Aprendizaje Automático III

Ana X. Ezquerro

ana.ezquerro@udc.es, GitHub

Grado en Ciencia e Ingeniería de Datos
Universidad de A Coruña (UDC)

Curso 2021-2022

Tabla de Contenidos

Tema 1: Técnicas avanzadas de preprocesado de datos

1.1. Métricas de evaluación ante el desbalanceo de datos

		Predicción	
		Positivos	Negativos
Real	Positivos	VP	FN
	Negativos	FP	VN

$$\underset{\text{(accuracy)}}{\text{Precisión}} = \frac{\text{VP}+\text{VN}}{N} \qquad \underset{\text{(error rate)}}{\text{Tasa error}} = \frac{\text{FP}+\text{FN}}{N}$$

$$\underset{\text{(recall)}}{\text{Sensibilidad}} = \frac{\text{VP}}{\text{VP}+\text{FN}} \qquad \underset{\text{(VPP, precision)}}{\text{Valor Predictivo Positivo}} = \frac{\text{VP}}{\text{VP}+\text{FP}}$$

$$\underset{\text{(specificity)}}{\text{Especificidad}} = \frac{\text{VN}}{\text{VN}+\text{FP}} \qquad \underset{\text{(VPN)}}{\text{Valor Predictivo Negativo}} = \frac{\text{VN}}{\text{VN}+\text{FN}}$$

$$\underset{\text{(Media armónica)}}{F_1\text{-score}} = \left(\frac{\text{Recall}^{-1}+\text{Precision}^{-1}}{2}\right)^{-1} = 2\cdot\frac{\text{Recall}\cdot\text{Precision}}{\text{Recall}+\text{Precision}}$$

$$F_\beta\text{-measure} = (1+\beta)^2\cdot\frac{\text{Recall}\cdot\text{Precision}}{\beta^2\cdot\text{Recall}+\text{Precision}}$$

$$\underset{\text{(geometric-mean)}}{G\text{-mean}} = \sqrt{\frac{\text{VP}}{\text{VP}+\text{FN}}\cdot\frac{\text{VN}}{\text{VN}+\text{FP}}} = \sqrt{\text{Recall}\cdot\text{Specificity}}$$

1.2. Algoritmos de extracción de características

1.2.1. Algoritmo del ReliefF

- Entradas: $D \in \mathbb{R}^{n\times m}$ dataset de n ejemplos y m atributos, t iteraciones del método, $C = c_1, ..., c_d$ clases.
- Salida: Vector W de las estimaciones de la calidad de los atributos.

1. Inicialización del vector W a 0.
2. Para $i = 1, ..., t$:
   - 2.1. Seleccionar una instancia X_i.
   - 2.2. Seleccionar las k instancias más cercanas de la misma clase c_X (*nearest hits* H_j para $j = 1, .., k$).
   - 2.3. Seleccionar las k instancias más cercanas para cada clase distinta de la de X_i (*nearest misses* $M_j(c)$ para $j = 1, ...k$ y para $\forall c \in C - c_X$)
   - 2.4. Para $f = 1, ..., m$ (actualización del vector W):

$$W[f] = W[f] - \frac{1}{k}\sum_{j=1}^{k}\text{diff}(f, X_i, H_j) + \frac{1}{k}\sum_{c\in\{C-c_X\}}\frac{\mathbb{P}(c)}{1-\mathbb{P}(c_X)}\sum_{j=1}^{k}\text{diff}(f, X_i, M_j(c))$$

   donde $\mathbb{P}(c)$ es la probabilidad *a priori* de la clase c (estimada a partir de D).

$$\text{diff}(f, X_1, X_2) = \begin{cases} 0 & X_1[f] = X_2[f] \\ 1 & \text{en otro caso} \end{cases}$$

   - Si X_i y H_j tienen valores diferentes para f, es un atributo no deseable.
   - Si X_i y M_j tienen valores diferentes para f, es un atributo deseable.

3. Finalmente, promediar por el número de iteraciones. $W = \frac{1}{t}W$

1.2.2. Algoritmo de mRMR (*Maximum Relevance Minimum Redundancy*)

- Entradas: Conjunto inicial de características S, número de características qu se quieren seleccionar m_{opt}, función para evaluar la correlación (información mutua) entre características y clases (`mutualInfo()`)
- Salidas: Conjunto de características seleccionadas S_{opt}.

