
# Cuestionario 3
# Aprendizaje Automático

**Alumno:** Christian Vigil Zamora
**DNI:**
**Curso:** 3º

27 de agosto de 2019

# 1. Preguntas de Teoría

**1.** Podría considerarse Bagging como una técnica para estimar el error de predicción de un modelo de aprendizaje? Diga si o no con argumentos. En caso afirmativo compárela con validación cruzada.

**Solución:**
Sí, podría considerarse como una técnica para estimar el error de predicción de un modelo de aprendizaje. Para argumentarlo, recurro a comentar que Bagging se basa en Bootstrapping, es decir, se remuestran de forma aleatoria y con reemplazamiento el conjuntos de datos para obtener nuevas muestras, siempre manteniendo el tamaño del conjunto de datos. Para cada una de las muestras generadas, que no dejan de ser conjuntos de entrenamiento, se podría entrenar un método de aprendizaje con ellos, y obtener modelos de predicción. Por lo tanto, recordando la definición de la técnica de Bootstrap, usada por Bagging, podríamos cuantificar la incertidumbre asociada de un modelo de aprendizaje. De ésta forma, éste proceso se repetiría múltiples veces, y Bagging obtendría el promedio de los resultados de predicción, lo que sería equivalente a estimar el error de predicción de un modelo. Comparar Bagging con Validación Cruzada es posible pues presentan cosas en común, pero también diferencias. Primero voy a comenzar hablando de las **cosas en común**: Ambas técnicas tienen la misma finalidad, la cuál es validad un modelo de aprendizaje. Ambas particionan el conjunto de datos de entrenamiento para generar nuevas muestras. Por último, podríamos decir que ambas técnicas utilizan un conjunto de los datos para considerarlo como Test, en Bagging se consideraría como Test los datos que no hayan sido seleccionados para la muestra (Out-of-Bag), ya que se pueden repetir y eso da lugar a que algunos datos no aparezcan y en Validación Cruzada, siempre se particiona el conjunto de datos, reservando una partición para Test. Ahora, voy a comentar las **diferencias** que presentan: Si bien es cierto que ambas particionan el conjunto de datos de entrenamiento para generar nuevas muestras, en Bagging está permitido el reemplazamiento mientras que en Validación Cruzada no. Otra diferencia sería que la técnica de la Validación Cruzada se ejecuta tantas veces como particiones se realicen sobre los datos, cosa que en Bagging no. Por otro lado, las muestras nuevas que genera Bagging mantienen el tamaño del conjunto de datos mientras que Validación Cruzada no, pues particiona según las proporciones que se le indiquen, por ejemplo 80 - 20 %, una de las más comunes. Como vemos es posible comparar ambas técnicas, aunque a efectos prácticos, Bagging suele ser más usado para calcular el error de predicción sobre un conjunto de entrenamiento mientras que la Validación Cruzada estaría más enfocada a calcular el error de predicción sobre el conjunto de datos Test, el motivo es porque en ocasiones, Bagging puede no considerar ciertos datos para las muestras, dando lugar a que nunca se entrene con algunos datos, cosa que con

Validación Cruzada no ocurre.

**2.** Considere que dispone de un conjunto de datos linealmente separable. Recuerde que una vez establecido un orden sobre los datos, el algoritmo perceptron encuentra un hiperplano separador iterando sobre los datos y adaptando los pesos de acuerdo al algoritmo

**Algorithm 1 Perceptron**
1: **Entradas**: $(\mathrm{x}_i, y_i), i = 1, \ldots, n, \mathrm{w} = 0, k = 0$
2: **repeat**
3 : $k \leftarrow (k+1)$ mód $n$
4: **if** $\text{sign}(y_i) \neq \text{sign}\left(\mathbf{w}^T \mathbf{x}_i\right)$ **then**
5: $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$
6 : **end if**
7 : **until** todos los puntos bien clasificados

Modificar este pseudo-código para adaptarlo a un algoritmo simple de SVM, considerando que en cada iteración adaptamos los pesos de acuerdo al caso peor clasificado de toda la muestra. Justificar adecuadamente/matematicamente el resultado, mostrando que al final del entrenamiento solo estaremos adaptando los vectores soporte.
**Solución:**

**Algorithm Perceptron-to-SVM**
1: **Entradas**: $(\mathrm{x}_i, y_i), i = 1, \ldots, n, \mathrm{w} = 0, k = 0$
2: **repeat**
3 : $k \leftarrow (k+1)$ mód $n$
4: **peor** $\leftarrow$ peorIndice(w,x,y)
5: $\mathbf{w} \leftarrow \mathbf{w} + y_{peor} \mathbf{x}_{peor}$
6 : **end if**
7 : **until** margen-óptimo

