UNIVERSIDAD NACIONAL
DEL NORDESTE

Módulo 3:

Aprendiendo y prediciendo con modelos lineales

Planteo: Aprendizaje supervisado

Dado un conjunto de $N_D \gg 1$ pares de datos $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N_D}$.

- x_n los llamamos entradas o “inputs”
- y_n son los “targets”

El objetivo del machine learning es determinar una función o mapa entre el espacio donde yacen los x_n y el de los y_n:

$$y = f(x)$$

lo que quiero encontrar es la f.
El modelo permite “generalizar” o “predecir” mas allá de los datos, es decir si tengo un nuevo x_j y he determinado la f, puedo encontrar el valor que le corresponde: $y_j = f(x_j)$.
Esto es una predicción del modelo ya que no tenemos ningun dato en x_j.

Objetivos

- Regresión
- Clasificación
- Maximo-verosimilitud y cuadrados mínimos
- Capacidad del modelo, over y underfitting
- Sesgo vs varianza en el modelo
- Regularización y regresión Bayesiana

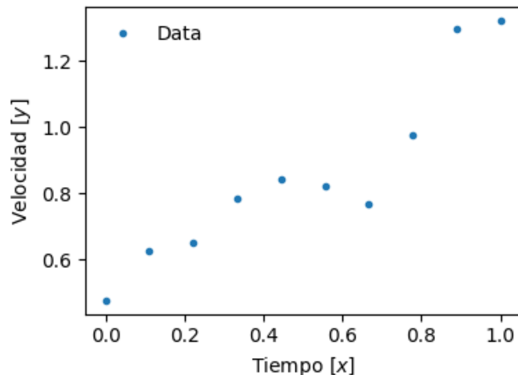
Bibliografía

- Bishop, C.M. 1996. Pattern recognition and machine learning. Springer.

Con perfil mas probabilista:

- Murphy, K.P., 2012. Machine learning: a probabilistic perspective. MIT press.

Ejemplo 1: Determinar la velocidad de una partícula/pez para un dado tiempo



$x \in \mathbb{R}$ y $y \in \mathbb{R}$ entonces queremos encontrar una $f : \mathbb{R} \to \mathbb{R}$
Este es denominado un problema de regresión.
Si la función f se asume apriori es una línea o una recta, $y = a_1 x + a_0$ se denomina regresión lineal.

También se suele llamar método de cuadrados mínimos.
Como asumimos que es una línea recta, debemos determinar los coeficientes a_1 y a_0 que mejor ajustan a los puntos/datos.

Ejemplo 2: Determinar la velocidad de un pez dada la posición

Supongamos que ponemos un GPS a un pez y medimos la posición y la velocidad. Asumiendo la corrientes fijas y el pez nada siempre a una velocidad fija queremos determinar:
$v = |\mathbf{v}| = f(\mathbf{x})$ donde $\mathbf{x} \in \mathbb{R}^3$ y $v \in \mathbb{R}$.

$$y = \mathbf{a}_1 \cdot \mathbf{x} + a_0 = a_{13}x_3 + a_{12}x_2 + a_{11}x_1 + a_0$$

donde $\mathbf{a}_1$ es una vector de dimensión $N_x = 3$ y a_0 es el bias/sesgo.
Ahora tendríamos que determinar 4 coeficientes, regresión lineal múltiple.

Ejemplo 3: Determinar la velocidad vectorial 3d de un pez dada la posición

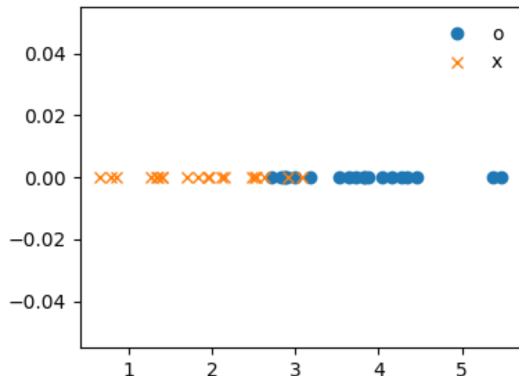
$\mathbf{v} = f(\mathbf{x})$ donde $\mathbf{x} \in \mathbb{R}^3$ y $\mathbf{v} \in \mathbb{R}^3$.
En general si tenemos que: $\mathbf{x} \in \mathbb{R}^{N_x}$ y $\mathbf{y} \in \mathbb{R}^{N_y}$ y asumimos una función lineal a priori, entonces tenemos que determinar:

$$\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}$$

donde $\mathbf{A}$ es una matriz de dimensión $N_y \times N_x$ y $\mathbf{b}$ el bias/sesgo es un vector de N_y dimensiones.

Este problema lo podemos subdividir en N_y regresiones lineales múltiples (excepto que los errores de las observaciones esten correlacionados).

Ejemplo 4: Determinar el tipo de pez en base a sus velocidades



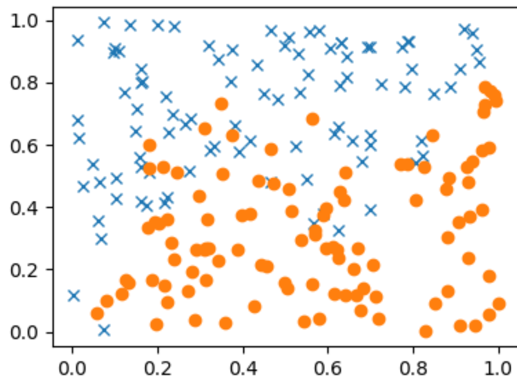
$x \in \mathbb{R}$ y $y \in \{o, x\}$ entonces queremos encontrar una $f : \mathbb{R} \to \{o, x\}$ Los dos tipos o “labels” se los denomina con los numeros binarios $\{0, 1\}$.

Este es denominado un problema de clasificación binaria.

Debemos encontrar donde dividir el dominio, x_D, tal que $x > x_D$ $f(x) = 'o'$. Mientras si $x \leq x_D$ $f(x) = 'x'$.