1. Para cada característica f_i en S:
   1.1. Calcula la **relevancia** como `mutualInfo`(f_i, clases).
   1.2. Inicializa la **redundancia** a 0.
   1.3. Para cada característica f_j en S, se suma a la redundancia `mutualInfo`(f_i, f_j).
   1.4. Se le asocia a la característica f_i una magnitud v_i.

$$v_i = \text{relevancia} - \text{redundancia}$$

2. Las características se seleccionarán a partir de las magnitudes v_i para $i = 1, ..., m$ en orden de mayor a menor.

1.2.3. Algoritmo de SVM-RFE

- Entradas: Conjunto de entrenamiento D, conjunto inicial de características S.
- Salidas: Selección y ranking de características del dataset.

1. Inicialización del conjunto $F = S$.
2. Mientras que F no esté vacío:
   2.1. Entrenar F con una SVM lineal.
   2.2. Calcular el vector de pesos w_i.
   2.3. Ordenar las características de F por el vector de pesos w_i^2.
   2.4. Eliminar la última característica (la de menor peso) de F.
3. De esta forma se obtiene las características que se deben seleccionar en orden de menor a mayor relevancia.

Tema 2: Modelos combinados (*ensembles*)

2.1. Algoritmos clásicos

Pseudocódigo del AdaBoost

Entrada: $D = \{(x_1, y_1), ..., (x_n, y_n)\} = (\mathbf{X}, \mathbf{y})$ dataset de entrenamiento donde $y_i \in [-1, 1]$, K modelos de clasificación y una distribución W de pesos para los ejemplos de entrenamiento.

1. Inicialización de $W = \text{Unif}(1, n)$.
2. Para cada modelo (for $k = 1, ...K$):
   - 2.1. h_k: Modelo entrenado en base a D ponderado con W.
   - 2.2. Cálculo del error cometido por el modelo h_k:

$$\varepsilon_k = \sum_{i=1}^{n} W(i)\mathcal{L}(y_i, \hat{y}_i)$$

     donde $W(i)$ expresa el peso asociado al ejemplo i y $\mathcal{L}$ es una función que expresa el error en la predicción.
   - 2.3. Si $\varepsilon_k \geq 0.5$, se detiene el bucle.
   - 2.4. Cálculo de nuevas estimaciones para los pesos.

$$\alpha_k = \frac{1}{2} \log \frac{1 - \varepsilon_k}{\varepsilon_k}$$

   - 2.5. Actualizar la distribución W:

$$W = \frac{1}{Z_k} W \exp(-\alpha_k \mathbf{y}\hat{\mathbf{y}})$$

     donde Z_k es un factor de normalización para que W sea una distribución válida.
3. Para un punto nuevo (x_0, y_0) sobre el que se quiere realizar una predicción, utilizar:

$$H(x_0) = \text{sign}\left(\sum_{k=1}^{K} \alpha_k h_k(x_0)\right)$$

Pseudocódigo de Bagging

Entrada: $D = \{(x_1, y_1), ..., (x_n, y_n)\} = (\mathbf{X}, \mathbf{y})$ dataset de entrenamiento donde $y_i \in [-1, 1]$ y K modelos de clasificación.

1. Para cada modelo de clasificación (para $k = 1, ...K$).
   - 1.1. Construir una muestra bootstrap $D^{(k)}$ a partir del muestreo con reemplazamiento de n elementos de D.
   - 1.2. Entrenar el modelo M_k.
2. Devolver los K modelos obtenidos.

Pseudocódigo de Random Forest

Entrada: $D = \{(x_1, y_1), ..., (x_n, y_n)\} = (\mathbf{X}, \mathbf{y})$ dataset de entrenamiento donde $y_i \in [-1, 1]$, K árboles (tamaño del ensemble), subespacio de dimensión d.

1. Para cada árbol (*base leaner*) $k = 1, ...K$.
   - 1.1. Construir una muestra bootstrap $D^{(k)}$ a partir del muestreo con reemplazamiento de n elementos de D.
   - 1.2. Escoger d características de forma aleatoria.

1.3. Entrenar el modelo M_k con $D^{(k)}$.

2. Devolver los modelos entrenados.

Pseudocódigo de Gradient Boosting

Entrada: $D = \{(x_1, y_1), ..., (x_n, y_n)\} = (\mathbf{X}, \mathbf{y})$ dataset de entrenamiento donde $y_i \in [-1, 1]$, $\mathcal{L}$ función de pérdida (diferenciable), M iteraciones del algoritmo.

