar entre las clases

CLASIFICADOR DE VECTOR DE SOPORTES

Por lo que vimos, si queremos seguir usando un hiperplano, debemos relajar las exigencias:

- Mayor robustez a observaciones individuales.
- Mejor clasificación de la **mayoría** (no todas) de las observaciones de entrenamiento.

Es decir, podría valer la pena clasificar erróneamente algunas observaciones de entrenamiento para poder clasificar mejor las observaciones restantes.

CLASIFICADOR DE VECTOR DE SOPORTES

Similar al clasificador, hay un pequeño numero de observaciones en el margen o dentro de él, que son los que determinan el hiperplano, esto se llaman **vectores de soporte**.

Las demás observaciones no tienen importancia para el modelo.

SUPPORT VECTOR MACHINE

El modelo llamado **Support Vector Machine (SVM)** o maquina de vector de soportes extiende al Clasificador de vector de soportes permitiendo extender el espacio de features, usando **funciones kernels**.

La frontera de decisión la podemos describir como:

$$f(X) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i K(X, X_i)$$

La forma de entrenamiento, la importancia de los vectores de soporte y el hiperparámetro C se mantienen. La gran diferencia es que ahora las fronteras de decisión no son necesariamente lineales y determinada por la función **kernel elegida**.

SUPPORT VECTOR MACHINE

Kernel polinomial de orden 3

Kernel radial

SUPPORT VECTOR MACHINE

SUPPORT VECTOR MACHINE EN REGRESIÓN

Lo que se optimiza ahora es el hiperplano que mejor que logra meter a todos los puntos de entrenamiento dentro del margen, minimizando el valor del margen.

SUPPORT VECTOR MACHINE EN REGRESIÓN

$$\sum_{i=1}^{n} (\epsilon_i + \epsilon_i^*) \leq C \qquad \epsilon_i, \epsilon_i^* \geq 0, \quad \forall i = 1, \ldots, n$$

La restricción es:

$$\begin{aligned} y_i - (< \widehat{\beta}, X > + b) &\leq m + \epsilon_i \\ (< \widehat{\beta}, X > + b) - y_i &\leq m + \epsilon_i^* \end{aligned} \qquad \forall i = 1, \ldots, n$$

REDES NEURONALES

REDES NEURONALES

Las redes neuronales originalmente se plantean como algoritmos matemáticos que tratan de imitar los cálculos complejos que tienen lugar en el cerebro.

Se busca imitar no solo la gran cantidad de unidades de procesamiento (neuronas) sino la interconexión entre ellas (sinapsis).

Esto genero dos grandes campos de aplicación, uno el de la neurociencia computacional y por otro el de **Deep Learning**. Siendo este ultimo el gran popular de hoy en día con avances que vemos constantemente en el publico general.

REDES NEURONALES

Un poquito de historia...

Las redes neuronales podemos dar comienzo con la neurona de **McCulloch-Pitts** en 1943. Este modelo es la primera formulación del proceso de cálculo que lleva a cabo una neurona, basado en una formulación algebraica. La unidad actual usada de **Deep Learning** no es muy diferente a este modelo.

En 1952, Hodgkin y Huxley crean el modelo basado en conductancia de la neurona, que modela como los potenciales de acción de las neuronas se generan y se propagan. Es un modelo de ecuaciones diferenciales. Los resultados fueron tan importantes que se llevaron el premio Nobel de medicina en 1963.

Este modelo que describen como funciona la neurona siguieron camino al desarrollo de la **neurociencia computacional**.

REDES NEURONALES

Lo importante es que en este estadio embrionario del área, es que las investigaciones estaban en su cúspide.

En 1958, Rosenblatt realizó la primera implementación del perceptrón (basado en la neurona de McCulloch-Pitts).

Rosenblatt es el padre del deep learning.

Es quien perfeccionó el perceptron moderno, y las redes

de dos o tres capas.

REDES NEURONALES

Pero en 1969 llego el **primer invierno** en redes neuronales. En 1969 se publicó un libro llamado Perceptrons de Minsky y Papert que enfatizaba los limites de lo que los perceptones podían hacer.

Este libro y su popularidad mató toda financiación en investigaciones hasta 1980.

Entre muchas criticas, se planteó que el **perceptrón** como modelo neuronal **no** podía resolver la función lógica XOR, algo que una neurona biológica si podría...

REDES NEURONALES

Pero en 1969 llego el **primer invierno** en redes neuronales. En 1969 se publicó un libro llamado Perceptrons de Minsky y Papert que enfatizaba los limites de lo que los perceptones podían hacer.