¿Cual les parece que tiene que ser el valor de x_D? Son los datos exactos?

Ejemplo 5: Determinar el tipo de pez en base a sus velocidades y profundidad



$x \in \mathbb{R}^2$ y $y \in \{o, x\}$ entonces queremos encontrar una $f : \mathbb{R}^2 \rightarrow \{o, x\}$

Este es denominado un problema de clasificación 2d.

Debemos encontrar donde dividir el dominio, $y(x)$, tal que $y_n > y(x_n)$ $f(x_n) =' \mathrm{o}'$. Mientras si $y_n \leq y(x_n)$ $f(x_n) =' \mathrm{x}'$.

¿Cual les parece que deba ser la función $y(x)$ que divide? Es una recta? Son los datos exactos? Otros ejemplos: pacientes con/sin cancer a partir de una imagen de diagnostico? dado un texto de review de un restaurante clasificar si este fue positivo o negativo.

Función objetivo, de pérdida, de costo

Supongamos que tenemos un conjunto de datos $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N_D}$.
La predicción realizada por el modelo es

$$y^f = f(\mathbf{x}, \mathbf{a}) + \epsilon$$

La función de pérdida la definimos con alguna métrica que mida las diferencias entre los datos (targets) y la prediccións del modelo.
La Mean Square Error (MSE), distancia euclidea, viene dada por

$$J(\mathbf{a}) = \frac{1}{2N_D} \sum_{n=1}^{N_D} (y_n - f(\mathbf{x}_n, \mathbf{a}))^2$$

La división por el número de datos (normalización) es conveniente para cuando se quiere comparar J con distintos conjuntos de datos.
Buscamos el conjunto de parámetros $\mathbf{a}$ cuya “función” pase lo mas cerca posible de los datos, produzca el menor error posible de predicción.

Regresión lineal

Proponemos: $y^f = f(\mathbf{x}) + \epsilon = \mathbf{a}^\top \mathbf{x} + b + \epsilon$

Queremos determinar el $\mathbf{a}$ y b que minimizan la función de pérdida:

$$J(\mathbf{a}, b) = \frac{1}{2N_D} \sum_{n=1}^{N_D} \left[y_n - (\mathbf{a}^\top \mathbf{x}_n + b) \right]^2$$

Podemos determinar los pesos óptimos, que minimizan J, en forma analítica (Solución de cuadrados mínimos) :

$$0 = \nabla_{\mathbf{a}} J = \frac{1}{N_D} \left[-\mathbf{a}^\top \sum_{n=1}^{N_D} \mathbf{x}_n \mathbf{x}_n^\top + \sum_{n=1}^{N_D} (y_n - b) \mathbf{x}_n^\top \right]$$

$$0 = \partial_b J = -\frac{1}{N_D} \sum_{n=1}^{N_D} \left[y_n - (\mathbf{a}^\top \mathbf{x}_n + b) \right]$$

$$b^* = \sum y_n - \mathbf{a}^\top \sum \mathbf{x}_n$$

$$\mathbf{a}^* = \left(\sum \mathbf{x}_n \mathbf{x}_n^\top - \sum \mathbf{x}_n \sum \mathbf{x}_n^\top \right)^{-1} \left(\sum y_n \mathbf{x}_n - \sum y_n \sum \mathbf{x}_n \right)$$

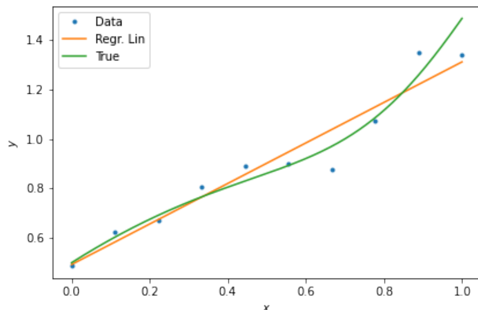
Mucho mas sencillo

Defino la design matrix

$\mathbf{X} = [\mathbf{1}, \mathbf{x}_1, \ldots, \mathbf{x}_{N_x}] \in [N_D, N_x + 1]$

Las columnas de $\mathbf{X}$ son las características. Las filas los datos.

$$J(\mathbf{a}) = \frac{1}{2N_D} (\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w})$$



$$0 = \nabla_w J(\mathbf{w}) = \frac{1}{N_D} \mathbf{X}^\top (\mathbf{y} - \mathbf{X}\mathbf{w})$$

Los pesos óptimos $\mathbf{w}^*$ vienen dados por:

$$\mathbf{X}^\top \mathbf{X}\mathbf{w} = \mathbf{X}^\top \mathbf{y} \qquad \rightarrow \qquad \mathbf{w}^* = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}$$

Inversión de matriz. $\mathbf{X}^\top \mathbf{X}$ Puede ser singular. Se puede hacer con SVD.

Extensión a la regresión nolineal. Modelo lineal

Cualquier superposición de un conjunto ortogonal de funciones es lineal con respecto a los parámetros (independientemente de las dependencias en las inputs):

$$y^f = \sum_{j=1}^{N_c} w_j \phi_j(\mathbf{x})$$

donde $\{\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \cdots, \phi_{N_C}(\mathbf{x})\}$ es un conjunto ortogonal de funciones. N_c es la capacidad o la complejidad del modelo (número de funciones base).

Si hay múltiples outputs las puedo trabajar por separado. A menos que ...

Design matrix para modelos lineales generalizados

Puedo trabajar con la design matrix

$$\mathbf{X} = [\phi_k(\mathbf{x}_n)]_{nk}$$

Las columnas de $\mathbf{X}$ son las características/features $\mathbf{\Phi}_k$. Las filas los datos.

$$\mathbf{X} = \begin{pmatrix} \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \cdots & \phi_{N_C}(\mathbf{x}_1) \\ \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \cdots & \phi_{N_C}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\mathbf{x}_{N_D}) & \phi_2(\mathbf{x}_{N_D}) & \cdots & \phi_{N_C}(\mathbf{x}_{N_D}) \end{pmatrix}$$

Los pesos óptimos vienen (de nuevo) dados por:

$$\mathbf{w}^* = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}$$

Para regresión polinomial: $\mathbf{\Phi}(x) = [1, x, x^2, x^3, \cdots, x^{N_c-1}]$ $\mathbf{X} \in [N_D, N_c]$

¿Que sucede si el input es de 2 variables? y de 3 variables? ¿como aumenta el número de coeficientes?