1. Inicialización de un modelo basado en una constante:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} \mathcal{L}(y_i, \gamma)$$

2. Para $m = 1, ..., M$:

2.1. Calcular los *pseudo*-residuos:

$$\varepsilon_{i,m} = -\left[\frac{\partial \mathcal{L}(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \qquad i = 1, ..., n$$

2.2. Entrenar un modelo $h_m()$ usando $\{(x_1, \varepsilon_{1,m}), ..., (x_n, \varepsilon_{n,m}\}$

2.3. Obtener el factor multiplicativo γ_m optimizando:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} \mathcal{L}(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$$

2.4. Finalmente, actualizar el modelo general:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x_i)$$

3. Devolver el modelo obtenido $F_M(x)$.

XGBoost es una variante de Gradient Boosting que consigue muy buenos resultados y es altamente paralelizable. Usa una formalización del modelo más regularizada para controlar el sobreajuste, lo que resulta en mejores resultados.

2.2. Reglas de decisión en modelos combinados

Sea la muestra x a la que le tenemos que asignar una de las d posibles clases del problema, y m el número de clasificadores con su y_i salidas, para $i = 1, ..., m$.

Cuando los clasificadores proporcionan un grado de certeza, la probabilidad a posterior puede estimarse como: $\mathbb{P}(c_j|x) = y_i$.

Sea $y_{ij}(x)$ la salida del clasificador i $(i = 1, ..., m)$ en la clase j $(j = 1, .., d)$ para la muestra x, asumiendo que lasa salidas y_i están normalizadas. Se puede asignar a la muestra x la clase c_k según las siguientes reglas:

- Regla del producto: $k = \arg\max_{j=1...d} \left\{ \prod_{i=1}^{m} y_{ij}(x) \right\}$
- Regla de la suma: $k = \arg\max_{j=1...d} \left\{ \sum_{i=1}^{m} y_{ij}(x) \right\}$.
- Regla de la media: $k = \arg\max_{j=1...d} \left\{ \frac{1}{m} \sum_{i=1}^{m} y_{ij}(x) \right\}$.
- Regla del máximo: $k = \arg\max_{j=1...d} \left\{ \max_{i=1...m} y_{ij}(x) \right\}$.
- Regla del mínimo: $k = \arg\max_{j=1...d} \left\{ \min_{i=1...m} y_{ij}(x) \right\}$.

2.3. Medidas de diversidad entre clasificadores

$$S_{\text{AMPLE}} = \left| \left(\frac{a}{a+b} \right) / \frac{c}{c+d} \right| = \left| \frac{a(c+d)}{c(a+b)} \right|$$

		Clasificador 2	
		Aciertos	Fallos
Clasificador 1	Aciertos	a	b
	Fallos	c	d

Tema 3: Aprendizaje por refuerzo

3.1. *k-Armed Bandit Problem*

Supongamos que:

- Existen k opciones/acciones a escoger.
- Al escoger una acción, se recibe una recompensa numérica escogida de una distribución de probabilidad que depende de la acción escogida.
- El objetivo es maximizar las recompensas obtenidas después de T iteraciones.

Sea A_t la acción tomada en la iteración t y R_t su recompensa correspondiente. El valor de una acción arbitraria a es el valor esperado de su recompensa dado que se ha escogido dicha acción:

$$q_*(a) = \mathbb{E}[R_t | A_t = a]$$

Sea $Q_t(a)$ la estimación de $q_*(a)$ en la iteración t (deseamos que $Q_t(a) = q_*(a)$):

$$Q_t(a) = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{I}(A_i = a)}{\sum_{i=1}^{t-1} \mathbb{I}(A_i = a)}$$

A la hora de seleccionar una acción en la iteración t, se pueden tomar distintas opciones:

a) *Greedy* action.

$$A_t = \underset{a}{\operatorname{argmax}}\ Q_t(a)$$

b) ε-*greedy* action.

c) *Optimistic Initial Values*: Consiste en, para favorecer la exploración de más acciones al inicio del algoritmo, iniciar los valores de $Q_1(a)$, $\forall a \in A$ con valores mucho mayores a sus recompensas.

3.2. Procesos de Decisión de Markov (*Markov Decision Processes*, MDPs)

Consideremos un agente y un entorno con el que interactúa en cada momento $t = 0, 1, 2, 3...$ (tiempo discreto).