Este libro y su popularidad mató toda financiación en investigaciones hasta 1980.

Entre muchas criticas, se planteó que el **perceptrón** como modelo neuronal **no** podía resolver la función lógica XOR, algo que una neurona biológica si podría...

Luego en experimentos fisiológicos se probo que la neurona biológica tampoco puede resolver la función lógica XOR, sino que necesita de una red.

REDES NEURONALES

Pero en 1969 llego el primer invierno en redes neuronales. En 1969 se publicó un libro llamado Perceptrons de Minsky y Papert que enfatizaba los limites de lo que los perceptones podían hacer.

Este libro y su popularidad mató toda financiación en investigaciones hasta 1980.

Entre muchas criticas, se planteó que el **perceptrón** como modelo neuronal **no** podía resolver la función lógica XOR, algo que una neurona biológica si podría...

Luego en experimentos fisiológicos se probo que la neurona biológica tampoco puede resolver la función lógica XOR, sino que necesita de una red.

The plot thickens... En 2020 se encontró que neuronas del cerebro humano si pueden.

REDES NEURONALES

El invierno termina en 1980-90 con los desarrollos de **Rumelhart**, **Hopfield**, entre otros. Aquí se desarrollaron el algoritmo de Back-propagation, redes de memoria y el concepto de propiedades emergentes.

Agregar mi implementación

Obtenido de https://github.com/TarinZ/hopfield-nets

REDES NEURONALES

El invierno termina en 1980-90 con los desarrollos de **Rumelhart**, **Hopfield**, entre otros. Aquí se desarrollaron el algoritmo de Back-propagation, redes de memoria y el concepto de propiedades emergentes.

Luego llego **un nuevo invierno** de redes neuronales con la llegada de algoritmos muchos mas sencillos de usar y sin tanta dificultad de retoques de híper parámetros, tales como SVM y los bosques aleatorios. Estos algoritmos eran más fácil de usar y rendían mucho mejor que las redes neuronales.

REDES NEURONALES

En 2010, volvió a la vida, y ahora, oficialmente se le empezó a llamar **Deep Learning**. Ahora se incorporaron nuevas arquitecturas de redes y características adicionales. Además ahora estos algoritmos empezaron a superar a los demás en tareas de clasificación de imágenes y videos, y el modelado de voz y texto.

Parte del éxito es la posibilidad de contar con datasets mas grandes (Big data) y mejores procesamiento de calculo y la llegada de Google, Meta, Microsoft.

PERCEPTRON Y NEURONAS SIGMOIDEAS

PERCEPTRÓN

Empecemos con una neurona biologica:

Tenemos:

Cuerpo celular o soma: Donde está el núcleo y las organelas de la célula.

Dendritas: Ramificaciones del soma, es donde la neurona recibe las sinapsis de otras neuronas.

Axón: Es la prolongación encargada de transmitir el impulso nervioso. Es como se comunica la neurona.

PERCEPTRÓN

En estado de reposo, tenemos...

PERCEPTRÓN

Si otra neurona excita, pero poco, aumenta le voltaje pero rápidamente vuelve a su estado.

PERCEPTRÓN

De similar forma, una sinapsis puede se inhibitoria

PERCEPTRÓN

Ahora, si la excitación supera el umbral, se genera un impulso dado un efecto **todo o nada**.

PERCEPTRÓN

Y este impulso se propaga del cuerpo hasta el terminal axónico, el cual la neurona da su salida.

PERCEPTRÓN

Niveles de excitación nos dan diferente tasas de disparo...

PERCEPTRÓN

Niveles de excitación nos dan diferente tasas de disparo...

PERCEPTRÓN

Y ante una misma excitación, hay sinapsis mas sensibles que otras

PERCEPTRÓN

Con esto en mente, armemos el modelo de neurona

$$y = f\left(\sum_{i=1}^{3} x_i w_i + b\right)$$

$$x_1 w_1 + x_2 w_2 + x_3 w_3 + b$$

$$\sum_{i=1}^{3} x_i w_i + b$$

PERCEPTRÓN

Con esto en mente, armemos el modelo de neurona

$$y = f\left(\sum_{i=1}^{3} x_i w_i + b\right)$$

$$x_1 w_1 + x_2 w_2 + x_3 w_3 + b$$

$$\sum_{i=1}^{3} x_i w_i + b$$

PERCEPTRÓN

Si tenemos una observación de **n** atributos, la neurona tendrá **n** entradas con **n+1** pesos sinápticos.