Ejemplos de funciones base

- Funciones polinómicas: $\phi_j(\mathbf{x}) = x^j$. Puedo incluir el sesgo: $\phi_0(\mathbf{x}) = 1$
- Otros ejemplos: senos y cosenos, polinomios de Legendre, etc.$\rightarrow$ Problemas son funciones globales cambios en una region llevan a cambios en otras.
  Locales:
- Funciones base radiales $\phi_j(\mathbf{x}) = \exp\left[-\frac{(x-\mu_j)^2}{2\sigma^2}\right]$ donde μ_j define la ubicación y σ la escala de la función (ligadas a los datos). Se suele definir el parametro: $\alpha = \sigma^{-2}/2$ denominado bandwidth o precisión.
- Base de funciones con localización en el espacio y en los números de ondas: Ondeletas.

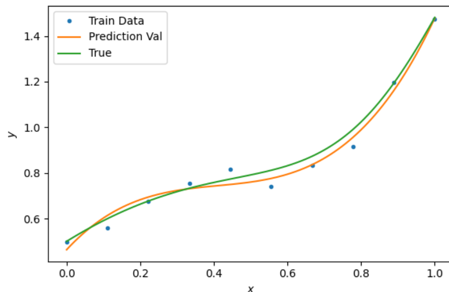
Definición de un modelo en pytorch para regresión polinomial

```
class poli_model(torch.nn.Module):
    def __init__(self,n_Degree,n_outputs):
    " Defino el modelo lineal. Grado del polinomio "
        super().__init__()
        self.n_inputs=n_Degree+1
        self.lm = torch.nn.Linear(n_inputs,n_outputs,bias=True)

    def forward(self, x):
        " Agrego como features los terminos del polinomio de grado n_inputs-1 "
        x_aug=[x[:,0]**i for i in range(1,self.n_inputs)]
        x_aug=torch.stack(x_aug)
        return self.lm(x_aug.T)
```

Ajuste polinómico

Dentro de la clase poli_model agrego dos funciones:

```
def design(self,x):
    x_aug=[x[:,0]**i for i in range(
                                self.n_inputs)]
    x_aug=torch.stack(x_aug)
    return x_aug.T

def optweights(self,x,y):
    with torch.no_grad(): # no training
        X=self.design(x)
        minv=torch.linalg.inv(X.T @ X)
        w= minv @ (X.T @ y)
        self.lm.bias[0]=w[0]
        self.lm.weight[0,:]=w[1:,0]
```



Polinomio optimo.

```
model=poli_model(3,1)
model.optweights(x_train,y_train)
with torch.no_grad(): # no training
    y_pred_val=model(x_val)
```

Entrenamiento = Optimización

Given a set of input vectors $\{\mathbf{x}_n\}_{n=1}^{N_D}$, and corresponding targets $\mathbf{y}$, the aim is to “learn” the relationship between inputs and targets given a network/model.

En general es imposible determinar los parámetros analíticamente.

“Training the network involves searching in the weight space of the network/model for a value of w that produces a function that fits the provided training data well” (MacKay, 2003).

$$J(\mathbf{w}) = \frac{1}{2N_D} \left(\mathbf{y} - \mathbf{X}\mathbf{w}\right)^\top \left(\mathbf{y} - \mathbf{X}\mathbf{w}\right)$$

donde $\mathbf{X} = [\phi_k(\mathbf{x}_n)]_{nk}$
El mínimo de $J(\mathbf{w})$ o la raíz de $\nabla_w J(\mathbf{w})$ nos dan el peso óptimo.
Search $\rightarrow$ Algoritmos iterativos.

Gradiente

Teorema de funciones multivaluada: Si J es diferenciable, el vector $\nabla J(\mathbf{w})$ apunta en la dirección de máximo crecimiento de J.

Demostración. Usamos derivada direccional + desigualdad de Cauchy-Schwarz:

$$\frac{\partial J}{\partial \mathbf{u}}(\mathbf{w}) = \nabla J(\mathbf{w}) \cdot \mathbf{u} \leq |\nabla J(\mathbf{w})|$$

pero con $\mathbf{u} = \nabla J(\mathbf{w})/|\nabla J(\mathbf{w})|$ se cumple la igualdad $\rightarrow$ es el supremo y esta en la dirección $\nabla J(\mathbf{w})$.

Debemos buscar el mínimo de la función a lo largo de $-\nabla_{\mathbf{w}} J\big|_{\hat{\mathbf{w}}}$ este vector apunta al valle de la función

Optimización por descenso de gradientes

Dados un conjunto de parámetros iniciales, $\mathbf{w}_{old}$ y la posibilidad de calcular el gradiente de la función de costo con respecto a los parámetros, se realiza los updates de los parámetros como:

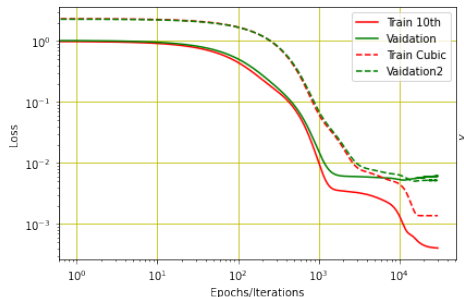
$$\mathbf{w}_{new} = \mathbf{w}_{old} - \eta \nabla J(\mathbf{w}_{old}, x_{data}, y_{data})$$

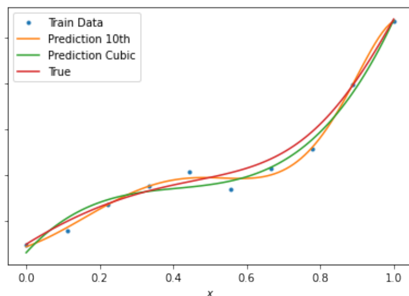
donde η es la learning rate.

```
for i in range ( n_epochs ):
    dJ_Dw = gradient ( loss_function , x_Data, y_Data , w )
    w = w - learning_rate * dJ_Dw
```

Notar que como parte del gradiente se requiere del gradiente de la función de predicción: $\nabla_w J = \nabla_f J \, \nabla_w f$

Ajuste polinómico por descenso de gradientes





Comparación de un modelo de polinomio cubico vs un modelo de polinomios de hasta 10mo orden.