- En cada momento t, el agente obtiene una representación del entorno denominada **estado** ($S_t \in \mathcal{S}$) y realiza una acción $A_t \in \mathcal{A}(s)$.
- En el siguiente momento $t+1$, recibirá una recompensa numérica ($R_{t+1} \in \mathcal{R} \subset \mathbb{R}$) y pasará al siguiente estado S_{t+1}.
- El MDP y el agente realizan una trayectoria del tipo:

$$S_0, A_0, R_1; \quad S_1, A_1, R_2; \quad ...$$

- En un MDP *finito*, el conjunto de estados, acciones y recompensas $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ tiene un número finito de elementos.

Para cada valor aleatorio de las variables R_t y S_t (en un tiempo cualquiera t) podemos definir su probabilidad de ocurrencia dado un estado anterior S_{t-1} y la acción tomada en $t-1$, A_{t-1}:

$$p : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \longrightarrow [0, 1]$$
$$p(s', r|s, a) = \mathbb{P}[S_t = s', R_t = r | S_{t-1} = s, A_{t-1} = a]$$
$$\forall s', s \in \mathcal{S}, r \in \mathcal{R}, a \in \mathcal{A}(s)$$

Esta función p caracteriza completamente la dinámica del entorno. A partir de ella se pueden definir otras funciones como:

- La **probabilidad de transición** entre estados:

$$p(s'|s,a) = \mathbb{P}[S_t = s'|S_{t-1} = s, A_{t-1} = a] = \sum_{r\in\mathcal{R}} p(s', r|s, a)$$

- La **recompensa esperada** en t dado el *estado* y *acción* anterior (en $t-1$):

$$r(s,a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a)$$

- La **recompensa esperada** en t dado el estado actual, el estado anterior y al acción anterior.

$$r(s,a,s') = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r\in\mathcal{R}} r\frac{p(s'|s,a)}{p(s', r|s, a)}$$

El objetivo del agente es maximizar la acumulación de recompensas a lo largo de un episodio. Sea G_t el **retorno esperado**, éste se define como:

$$G_t = R_{t+1} + R_{t+2} + \cdots + R_t$$

donde T es el tiempo final del episodio. Otra forma de definir el retorno esperado es descontando las recompensas a la largo plazo:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{T} \gamma^k R_{t+k+1}$$

donde $\gamma \in [0, 1]$ es la **tasa de descuento**. Nótese que:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots), \qquad G_T = 0 \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned}$$

3.2.1. Políticas y funciones de evaluación

Una **política** π en el tiempo t se define como la probabilidad de tomar la acción $A_t = a$ dado un estado $S_t = s$.

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s], \qquad a \in \mathcal{A}(S_t)$$

- El **valor del estado s bajo la política π**:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \Big| S_t = s\right], \qquad \forall s \in \mathcal{S}$$

  Podemos reescribir $v_\pi(s)$ como (ecuaciones de Bellman):

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\Big(r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\Big) \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big(r + \gamma v_\pi(s')\Big) \end{aligned}$$

- El **valor de la acción a en el estado s bajo la política π**:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \Big| S_t = s, A_t = a\right]$$

- Una políica π es mejor que otra π' si y sólo si $v_\pi(s) \geq v_{\pi'}(s)$, $\forall s \in \mathbb{S}$.
- Función de valor de estado óptima:

$$v^*(s) = \max_\pi v_\pi(s), \quad \forall s \in \mathbb{S}$$

- Función de valor de acción óptima:

$$q^*(s,a) = \max_\pi q_\pi(s,a), \qquad \forall s \in \mathbb{S},\ a \in \mathbb{A}$$

3.2.2. Ecuaciones de Bellman

$$\begin{aligned} v_\pi(s) &= \mathbb{E}[G_t|S_t = s] = \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big(r + \gamma \mathbb{E}[G_{t+1}|S_{t+1} = s']\Big) \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big(r + \gamma v_\pi(s')\Big) \end{aligned}$$

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}[G_t|S_t = s, A_t = a] = \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)\Big(r + \gamma \mathbb{E}[G_{t+1}|S_{t+1} = s']\Big) \\ &= \sum_{s',r} p(s', r|s, a)\Big(r + \gamma \sum_{a'} \pi(a'|s') \mathbb{E}[G_{t+1}|S_{t+1} = s', A_{t+1} = a']\Big) \\ &= \sum_{s',r} p(s', r|s, a)\Big(r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a')\Big) \end{aligned}$$

3.3. Programación dinámica

Se basa en utilizar las ecuaciones de Bellman para calcular $v^*(s)$ y derivar a partir de ahí π^*.