Por lo que la parte lineal es:

$$\sum_{i=1}^{n} x_i w_i + b$$

Donde w_i, b son los pesos sinápticos, que representan si la conexión es excitatoria ($w_i > 0$) o inhibitoria ($w_i < 0$), o que no haya conexión sináptica ($w_i = 0$)

PERCEPTRÓN

Veamos la función de activación. Qué estamos modelando con ella?

En nuestro modelo, no nos interesa modela el impulso nervioso, ni siquiera cuando ocurre, sino modelar la tasa de impulsos.

Hagamos un experimento: Pinchemos una neurona e inyectemos voltaje, y midamos la tasa de impulsos nervioso versus el voltaje. Al principio, antes que llegue al valor umbral de disparo, la neurona no dispara y por lo tanto la tasa es 0 Hz.

PERCEPTRÓN

Veamos la función de activación. Qué estamos modelando con ella?

En nuestro modelo, no nos interesa modela el impulso nervioso, ni siquiera cuando ocurre, sino modelar la tasa de impulsos.

Hagamos un experimento: Pinchemos una neurona e inyectemos voltaje, y midamos la tasa de impulsos nervioso versus el voltaje. Cuando aumentamos y pasamos el umbral, la neurona empieza a disparar.

PERCEPTRÓN

Veamos la función de activación. Qué estamos modelando con ella?

En nuestro modelo, no nos interesa modela el impulso nervioso, ni siquiera cuando ocurre, sino modelar la tasa de impulsos.

Hagamos un experimento: Pinchemos una neurona e inyectemos voltaje, y midamos la tasa de impulsos nervioso versus el voltaje. Durante la etapa lineal, si duplicamos el voltaje, la neurona dispara al doble de la frecuencia.

PERCEPTRÓN

Veamos la función de activación. Qué estamos modelando con ella?

En nuestro modelo, no nos interesa modela el impulso nervioso, ni siquiera cuando ocurre, sino modelar la tasa de impulsos.

Hagamos un experimento: Pinchemos una neurona e inyectemos voltaje, y midamos la tasa de impulsos nervioso versus el voltaje. Pero llega una etapa que el periodo refractario deja de que aumente la tasa de disparo y se empiece a saturar

PERCEPTRÓN

Veamos la función de activación. Qué estamos modelando con ella?

En nuestro modelo, no nos interesa modela el impulso nervioso, ni siquiera cuando ocurre, sino modelar la tasa de impulsos.

Hagamos un experimento: Pinchemos una neurona e inyectemos voltaje, y midamos la tasa de impulsos nervioso versus el voltaje. Cuando llegamos a un punto, no importa cuanto voltaje introduzcamos, al neurona dispara a la máxima frecuencia.

PERCEPTRÓN

Por lo que la función de activación es una función no lineal que modela la variación de la tasa de disparo de la neurona.

Y con esto último, tenemos la expresión mínima de una neurona, el cual es una **unidad de calculo no lineal**.

PERCEPTRÓN

La función de activación que se usa en redes neuronales es una decisión de diseño:

$$y = f\left(\sum_{i=1}^{n} x_i w_i + b\right)$$

Función escalón $$f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

Función escalón $$f(x) = \begin{cases} -1 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

Función lineal $$f(x) = x$$

PERCEPTRÓN

La función de activación que se usa en redes neuronales es una decisión de diseño:

$$y = f\left(\sum_{i=1}^{n} x_i w_i + b\right)$$

Funciones sigmoideas

$$f(x) = \frac{e^x}{1 + e^x}$$

$$f(x) = tanh(x)$$

Función ReLu

$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$

PERCEPTRÓN

Con un perceptrón o neurona o red de una sola capa, se toma un vector de variables:

$$X = (x_1, x_2, \ldots, x_n)$$

Donde, una neurona entrenada, puede predecir un label o variable a predecir,

$$y = f\left(\sum_{i=1}^{n} x_i w_i + b\right)$$

Si usamos la **función de activación lineal**... podemos ver una regresión lineal

Si usamos la **función sigmoidea**... podemos ver una regresión logística. Al usar funciones de activación con salida continua, tenemos una métrica no solo de que clase es, sino un valor probabilístico de que tan seguro es la predicción.

Como podemos ver una neurona puede hacer clasificaciones o regresiones. Una neurona en un espacio n-dimensional pone un hiperplano para separar clases o realizar una regresión.