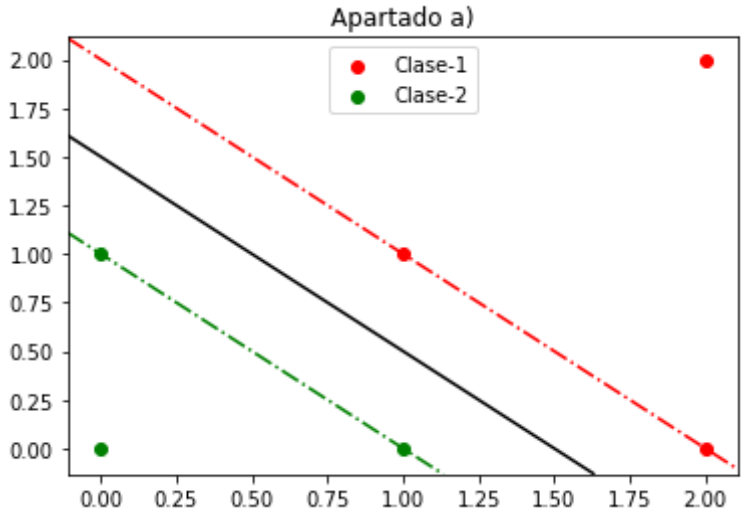
A continuación voy a explicar el por qué de la modificación y el funcionamiento del método **peorIndice**: Como se puede observar, no hay demasiados cambios en el pseudo-código, principalmente porque el objetivo es muy similar, ya que los pesos se van adaptando también, pero en ésta ocasión considerando al peor clasificado. El motivo de adaptar los pesos de acuerdo al peor clasificado es porque contra peor sea la clasificación, más alejado estaremos del hiperplano, por lo tanto el objetivo es conseguir que $y_n\left(\mathbf{w}^{\mathrm{T}} \mathbf{x}_{\mathrm{n}} + b\right) \geq 1, \quad (\mathrm{n} = 1, \ldots, N)$. Para obtener el índice del peor clasificado recurro a la función **peorIndice**, la cuál se encarga de

iterar sobre los datos y buscar el peor de entre los que no satisfacen la condición mencionada anteriormente ($y_n \left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_{\mathrm{n}} + b\right) \geq 1$), además de considerar la distancia de ellos al hiperplano, buscando el que tenga la máxima, de entre los peores clasificados. Una vez se obtiene el índice del peor, se adaptan los pesos de igual forma que en el Perceptrón pero considerando ese índice.Por lo tanto, conforme vayamos adaptando los pesos llegará un momento en el que estaremos adaptando los vectores soporte pues son los más cercanos al hiperplano, es decir, aquellos datos bien clasificados ya, cuya distancia al hiperplano es la mínima.

**3.** Considerar un modelo SVM y los siguientes datos de entrenamiento: Clase-1: $\{(1,1),(2,2),(2,0)\}$Clase- 2 : $\{(0,0),(1,0),(0,1)\}$
a) Dibujar los puntos y construir por inspección el vector de pesos para el hiperplano óptimo y el margen óptimo.
b) ¿Cuáles son los vectores soporte?
c) Construir la solución en el espacio dual. Comparar la solución con la del apartado (a)

**Solución:**

**a)**




El apartado puede ser resuelto por medio de la librería Scikit-Learn, aunque voy a comentar el funcionamiento de la construcción por inspección del vector de pesos para el hiperplano y el margen óptimo. Si observamos los datos, fácilmente apreciamos que al margen por parte de la Clase-1 va a estar delimitado por la ecuación y = 1 - x, mientras que el margen por parte de la Clase-2 por la ecuación

y = 2 - x. Una vez tenemos dichas ecuaciones podemos determinar que la separación de ambas clases vendrá dada por la ecuación y = $\frac{3}{2} - x$. Ya tenemos todos los elementos necesarios para construir el vector de pesos del hiperplano. Basta con resolver un sistema de ecuaciones, tomando valores de puntos dibujados, para finalmente obtener que el vector de pesos **w** = (0,0). Como he dicho al principio, me he basado en realizar el problema con Scikit-Learn y así comprender y visualizar de mejor forma el resultado.
En la imagen, los puntos rojos se corresponden con la Clase-1 y los puntos verdes con la Clase-2. El margen óptimo viene delimitado por la diferencia de espacio entre las líneas discontínuas y el hiperplano óptimo, representando en color negro.