1. Partimos de una política inicial $\pi(s)$, a partir de la cual estimamos $v_\pi(s)$, $\forall s \in \mathbb{S}$.

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r'} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

2. Podemos definir una nueva política π' *greedy* con respecto a v_π:

$$\pi'(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] = \arg\max_a q_\pi(s, a)$$

3.3.1. Iteración de valores para estimar $\pi \approx \pi^*$

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

3.4. Métodos de MonteCarlo

First-visit MC prediction, for estimating $V \approx v_\pi$

Input: a policy π to be evaluated
Initialize:
  $V(s) \in \mathbb{R}$, arbitrarily, for all $s \in \mathcal{S}$
  $Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Loop forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow G + R_{t+1}$
    Unless S_t appears in $S_0, S_1, \ldots, S_{t-1}$:
      Append G to $Returns(S_t)$
      $V(S_t) \leftarrow$ average$(Returns(S_t))$

Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s,a) \in \mathbb{R}$ (arbitrarily)
  $C(s,a) \leftarrow 0$
  $\pi(s) \leftarrow \arg\max_a Q(s,a)$ (with ties broken consistently)

Loop forever (for each episode):
  $b \leftarrow$ any soft policy
  Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
    $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
    If $A_t \neq \pi(S_t)$ then exit For loop
    $W \leftarrow W \frac{1}{b(A_t|S_t)}$

3.5. Temporal Difference

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    $A \leftarrow$ action given by π for S
    Take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
    $S \leftarrow S'$
  until S is terminal

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ϵ-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ϵ-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S', A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ϵ-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

Expected SARSA: Actualización de valores:

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \cdot \mathbb{E}[Q(S_{t+1}, A_{t+1})|S_{t+1}] - Q(S_t, A_t)\Big] \\ &\leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \cdot \sum_{a \in \mathcal{A}(S_{t+1})} \Big(\mathbb{P}(a|S_{t+1}) Q(S_{t+1}, a)\Big) - Q(S_t, A_t)\Big] \end{aligned}$$

donde A_{t+1} se obtiene siguiendo una políitca ε-*greedy*.

3.6. Optimización de una política aprendiendo el modelo

Tabular Dyna-Q

Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
  (a) $S \leftarrow$ current (nonterminal) state
  (b) $A \leftarrow \epsilon$-greedy(S, Q)
  (c) Take action A; observe resultant reward, R, and state, S'
  (d) $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
  (e) $Model(S, A) \leftarrow R, S'$ (assuming deterministic environment)
  (f) Loop repeat n times:
    $S \leftarrow$ random previously observed state
    $A \leftarrow$ random action previously taken in S
    $R, S' \leftarrow Model(S, A)$
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$

Dyna-Q+: La recompensa viene ponderada por el número de *steps* sin explorar ese estado (τ):

$$r = r + \kappa\sqrt{\tau}, \qquad \kappa > 0$$

3.7. Aproximación de funciones

Gradient Monte Carlo Algorithm for Estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, S_1, A_1, \ldots, R_T, S_T$ using π
  Loop for each step of episode, $t = 0, 1, \ldots, T-1$:
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[G_t - \hat{v}(S_t,\mathbf{w})\big]\nabla\hat{v}(S_t,\mathbf{w})$

Semi-gradient TD(0) for estimating $\hat{v} \approx v_\pi$

Input: the policy π to be evaluated
Input: a differentiable function $\hat{v} : \mathcal{S}^+ \times \mathbb{R}^d \to \mathbb{R}$ such that $\hat{v}(\text{terminal},\cdot) = 0$
Algorithm parameter: step size $\alpha > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose $A \sim \pi(\cdot|S)$
    Take action A, observe R, S'
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})\big]\nabla\hat{v}(S,\mathbf{w})$
    $S \leftarrow S'$
  until S' is terminal

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

Tema 4: Aprendizaje semi supervisado

4.1. Modelos generativos

4.1.1. Cluster and label

Hipótesis: Los clusters coinciden con las fronteras de decisión.

Entrada:

- Conjunto de datos etiquetados $\mathbf{X}^l = \{(x_i, y_i),\ i = 1, ..., n^l\}$ y sin etiquetar $\mathbf{X}^u = \{x_i,\ i = 1, ..., n^u\}$.
- Algoritmo de supervisado $f : x \to y$.