PERCEPTRÓN

Veamos el ejemplo de funciones lógicas:

AND (2 entradas)

X_1	X_2	Y
0	0	0
0	1	0
1	0	0
1	1	1

PERCEPTRÓN

Veamos el ejemplo de funciones lógicas:

OR (2 entradas)

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	1

PERCEPTRÓN

Veamos el ejemplo de funciones lógicas:

XOR (2 entradas)

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

REDES FEED-FOWARD

REDES FEED-FOWARD

La forma de tener fronteras de decisión mas complejas que la lineales o regresiones mas complejas que la lineal, una forma de resolverlo es armando una red formadas por neuronas en capas, en donde cada capa se conecta con la siguiente pero nunca hay retroalimentación.

Ademas este tipo de redes, también llamadas redes densas, una neurona de una capa se conecta con todas las neuronas de la siguiente capa.

REDES FEED-FOWARD

REDES FEED-FOWARD

Este tipo de redes, la capa intermedia transforma el espacio de entrada en diferentes espacios generalmente no lineales.

Si todas las neuronas tienen función de activación lineal, la red colapsa a una sola neurona, por lo que una red feed-foward se justifica en casos no lineales.

REDES FEED-FOWARD

Cada neurona de la capa intermedia tiene la misma función de activación y su salida es:

$$A_k = f\left(\sum_{i=1}^{n} x_i w_{ki} + w_{k0}\right)$$

Esta transformación producidas por las capa intermedia, luego ingresan como entrada a la capa final:

$$y = g\left(\sum_{k=1}^{K} A_k \beta_k + \beta_0\right)$$

BREVE INTRODUCCIÓN DE ENTRENAMIENTO DE REDES

ENTRENAMIENTO

El entrenamiento de redes es algo complejo, y lo verán en más detalles en otra asignatura. Entrenar una red neuronal, es ajustar los pesos sinápticos para que los resultados coincidan con los valores a predecir del set de entrenamiento. Es decir, las redes neuronales feed-foward es de **aprendizaje supervisado**.

Para entrenar nuestro modelo, como vimos otros, es minimizar una función de perdida que sea acotada, de tal forma que podemos llegar, mediante alguna técnica de optimización al valor mínimo.

Para este caso, veamos en un caso de clasificación, el error cuadrático medio:

$$\underset{w_{kj},\beta_k}{\text{minimizar}} \; \frac{1}{2} \sum_{i=1}^{n} \left(y_i - f(X_i)\right)^2$$

ENTRENAMIENTO

Para ahora vamos a basarnos en esta red:

La función de activación **g** de la capa oculta es genérica, pero la de salida es lineal.

ENTRENAMIENTO

$$\underset{w_{kj},\beta_k}{\text{minimizar}} \; \frac{1}{2}\sum_{i=1}^{n}\left(y_i - f(X_i)\right)^2$$

Donde: $$f(X_i) = \beta_0 + \sum_{k=1}^{K}\beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj}x_{ij}\right)$$

Esta función de perdida, que tiene tantas dimensiones como pesos sinápticos tiene, no es convexa y, por lo tanto, existen múltiples soluciones. Veamos a un ejemplo a que nos referimos a esto de un caso de optimización con un solo parámetro:

ENTRENAMIENTO

Para lograr evitar caer en mínimos locales y también de sobreajustes lo entrenamos lentamente usando un algoritmo de gradiente descendiente y el proceso se corta cuando se detecta sobreajuste usando set de validación.

Para ver como funciona gradiente descendiente, supongamos que ponemos a todos los pesos sinápticos como un gran vector θ, por lo que podemos reescribir a nuestra objetivo de minimización como:

$$R(\theta) = \frac{1}{2} \sum_{i=1}^{n} \left(y_i - f_\theta(X_i)\right)^2$$

ENTRENAMIENTO

La idea de gradiente descendiente es:

1. Comenzar con un valor de θ al azar θ^0 en t=0
2. Iterar hasta que $R(\theta) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f_\theta(X_i))^2$ deja de crecer o llega a un valor objetivo:
    1. Buscar un vector δ que refleje un pequeño cambio en θ, de tal forma que $\theta^{t+1}=\theta^t+\delta$ reduzca a R(θ).
    2. Cambiar t a t+1, e iterar.

BACKPROPAGATION

Bien, tenemos la idea, ahora como sabemos en que dirección mover a θ para decrecer a R(θ)?

Eso lo podemos ver usando el gradiente de R(θ), evaluada en un punto θ^m, que es el vector de las derivadas parciales:

$$\bigtriangledown R(\theta^m) = \frac{\delta R(\theta)}{\delta \theta}|_{\theta=\theta^m}$$

El gradiente nos da la dirección en el espacio θ en donde R(θ) incrementa más cuando estamos parado en θ^m. La idea en gradiente descendiente es movernos un pasito en la dirección contraria:

$$\theta^{m+1} = \theta^m - \rho \bigtriangledown R(\theta^m)$$

BACKPROPAGATION

ρ es la constante de aprendizaje, que es un valor que elegimos para determinar que tan rápido queremos que sea el entrenamiento.