**b)** Los vectores soporte son 2, en éste caso, el vector soporte perteneciente a la Clase-1 sería el formado por (0,1) y (1,0). Por otro lado, el vector soporte perteneciente a la Clase-2 sería el formado por los puntos (1,1) (2,0). Si visualizamos la imagen anterior, es fácil y razonable llegar a dicha conclusión, puesto que son los puntos que están a menor distancia del hiperplano.

**c)**

**4.** ¿Cúal es el criterio de optimalidad en la construcción de un árbol? Analice un clasificador en árbol en términos de sesgo y varianza. ¿Que estrategia de mejora propondria?

**Solución:**
El criterio de optimalidad en la construcción de un árbol sería buscar modelos más simples y opitimizar la compensación entre la complejidad del modelo y la precisión de la descripción del modelos de los datos de entrenamiento. En definitiva, lo que se conoce como 'La Navaja de Occam', que nos dice que los árboles más simples son preferibles a los grandes, puesto que el número de ellos es menor que el número de árboles complejos, por tanto es más probable que un árbol simple que se ajuste a los datos sea el correcto. Además, al ser más simple, es más probable que sea más general que un árbol complejo y que requiera menos espacio. Voy a considerar los Árboles de Decisión como clasificador para analizar en términos de sesgo y varianza. Los Árboles de Decisión presentan un bajo sesgo, pues tienen en cuenta pocas suposiciones sobre la función objetivo, lo que hace que el aprendizaje sea más lento pero con un mejor rendimiento. Como contra, presentan una alta varianza, es decir, sugiere grandes cambios en la estimación de la función objetivo de un conjunto de entrenamiento a otro. Evidentemente, ese aspecto es negativo. La estrategia de mejora que propongo es lo que se conoce como **Bias-Variance Trade-Off**, la cuál consiste en conseguir un equilibrio entre sesgo y varianza, de

forma que ambas sean bajas, y así conseguir mejores resultados en la predicción. Por tanto, como en los Árboles de Decisión la varianza es alta, propongo reducir la varianza a través de la combinación de los resultados de varios clasificadores, cada uno de ellos modelados con diferentes muestras generadas a partir del mismo conjunto. Una técnica para reducir la varianza sería **Bagging**, comentado en el primer ejercicio.

**5.** ¿Cómo influye la dimensión del vector de entrada en los modelos: SVM, RF, Boosting and NN?.

**Solución:**

- **SVM:** En dicho modelo, particularmente la dimensión del vector de entrada no es un problema. Si la dimensión es alta, ofrece un rendimiento muy efectivo. Si la dimensión no es tan alta, pero sigue siendo mayor que el número de muestras, es efectivo también. En otros casos, permite elegir la función Kernel que mejor se ajusta a los datos y el término de Regularización para evitar Overfitting.

- **RF:** De nuevo, en éste modelo la dimensión del vector de entrada no influye. Permite vectores de entrada tanto con alta dimensionalidad como con baja, aunque es obvio, pues Random Forest trabaja con subconjuntos extraídos del conjunto inicial.

- **Boosting:** Nuevamente, en éste método, la dimensión del vector de entrada no influye. La única precondición que impone Boosting a los datos es que posean al menos 2 características.

- **NN:** Para las Redes Neuronales, la dimensión del vector de entrada sí es crucial, puesto que un vector de entrada de dimensión reducida pues ocasionar Overfitting, por lo tanto, siempre es beneficioso que la dimensión del vector de entrada sea grande tanto para estimar los parámetros de aprendizaje como para la predicción.

**6.** El método de Boosting representa una forma alternativa en la búsqueda del mejor clasificador respecto del enfoque tradicional implementado por los algoritmos PLA, SVM, NN, etc.

a) Identifique de forma clara y concisa las novedades del enfoque;
b) Diga las razones profundas por las que la técnica funciona produciendo buenos ajustes (no ponga el algoritmo);
c) Identifique sus principales debilidades;
d) ¿Cuál es su capacidad de generalización comparado con SVM?

**Solución:**

**a)** La principal novedad del enfoque es que el método de Boosting es una combinación de predictores simples, entendiendo por simples predictores aquellos casi aleatorios, con una tasa de predicción en torno al 50 %. Por lo tanto, la clasificación es obtenida mediante la suma ponderada de las predicciones obtenidas por parte de los predictores.

**b)** La técnica funciona produciendo buenos ajustes debido a que los predictores simples se ponderan de acuerdo con su precisión. ¿En qué influye ese hecho? En cada iteración, los datos con los que se entrena se vuelven a ponderar, de forma que los datos bien clasificados pierden peso y los mal clasificados ganan peso. Por lo tanto, en próximas iteraciones los predictores con más peso, se centrarán principalmente en aquellos datos con mayor peso, es decir, en los datos MAL clasificados.