¿Cual les parece que es mejor modelo?

Error de entrenamiento y de validación

El error de “entrenamiento” es el que determinamos en la función de pérdida con los datos.
Si determinamos un conjunto de parámetros “óptimos” durante el entrenamiento esperamos que el error de entrenamiento sea el menor posible de todos los conjuntos de parámetros. Esto NO garantiza que ese sea el menor error para cualquier otro conjunto de datos no usados en el entrenamiento

Durante la optimización es necesario probar cual es el error de la predicción con un conjunto de datos independiente del de entrenamiento.

Conjunto de datos de validación: Nos sirve para ver como generaliza el entrenamiento del modelo.
Cuantos son necesarios? En general conviene dividir 80% - 20%. El set de validación es tan chico como sea posible pero sin error de muestreo.
Se requiere que los datos de entrenamiento y validación sean IID (indep e ident distribuidos).

Generalización

Supongamos que tenemos dos estudiantes que estudian "Análisis matemático" de manera distinta, uno memoriza y el otro es inductivo.

¿Quien es el que se sacara mejor nota en el examen final?

- Si los ejercicios de los examenes son distintos a los que estudio/ que pasa?
- Cual es el que tiene mayor poder de generalización?

Capacidad

- ¿Que capacidad N_C del modelo tenemos que seleccionar?
- De que depende la capacidad que seleccionemos?
- Cuantas iteraciones de optimización tenemos que realizar? De que depende? Cuando nos detenemos?

¿Cuando estamos realizando un overfitting/sobreajuste de los datos?

Descomposición varianza-sesgo de la estimación

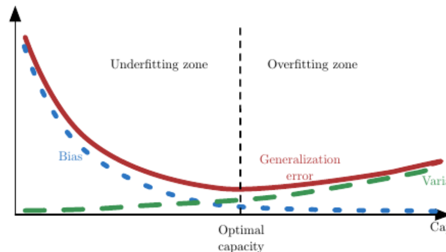
¿Como determinamos la complejidad del modelo?

Estimador que usa N_D IID data points: $w^*_{N_D} = g(\mathbf{x}_1, \cdots, \mathbf{x}_{N_D}$.

- Sesgo del estimador. Sesgo$(w^*_{N_D}) = \mathbb{E}(w^*_{N_D}) - w^{Tr}$
  Promedio de estimaciones (con el estimador) menos la verdad.
- Varianza del estimador. Var$(w^*_{N_D})$

La MSE de las estimaciones viene dada por

$$
\begin{aligned}
MSE &= \mathbb{E}[(w^*_{N_D} - w^{Tr})^2] \\
&= \mathbb{E}[(w^*_{N_D} - \mathbb{E}(w^*_{N_D}) + \mathbb{E}(w^*_{N_D}) - w^{Tr})^2] \\
&= (\mathbb{E}(w^*_{N_D}) - w^{Tr})^2 + \mathbb{E}[(w^*_{N_D} - \mathbb{E}(w^*_{N_D}))^2]
\end{aligned}
$$



$$MSE = [\text{Sesgo}(w^*_{N_D})]^2 + \text{Var}(w^*_{N_D})$$

Descomposición de la función de pérdida

Descompongamos la función de pérdida en dos términos

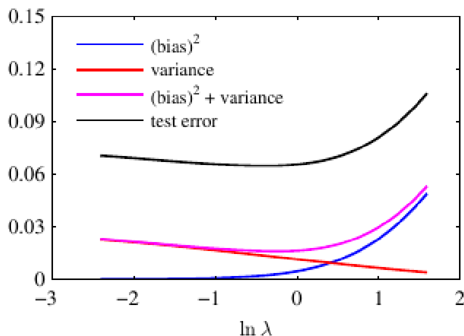
$$
\begin{aligned}
J &= \iint |f(\mathbf{x}) - y|^2 p(\mathbf{x}, y) \mathrm{d}\mathbf{x} \mathrm{d}y \\
&= \int [f(\mathbf{x}) - \mathbb{E}(y|\mathbf{x})]^2 p(\mathbf{x}) \mathrm{d}x + \int [\mathbb{E}(y|\mathbf{x}) - y]^2 p(\mathbf{x}, y) \mathrm{d}\mathbf{x} \mathrm{d}y
\end{aligned}
$$

$f_{Tr}(x) \doteq \mathbb{E}(y|\mathbf{x})$.

La función de ajuste $f_{\mathcal{D}} = f(\mathbf{x}; \mathcal{D})$ depende del conjunto de datos.

Usemos ensambles y descompongamos nuevamente:

$$
\begin{aligned}
J = &\int \mathbb{E}_{\mathcal{D}}[f_{\mathcal{D}} - f_{Tr}]^2 p(\mathbf{x}) \mathrm{d}\mathbf{x} \\
&+ \int \mathbb{E}_{\mathcal{D}}[f_{\mathcal{D}} - \bar{f}]^2 p(\mathbf{x}) \mathrm{d}\mathbf{x} \\
&+ \iint [f_{Tr} - y]^2 p(\mathbf{x}, y) \mathrm{d}\mathbf{x} \mathrm{d}y
\end{aligned}
$$



De Bishop

Regularización. Weight decay

Queremos castigar en la función de costo los valores altos de los parámetros (favoreciendo los pequeños)

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N_D}\left(y^{(n)} - f(\mathbf{x}^{(n)};\mathbf{w})\right)^2 + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}$$

ridge regression. Con esto “regularizamos” la función de pérdida (un solo parámetro óptimo).