Obtengamos el gradiente para nuestra red, volvamos a R:

$$R(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - f_\theta(X_i)\right)^2$$

Esto es una suma de n observaciones, por lo tanto el gradiente es una suma de n. Cada término de la suma de R la podemos escribir como:

$$R_i(\theta) = \frac{1}{2}\left(y_i - \beta_0 - \sum_{k=1}^{K}\beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj}x_{ij}\right)\right)^2$$

BACKPROPAGATION

ρ es la constante de aprendizaje, que es un valor que elegimos para determinar que tan rápido queremos que sea el entrenamiento.

Obtengamos el gradiente para nuestra red, volvamos a R:

$$R(\theta) = \frac{1}{2}\sum_{i=1}^{n}\left(y_i - f_\theta(X_i)\right)^2$$

Esto es una suma de n observaciones, por lo tanto el gradiente es una suma de n. Cada término de la suma de R la podemos escribir como:

$$R_i(\theta) = \frac{1}{2}\left(y_i - \beta_0 - \sum_{k=1}^{K}\beta_k g\left(\underline{w_{k0} + \sum_{j=1}^{p} w_{kj}x_{ij}}\right)\right)^2$$

$$\boxed{z_{ik} = w_{k0} + \sum_{j=1}^{p} w_{kj}x_{ij}}$$

Para simplificar creamos la siguiente variable intermedia

BACKPROPAGATION

Realizamos la derivada parcial de cada peso sináptico, empezamos por los pesos de la ultima capa:

$$\frac{\partial R_i(\theta)}{\partial \beta_k} = \frac{\partial R_i(\theta)}{\partial f_\theta(x_i)} \frac{\partial f_\theta(x_i)}{\partial \beta_k} = -(y_i - f_\theta(x_i))g(z_{ik})$$

Y los pesos de la capa oculta:

$$\begin{aligned}\frac{\partial R_i(\theta)}{\partial w_{kj}} &= \frac{\partial R_i(\theta)}{\partial f_\theta(x_i)} \frac{\partial f_\theta(x_i)}{\partial g(z_{ik})} \frac{\partial g(z_{ik})}{\partial z_{ik}} \frac{\partial z_{ik})}{\partial w_{kj}} \\ &= -(y_i - f_\theta(x_i))\beta_k g'(z_{ik})x_{ij}\end{aligned}$$

Notamos que el residuo aparece en todas las derivadas. Esta traslado del residuo desde la salida para encontrar los valores es lo que llamamos **backpropagation**. Ya que los pesos de la red se modifican paso a paso trasladando desde la ultima capa hacia atrás.

BACKPROPAGATION

Entonces el algoritmo de gradiente descendiente mediante backpropagation lo podemos simplificar como:

1. Comenzar con un valor de θ al azar θ^0
2. Bajaramos al azar las observaciones del set de entrenamiento. Elegimos la primera observación.
    1. Calculamos el valor del gradiente con las formulas que vimos y la observación elegida.
    2. Modificamos los pesos haciendo $\theta^{m+1} = \theta^m - \rho \bigtriangledown R(\theta^m)$
    3. Si llegamos a un valor umbral terminamos.
    4. Si no, elegimos la siguiente observación y vamos al paso 1 de este subitems. Si es la ultima observación, volvemos a barajar el set de observaciónes y comenzamos de nuevo.

BACKPROPAGATION

Este algoritmo tiene un problema el cual es muy lento para set de entrenamientos muy largos, ya que debe pasar por todos los valores y evaluar a R cada vez.

Una forma que podemos acelerar esto es submuestrando en batch (minibatch) de m observaciones,

Para cada batch, calcular los gradientes de las m observaciones y los sumamos. Luego evaluamos R.

De esta forma se acelera el proceso porque no estamos calculando R cada paso, y esto acelera el proceso, pero aumentamos la probabilidad de no llegar a un resultado más optimo.

PYTORCH

PYTORCH

PyTorch es una librería de Deep Learning. Es mantenida por Meta y es la principal competencia de otra famosa librería (tensorflow de Google).

Tiene un API para Python, como también para C++. PyTorch nos da toda una infraestructura de:

- Computación de tensores. Es similar a los array de Numpy pero con posibilidad de usarlos en GPU.
- Redes neuronales profundas.

Aquí vamos a usar lo básico, esta librería lo verán en mas profundidad en otras asignaturas.

VAMOS A PRÁCTICAR UN POCO...