**c)** Una de sus principales desventajas es que le afecta de mayor forma los outliers y el ruido en los datos. Otra desventaja sería el hecho de tener que adaptarse para soportar problemas multi-clase. Y por último, coste computacional es más elevado.

**d)** La capacidad de generalización del método de Boosting es mayor con respecto a SVM. El hecho de combinar las predicciones de diferentes estimadores dentro del propio método hace que su capacidad de generalización sea mayor, pues es bastante más probable de que con alguno de los predictores consigamos generalizar ante cualquier conjunto de datos, mientras que en SVM también se puede alcanzar una buena generalización, pero de forma más complicada y entrando en juego la correcta estimación de los parámetros.

**7.** Discuta pros y contras de los clasificadores SVM y Random Forest (RF). Considera que SVM por su construcción a través de un problema de optimización debería ser un mejor clasificador que RF. Justificar las respuestas.

**Solución:**
En primer lugar, comienzo discutiendo los pros y contras del clasificador **SVM:**

- **Pros**: Es efectivo en espacios de alta dimensión. Utiliza un subconjunto de

los puntos de entrenamiento (**vectores soporte**) en la función de decisión. El hecho de usar sólo dichos puntos hace que sea eficiente en términos de memoria. Permite adaptarse al conjunto de datos eligiendo un tipo de Kernel. Permite Regularización. Si las condiciones lo permiten, encuentra el hiperplano óptimo con respecto a los datos, pues se construye a través de un problema de optimización.

- **Contras**: La elección del tipo de Kernel y de sus parámetros asociados es un proceso generalmente difícil. Si no se consigue de forma óptima lo anterior, es probable que aparezca Overfitting. Dificultad para afrontar problemas multi-clase.

A continuación, voy a discutir sobre **Random Forest**

- **Pros**: El hecho de combinar los resultados de diferentes predictores, ayuda a evitar Overfitting. Presenta menos Varianza que un Árbol de Decisión a secas, eso es debido a la combinación que lleva a cabo. Es un método flexible y con buena precisión. No importa el formato de los datos, no necesita adaptarse a ellos. Funciona bien con grandes dimensiones de datos. Afronta sin dificultad, problemas multi-clase.

- **Contras**: Es un modelo más complejo. No presenta los resultados de forma fácilmente interpretable. Necesita un número decente de datos para obtener buenos resultados. Puede aparecer Overfitting en casos de ruido.

No considero que SVM por su construcción deba ser mejor clasificador que RF, de hecho, uno de los factores que más influyen para decidir cuál es mejor o peor es el conjunto de datos. En términos teóricos SVM si debería ser mejor, pero en términos prácticos no siempre, ya que por suerte o por desgracia los datos no son siempre fácilmente ajustables para SVM, situación en la que RF no tiene problema alguno. Por lo tanto, considero que en términos prácticos, RF sería mejor clasificador.

**8.** ¿Cuál es a su criterio lo que permite a clasificadores como Random Forest basados en un conjunto de clasificadores simples aprender de forma más eficiente? ¿Cuales son las mejoras que introduce frente a los clasificadores simples? ¿Es Random Forest óptimo en algún sentido? Justifique con precisión las contestaciones.

**Solución:**
Precisamente, la propia pregunta casi contiene la respuesta. Bajo mi criterio, el hecho de combinar diferentes clasificadores simples para conseguir la mejor predicción es lo que permite aprender de forma más eficiente. Es decir, es mucho más

probable que mediante la conjunción de clasificadores simples se consiga llegar a una predicción aceptable dado que es posible abarcar diferentes escenarios gracias a los clasificadores simples, a diferencia de si tuvieramos un sólo clasificador más complejo, en el que el proceso sería mucho más ineficiente.
La principal mejora que aporta es el hecho de no considerar la correlación, es decir, se basa en Bagging pero los árboles que va construyendo en el desarollo de su ejecución no están correlados, de forma que los árboles no se parecerán, lo que permite explorar diferentes escenarios y conseguir predicciones que no estén correladas. Esa mejora a su vez conlleva obtener una mayor reducción de la Varianza.
En cuánto a si es óptimo en algún sentido, yo diría que no. De hecho considero que es el precio a pagar a cambio de las ventajas que presenta. Que RandomForest sea óptimo es un proceso muy complicado pues si tenemos en cuenta que en cada partición considera una muestra aleatoria de predictores de entre todos, un número aleatorio de variables para las ramificaciones de los árboles y selecciona de forma los valores para ellas de forma aleatoria también. Si recordamos que TODO ello es realizado para la partición, observamos que es un método que difícilmente puede llegar a ser óptimo, dado el coste que representa cada una de sus acciones.