- Concepto útil cuando trabajamos con pocos datos y modelos complejos (redes profundas).

Definición general de la función de pérdida

La función de pérdida general de Mikonski la definimos funcionalmente con

$$L_q(\mathbf{x}, y) = |f(\mathbf{x}) - y|^q$$

Con $q = 2$, norma L2, tenemos el MSE. $q = 1$ es la norma L1.

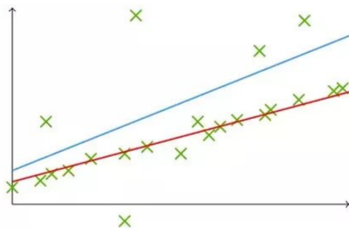
Obteniendo el valor esperado de la densidad conjunta,

$$J = \mathbb{E}(L_q) \doteq \iint |f(\mathbf{x}) - y|^q p(\mathbf{x}, y) \mathrm{d}\mathbf{x}\mathrm{d}y$$

Dados $\{\mathbf{x}^{(n)}, t^{(n)}\}_{n=1}^{N_D}$, la integral de Monte Carlo de J nos termina dando

$$J = \sum_{n=1}^{N_D} |f(\mathbf{x}^{(n)}) - y^{(n)}|^q$$

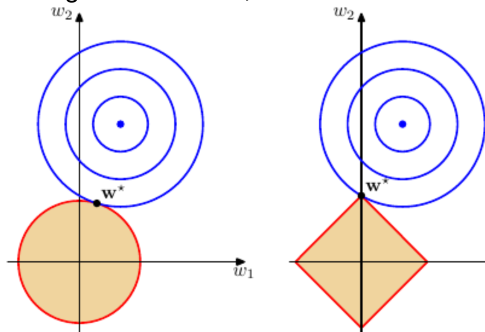
En forma comparativa $L2$ le da mayor peso a los outliers.



L_1 rojo. L_2 celeste.

Regularización lasso. Parámetros esparsos

La regularización L1, se conoce como lasso, produce soluciones esparsas.

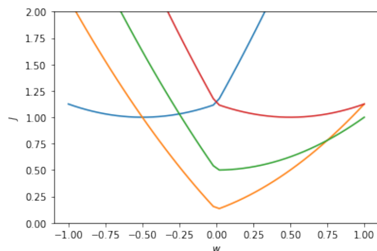


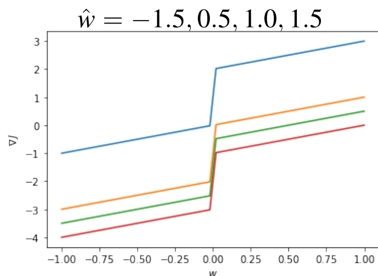
Contornos azules función de pérdida sin regularización.

Contornos rojos restricciones establecidas por la regularización.

De Bishop 1996.

$\hat{w} = -1.5, 0.5, 1.0, 1.5$





Interpretación estadística de la optimización

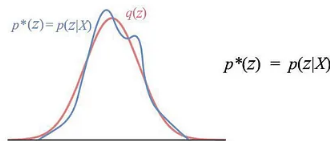
Si tenemos datos de un problema supervisado, $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^{N_D}$, estos son muestras de una distribución de probabilidad,

$$\text{Datos:} \qquad p_{\mathcal{D}}(\mathbf{y}|\mathbf{x})$$

Esta es desconocida y es el objetivo de machine learning determinarla. Supongamos nuestro modelo probabilístico de predicción se define por una familia de densidades parametrizadas a través de $\mathbf{w}$,

$$\text{Modelo:} \qquad p_f(\mathbf{y}|\mathbf{x}, \mathbf{w})$$

La idea es encontrar el conjunto de parámetros que acerque lo mas posible la densidad de predicción a la densidad de los datos,



Diferencias entre la distribución de los datos y la del modelo propuesto

La divergencia de Kullback-Leibler $\mathcal{D}_{KL}(p_{\mathcal{D}}\|p_f)$ nos “mide” la diferencias entre dos distribuciones:

$$\mathcal{D}_{KL}(p_{\mathcal{D}}\|p_f) \doteq \int p_{\mathcal{D}}(y|\mathbf{x}) \log\left(\frac{p_{\mathcal{D}}(y|\mathbf{x})}{p_f(y|\mathbf{x},\mathbf{w})}\right) \mathrm{d}\mathbf{x}$$

Aprovecho la asimetría en la definición de la $\mathcal{D}_{KL}$.

Usando los datos:

$$\mathcal{D}_{KL}(p_{\mathcal{D}}\|p_f|) \doteq \mathbb{E}_{p_{\mathcal{D}}}\left[\log\left(\frac{p_{\mathcal{D}}(y|\mathbf{x})}{p_f(y|\mathbf{x},\mathbf{w})}\right)\right] = -\sum_{n=1}^{N_D} \log p_f(y_n|\mathbf{x}_n,\mathbf{w}) + C$$

Estoy haciendo Monte Carlo con la integral.

Puedo interpretar a la distribución *en la* integral como:

$p_{\mathcal{D}}(y|\mathbf{x}) = \sum_n \delta(x - x_n)$

Máximo verosimilitud

Si los datos son asumidos iid la función verosimilitud, $\mathbf{w}$ y $\boldsymbol{\Sigma}$ determinan la media y la covarianza de la distribución

$$p(\mathbf{y}^{(1)}, \cdots, \mathbf{y}^{(N_D)} | \mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(N_D)}; \mathbf{w}, \boldsymbol{\Sigma}) = \prod_{n=1}^{N_D} p(\mathbf{y}^{(n)} | \mathbf{x}^{(n)}; \mathbf{w}, \boldsymbol{\Sigma})$$

El logaritmo de la verosimilitud viene dado por

$$l(\mathbf{w}, \boldsymbol{\Sigma}) = \log p(\mathbf{y}^{(1:N_D)} | \mathbf{x}^{(1:N_D)}; \mathbf{w}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N_D} \log p(\mathbf{y}^{(n)} | \mathbf{x}^{(n)}; \mathbf{w}, \boldsymbol{\Sigma})$$

Hemos obtenido el negativo de la divergencia de KL.