1. Obtener el conjunto de clusters $\mathbb{S} = \{S_i,\ i = 1, ..., n^c\}$ para todos los datos: $\mathbf{X}^l \cup \mathbf{X}^u$.
2. Para cada cluster k, entrenar el algoritmo supervisado con las muestras de $\mathbf{X}^l$ que pertenezcan a ese cluster. El resultado es un clasificador $f_k : x \to y$ donde $x \in S_k \cap \mathbf{X}^l$.
3. Aplicar f_k a las muestras sin etiquetar de S_k.
4. Resultado: Etiquetas para $\mathbf{X}^u$.

4.2. Modelos discriminativos

4.2.1. S3VM

Entrada:

- Conjunto de datos etiquetados $\mathbf{X}^l = \{(x_i, y_i),\ i = 1, ..., n^l\}$ y sin etiquetar $\mathbf{X}^u = \{x_i,\ i = 1, ..., n^u\}$.
- Función de separación: $f(x) = h(x) + b$, donde h es una función kernel en el espacio de Hilbert H_k.
- Clasificación de un x_i: sign$(f(x_i))$
- Función de pérdida (Hinge): $(1 - y_i f(x_i))_+$.

$$\mathcal{L} = \sum_{i=1}^{n^l} (1 - y_i f(x_i))_+ + \lambda \|h\|^2$$

1. Añadimos los datos sin etiquetar con la función de pérdida: $(1 - \text{sign}(f(x_i)) f(x_i))_+ = (1 - |f(x_i)|)_+$:

$$\mathcal{L} = \sum_{i=1}^{n^l} (1 - y_i f(x_i))_+ + \lambda \|h\|^2 + \gamma \sum_{i=1}^{n^u} (1 - |f(x_i)|)_+$$

4.3. Modelos basados en grafos

Entrada:

- Conjunto de datos etiquetados $\mathbf{X}^l = \{(x_i, y_i),\ i = 1, ..., n^l\}$ y sin etiquetar $\mathbf{X}^u = \{x_i,\ i = 1, ..., n^u\}$.
- Grafo G donde los nodos son las instancias y las aristas indican el grado de similitud entre las instancias.
- w_{ij} = Similitud directa entre x_i y x_j (instancias de $\mathbf{X}^l \cup \mathbf{X}^u$).

1. Hipótesis: Las predicciones de todas las observaciones se pueden definir con una función f.
2. Función de regularización:

$$\mathcal{L} = \min_f \sum_{i=1}^{l} (y_i - f(x_i))^2 + \lambda \sum_{x_i, x_j \in \mathbf{X}^l \cup \mathbf{X}^u} w_{ij} (f(x_i) - f(x_k))^2$$

3. Resoluciones:
   - Algoritmo de mincut: Fijar $f(x_i)$ para obtener $f(x_j)$.

- Paseo aleatorio de un nodo x_i a x_j con probabiildad: $w_{ij}/\sum_k w_{ik}$ y parar cuando se encuentra un nodo etiquetado.

$$f(x_i) = \mathbb{P}(x_j \in \mathbf{X}^l | x_i)$$

4.4. Algoritmo de co-entrenamiento

Entrada:

- Conjunto de datos etiquetados $\mathbf{X}^l = \{(x_i, y_i),\ i = 1, ..., n^l\}$ y sin etiquetar $\mathbf{X}^u = \{x_i,\ i = 1, ..., n^u\}$.
- Dos clasificadores f_1 y f_2 para distintas características de las observaciones.

1. Asumimos que $\mathbf{X}_1$ y $\mathbf{X}_2$ son los conjuntos de entrenamiento de f_1 y f_2 respectivamente. Inicialmente $\mathbf{X}_1 = \mathbf{X}_2 = \mathbf{X}^l$
2. Entrenar f_1 y f_2 con sus conjuntos de entrenamiento.
3. Predecir $\mathbf{X}^u$ con f_1 y f_2.
4. Los k pares con más certeza de $(x_i, f_1(x_i))$, $x_i \in \mathbf{X}^u$ se añaden $\mathbf{X}_2$.
5. Los k pares con más certeza de $(x_i, f_2(x_i))$, $x_i \in \mathbf{X}^u$ se añaden $\mathbf{X}_1$.
6. Volver al paso 2.