**9.** En un experimento para determinar la distribución del tamaño de los peces en un lago, se decide echar una red para capturar una muestra representativa. Asi se hace y se obtiene una muestra suficientemente grande de la que se pueden obtener conclusiones estadisticas sobre los peces del lago. Se obtiene la distribución de peces por tamaño y se entregan las conclusiones. Discuta si las conclusiones obtenidas servirán para el objetivo que se persigue e identifique si hay algo que lo impida.

**Solución:**
Evidentemente, las conclusiones obtenidas **NO** servirán para el objetivo que se persigue y se debe a diferentes motivos, los cuáles describo a continuación. Obtener una muestra representativa sin tener ninguna información de los peces es complicado. En primer lugar, los peces grandes van a ser capturados por la red con mayor facilidad que los peces pequeños, de forma que la muestra ya no es representativa. Por otro parte, estamos asumiendo que justo en zona del lago donde se echa la red la distribución de peces grandes y pequeños sea equilibrada, pero obviamente ese dato no lo tenemos, por lo que existe la probabilidad de que en esa zona haya muchos más peces grandes que chicos, más peces chicos que grandes u cualquier combinación que hace que la muestra capturada no sea representativa.
Así que para finalizar, concluyo que las conclusiones obtenidas no servirán para el objetivo y la identificación de lo que lo impide está bastante claro que es el proceso de capturar una muestra, ya que no se tiene en cuenta ninguno de los factores comentados anteriormente, dando lugar a que la probabilidad de que la muestra

no sea representativa es bastante alta.

**10.** Identifique que pasos daría y en que orden para conseguir con el menor esfuerzo posible un buen modelo de red neuronal a partir una muestra de datos. Justifique los pasos propuestos, el orden de los mismos y argumente que son adecuados para conseguir un buen óptimo. Considere que tiene suficientes datos tanto para el ajuste como para el test.

**Solución:**
Antes debo comenzar, debo aclara que doy por hecho que los datos ya están leídos, por lo tanto no incluyo el procesado de datos como uno de los pasos que daría. Los pasos por tanto que daría para conseguir un buen modelo de red neuronal son:

1. Establecer el **Criterio de Inicialización** de los pesos, procurando no inicializarlos a 0 ó al mismo valor todos, siendo lo óptimo una inicialización con valores pseudo-aleatorios. Establecer el **Criterio de Parada**, pudiendo ser el número de iteraciones, el tamaño del Gradiente, el valor de Ein, aunque la mejor opción seria combinar diferentes criterios.

2. **Propagación hacia Adelante (Forward Propagation)**, con ésta acción "damos pasos"hacia adelante y comparamos los resultados obtenidos con los valores reales para obtener la diferencia entre la salida de la Red Neuronal y lo que debería haber salido. Es por tanto, una forma de comprobar cómo está funcionando la RN y encontrar errores. En ella se utilizan **funciones de activación**.

3. **Propagación hacia Atrás (Backpropagation)**, tras realizar la Propagación hacia Adelante, propagamos hacia atrás el Gradiente de error para poder actualizar los pesos. Es decir, conocemos el error y en éste paso se minimiza tanto como sea posible, y eso se consigue mediante la derivada de la función de error con respecto de los pesos, usando **SGD**.

4. **Fase de Entrenamiento**, vital y obvia a la hora de conseguir un modelo. Imprescindible para conseguir unos valores óptimos de los pesos y de sesgo. Será crucial también el valor considerado para el learning rate.

Esos son los pasos que daría para conseguir un buen modelo de Red Neuronal, en ese orden. El motivo queda reflejado en la descripción de cada paso, pues la Inicialización y el Criterio de Parada son dos puntos vitales a la hora de obtener un buen modelo. Forward Propagation y Backpropagation, con éstos pasos comprobamos que la Red Neuronal está funcionando como queremos, o en su defecto, tratamos de corregir y minimizar los fallos que encontremos. Por último, y como más obvio, la fase de Entrenamiento, crucial a la hora de obtener un modelo.

# 2. Referencias Bibliográficas


- https://scikit-learn.org/stable/
- https://medium.freecodecamp.org/building-a-3-layer-neural-network-from-scratc
- https://medium.com/
- https://towardsdatascience.com/
- Aurélien Géron.,(2019).Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow,EEUU