Maximizar la log-verosimilitud es equivalente a minimizar la DKL.

Máximo verosimilitud para distribuciones Gaussianas

Hipótesis Gaussiana: $p(y^{(n)}|\mathbf{x}^{(n)};\mathbf{w},\Sigma) = \mathcal{N}(y_n|f(\mathbf{x}_n,\mathbf{w}),\Sigma)$

$$l(\mathbf{w},\mathbf{\Sigma}) = C + \frac{N}{2}\log|\mathbf{\Sigma}| - \sum_{n=1}^{N_D}[\mathbf{y}_n - f(\mathbf{x}_n,\mathbf{w})]^\top \mathbf{\Sigma}^{-1}[\mathbf{y}_n - f(\mathbf{x}_n,\mathbf{w})]$$

Para $\mathbf{\Sigma} = \mathbf{I}$,

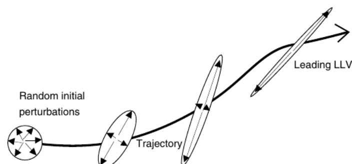
$$J(\mathbf{w}) = C - \sum_{n=1}^{N_D}[\mathbf{y}_n - f(\mathbf{x}_n,\mathbf{w})]^\top[\mathbf{y}_n - f(\mathbf{x}_n,\mathbf{w})]$$

Cuando minizamos la función de pérdida J:

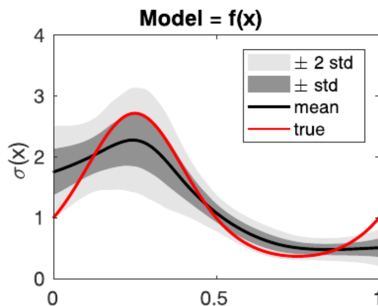
- Estamos maximizando la log-verosimilitud con respecto a los parámetros.
- Estamos buscando la densidad que mejor ajusta los datos (en una flia).

Rol de la covarianza

Ejemplo: Pronósticos meteorológicos



- Las situaciones de mal tiempo/tormenta son inestables y dificiles de pronosticar.
- Las situaciones de buen tiempo (despejado/centros de alta presión) son altamente probables.
- De acuerdo al estado del sistema tenemos predicciones que son mas precisas (con menor error de predicción) y para otros estados del sistema son menos precisas.



Ruido heterodástico

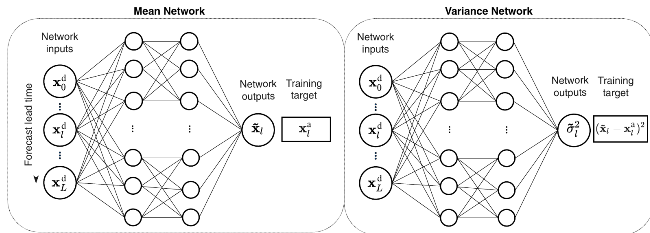
- Si el Σ depende del estado del sistema, debemos tenerlo en cuenta en la función de costo.
- En el caso del ejemplo, a mayores valores de las inputs el error de medición crece.
- De tal manera que las muestras que tienen mayor error pesen menos que la que tienen menor error $\rightarrow$ los datos con menor error son mas importantes.

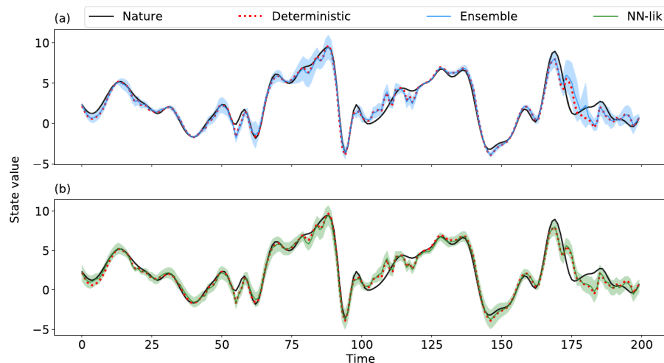
$$l(\mathbf{w}, \Sigma) = C + \sum_{n=1}^{N_D} \Sigma(\mathbf{x}^{(n)})^{-1} [y^n - f(\mathbf{x}^{(n)}, \mathbf{w})]^2$$

El problema es que muchas veces desconocemos a $\Sigma(\mathbf{x}^{(n)})$ pero se puede estimar con NNs.

Estimación de la media y la covarianza

Inferencia variacional Gaussiana (e.g. Sacco et al 2022)





Marco Bayesiano

Limitaciones de la estimación por verosimilitud: No nos dice nada sobre la incerteza de la predicción. Asume total desconocimiento a priori de los parámetros.
¿Que pasa si asumimos que tenemos un conocimiento apriori de los parametros expresado en la densidad de probabilidad a priori,

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})}$$

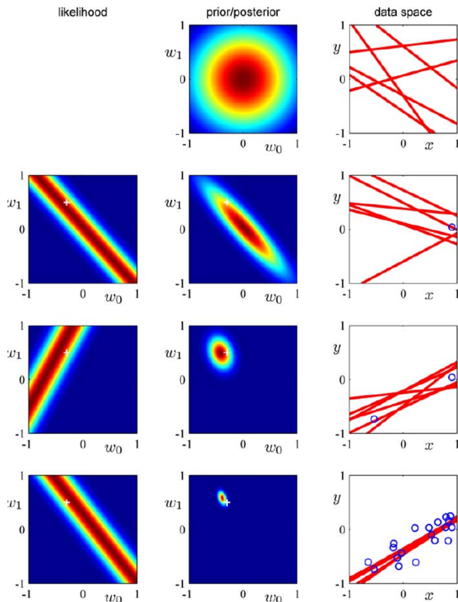
$$p(\mathcal{D}|\mathbf{w}) = \prod_{n=1}^{N_D} p(y_n|\mathbf{w}, \mathbf{x}_n)p(\mathbf{x}_n)$$

Asumiendo Gaussianidad: $p(y|\mathbf{w}, \mathbf{x}) = \mathcal{N}(y|f(\mathbf{x}; \mathbf{w}), \sigma^2)$

$$p(y|\mathbf{w}, \mathbf{x}) = (2\pi\sigma^2)^{-1/2} \exp\left(-\frac{1}{2\sigma^2}[y - f(\mathbf{x}; \mathbf{w})]^2\right)$$

Misma función verosimilitud.

Aprendizaje secuencial



Aprendizaje secuencial en un modelo Bayesiano lineal.

$$\log p(\mathbf{w}|\mathcal{D}) = -\frac{1}{2\sigma^2}\sum_{n=1}^{N_D}[y_n - f(\mathbf{x}_n;\mathbf{w})]^2 + \log p(\mathbf{w}) - N_D \log \sigma + C$$

Interpretación estadística de la regularización

La regularización L2 (ridge regression):

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N_D}\left(y^{(n)} - f(\mathbf{x}^{(n)};\mathbf{w})\right)^2 + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}$$

Renombrando constantes, puede reinterpretarse por el logaritmo de Gaussianas:

$$J(\mathbf{w}) = -\log\left\{\exp\left[-\sum_{n=1}^{N_D}\frac{\Sigma^{-1}}{2}\left(y^{(n)} - f(\mathbf{x}^{(n)};\mathbf{w})\right)^2\right]\exp\left[-\sum_{n=1}^{N_D}\frac{B^{-1}}{2}\mathbf{w}^\top\mathbf{w}\right]\right\} + Norm$$

Esto puede escribirse como productos de Gaussianas:

$$J(\mathbf{w}) = -\log\left(\prod_{n=1}^{N_D} p(y^{(n)}|\mathbf{x}^{(n)};\mathbf{w},\Sigma)p(\mathbf{w})\right) = -\log\left(\frac{p(\mathbf{w}|\mathbf{X},\mathbf{y})}{p(\mathbf{y})}\right)$$

Entonces la regularización puede ser interpretada como una densidad a-priori de los parametros asumiendo tienen media 0.

En lugar de MaxLik estamos haciendo MAP!

Clasificación

$$\phi : \mathbb{R}^{N_x} \to \{0, 1\}$$

Clasificación binaria

Como se vio en el ejemplo, tenemos dos clases de targets las cuales representamos por $\{0, 1\}$.
Entonces tenemos una variable de salida, la probabilidad de y si es 1 tenemos una probabilidad de 1 que sea 1. Si es 0 tenemos una probabilidad nula que sea 1.
Distribución de Bernoulli:

$$Pr(y|\lambda) = \begin{cases} 1 - \lambda & y = 0 \\ \lambda & y = 1 \end{cases}$$

donde λ es la probabilidad de que $y = 1$. En forma equivalente:

$$Pr(y|\lambda) = (1 - \lambda)^{1-y} \lambda^{y}$$

Recordamos que la media de la distribución de Bernoulli es $\mathbb{E}(y) = \lambda$.
Es decir que λ es un parámetro de la distribución (que la define totalmente).

Estimación por máximo verosimilitud

Supongamos que vamos a predecir la probabilidad de que $y = 1$ o equivalentemente la media de la distribución de Bernoulli con un modelo:

$$\lambda = f(\mathbf{x}, \mathbf{w})$$

A diferencia del problema de regresión aqui predecimos la probabilidad de la variable. (que es a su vez la media).

El logaritmo de la verosimilitud de todos los datos (dadas las inputs y los parametros) es

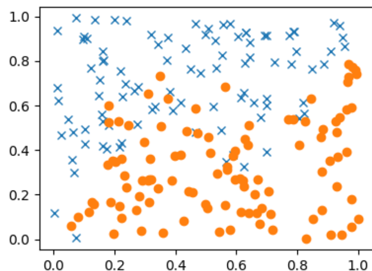
$$Pr(\mathbf{y}^{(1)}, \cdots, \mathbf{y}^{(N_D)} | \mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(N_D)}; \mathbf{w}) = \prod_{n=1}^{N_D} Pr(\mathbf{y}^{(n)} | \mathbf{x}^{(n)}; \mathbf{w})$$

$$\log Pr(\mathbf{Y}|\mathbf{X}; \mathbf{w}) = \sum_{n=1}^{N_D} (1 - y_n) \log[1 - f(\mathbf{x}_n, \mathbf{w})] + y_n \log[f(\mathbf{x}_n, \mathbf{w})]$$

Neg Log Lik o Cross entropy? No podemos usar MSE para clasificación.

Predicciones para un modelo de clasificación binaria

Que función $f(\mathbf{x},\mathbf{w})$ proponemos?



Lo mas sencillo seria dividir el plano por una linea, y asignar a todos los puntos de un lado con 1s y a los del otro lado con 0s:

$$y = \begin{cases} 0 & \text{si } y < \mathbf{w}_1 \cdot \mathbf{x} + w_0 \\ 1 & \text{si } y \geq \mathbf{w}_1 \cdot \mathbf{x} + w_0 \end{cases}$$

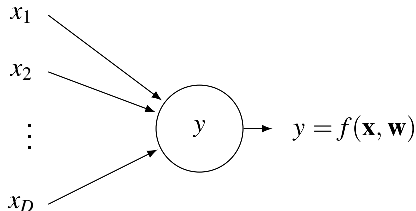
Usando la función de Heaviside Θ escribimos en forma compacta:

$$y = f(\mathbf{x},\mathbf{w}) = \Theta(\mathbf{w}_1 \cdot \mathbf{x} + w_0)$$

$$Pr(y|\lambda) = \begin{cases} 1-\lambda & y = 0 \\ \lambda & y = 1 \end{cases}$$

Los pesos son los que definen la región de decisión. Contornos entre clases.

Perceptrón



Identificada como la base de una neurona.

Θ es la función de activación.

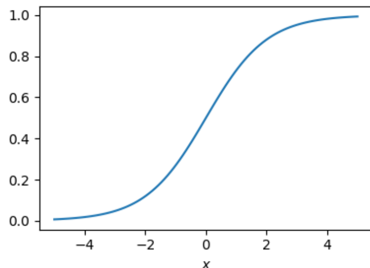
$$f(\mathbf{x}, \mathbf{w}) = \Theta(\mathbf{w}_1 \cdot \mathbf{x} + w_0)$$

- Propuesto por McCulloch and Pitts (1943).
- Implementado como una máquina por Rosenblatt (1958).
- Identificado (correctamente!) en aquel entonces como el embrión de la inteligencia artificial.

Recibieron una fuerte crítica en Perceptrons (Minsky and Papert, 1969). No son capaces de producir la función XOR.

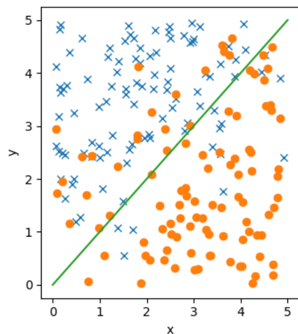
Los perceptrones de una sola capa son solo capaces de aprender patrones separables linealmente.

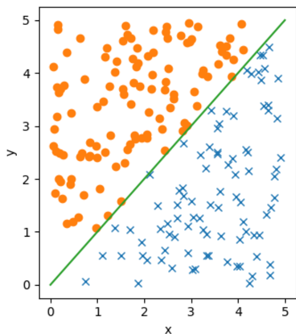
Sigmoide. Función de activación suave



La función de Heaviside tiene un cambio abrupto, en su lugar proponemos la sigmoide que las asíntotas son 0 y 1 pero cambia suavemente entre estos extremos alrededor del 0:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$





La sigmoide le da una probabilidad no nula a los puntos.

Regresión logística

Como se hizo con regresión lineal podemos trabajar con un conjunto base de funciones $\mathbf{\Phi}$,

$$f(\mathbf{x},\mathbf{w}) = \sigma(\mathbf{w}\cdot\mathbf{\Phi}(\mathbf{x}))$$

¿Como hacemos para determinar los pesos? $\nabla J(\mathbf{w}) = 0$

$$J(\mathbf{w}) = \sum_{n=1}^{N_D} (1-y_n)\log[1-\sigma(\mathbf{w}\cdot\mathbf{\Phi}(\mathbf{x}_n))] + y_n \log[\sigma(\mathbf{w}\cdot\mathbf{\Phi}(\mathbf{x}_n))]$$

$$\nabla J(\mathbf{w}) = \sum_{n=1}^{N_D} [y_n - \sigma(\mathbf{w}\cdot\mathbf{\Phi}(\mathbf{x}_n))] = 0$$

No hay solución analítica

Uso de algoritmos de optimización: descenso de gradientes o métodos newtonianos (uso del Hessiano).

Clasificación multiclase

Si tenemos $N = 7$ clases los vamos a representar por un vector binario:
$\mathbf{y} = (0, 0, 0, 0, 1, 0, 0)$
Vale la interpretación de que el valor de la variable también es la probabilidad de la variable.
Definimos a la distribución de probabilidad categórica:

$$Pr(y = k) = \lambda_k$$

donde λ_k es la probabilidad de la categoría k.
Basada en el vector binario la podemos reescribir como: $Pr(\mathbf{y}) = \prod_{k=1}^{K} \lambda_k^{y_k}$

Función softmax

$$\sigma_k(\mathbf{x}) = \frac{\exp x_k}{\sum_{k'=1}^{K} \exp x_{k'}}$$

Notar que si sumamos la función softmax de cada categoría:

$$\sum_{k=1}^{K} \sigma_k(\mathbf{x}) = \frac{\sum_{k=1}^{K} \exp x_k}{\sum_{k'=1}^{K} \exp x_{k'}} = 1$$

Por otro lado para cualquier k tenemos que $0 < \sigma_k(\mathbf{x}) \leq 1$.
La función tiene las restricciones necesarias para que la salida sean las probabilidades λ_k.

Regresión softmax

Modelo propuesto:

$$\lambda_k = f_k(\mathbf{x}, \mathbf{w}) = \sigma_k(\mathbf{w} \cdot \mathbf{\Phi}(\mathbf{x}))$$

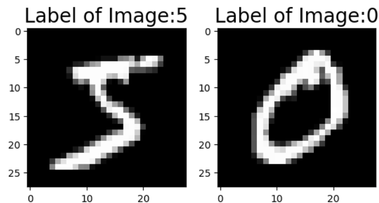
La función de log-verosimilitud es

$$\begin{aligned} l(\mathbf{w}) &= \log \prod_{n=1}^{N_D} \prod_{k=1}^{N_K} \lambda_{nk}^{y_{nk}} \\ &= \sum_{n=1}^{N_D} \sum_{k=1}^{N_K} y_{nk} \log \lambda_{nk} \\ &= \sum_{n=1}^{N_D} \left[\sum_{k=1}^{N_K} y_{nk} \right] \end{aligned}$$

Ejemplo con el MNIST

Digitalización de imágenes con números manuscritos (Lecun 198xs)

Tengo que clasificar (mapear) a la imagen con un dígito.



Definición del modelo en pytorch

```
class LogisticRegression(tn.Module):
    def __init__(self,input_size,
                          num_classes):
        super().__init__()
        self.Linear = nn.Linear(input_size,
                               num_classes)
        self.Softmax= torch.softmax(dim=1)#
                                 activacion
    def forward(self,feature):
        output = self.Linear(feature)
        output = self.Softmax(output)
        return output

loss = torch.nn.CrossEntropyLoss()  #
                      perdida
```