# Examen teórico módulo 5. Año 2020

Conteste a las siguientes cinco preguntas, utilizando como máximo un total de 400 palabras en cada una.

Bárbara Bellón Lara

1. **Señale diferencias y semejanzas entre los modelos de bagging y de boosting.**

Los modelos de bagging y boosting se basan ambos en la manipulación de los datos de entrenamiento donde se construye un grupo de modelos y se repiten los algoritmos varias veces. Lo que se quiere conseguir con ambos métodos es un equilibrio entre sesgo y varianza. Son modelos que al combinar varios métodos resuelven el problema del overfitting.

La principal diferencia entre estos dos modelos es que uno parte de bajo sesgo y alta varianza, estos son los modelos de bagging. Por otro lado, los modelos de boosting son modelos simples con baja varianza, pero bajo sesgo.

Los modelos de bagging se basan dividir el conjunto de entrenamiento en distintos conjuntos con reemplazamiento de n observaciones y agregar los resultados de estos modelos. El método de agregación suele ser la media para regresión y la moda para clasificación. Por tanto, partimos de k conjuntos de entrenamiento para los que se usa el mismo algoritmo de aprendizaje y las predicciones de los k modelos se agregan para obtener la predicción final. Los resultados de cada conjunto de entrenamiento, normalmente se utilizan algoritmos basados en árboles, tienen pocas observaciones en los nodos finales, por tanto, un sesgo muy bajo, son árboles muy grandes y con mucha varianza. Para reducir esta varianza se realizan k árboles en los que se combinan los resultados de muchos nodos terminales.

Boosting sin embargo, parte de árboles pequeños, con nodos terminales con muchas observaciones, por lo que hay poca varianza, pero un sesgo alto. La manera de reducir ese sesgo es realizando n iteraciones utilizando el mismo conjunto de entrenamiento y el mismo algoritmo. Al terminar las iteraciones se actualizan los pesos, incrementando el peso de los mal clasificados y reduciendo el de los correctamente clasificados.

Por tanto, las diferencias entre bagging y boosting se pueden resumir de la siguiente manera:

Bagging:

- Bajo sesgo, alta varianza
- K modelos complejos que se ponderan para reducir la varianza

Boosting

- Alto sesgo, baja varianza
- K iteraciones del modelo, en el que se actualizan los pesos de entrada en el conjunto de entrenamiento para reducir el sesgo.

**2. Explique al menos cinco híper parámetros del Perceptrón Multicapa.**

En los algoritmos MLP se pueden modificar diferentes parámetros:

- Valores de iniciación de los pesos, los pesos son la manera en la que se pondera la conexión de una neurona con otra, suelen elegirse de manera aleatoria, pero también se pueden hacer mediante algoritmos genéticos, autoencoder…
- El número de capas y el número de neuronas en cada capa. La elección del numero de capas y del numero de neuronas de cada una tiene que ser tal que se elija el mínimo numero de estas para que la red rinda de manera adecuada. Para comprobar el rendimiento se utilizará el conjunto de validación y se elegirá la que mejores resultados proporcione.
- Tasa de aprendizaje: es la que controla cuanto cambia la red en cada iteración. Si la tasa de aprendizaje es muy pequeña, la velocidad a la que la red converge es pequeña y por tanto es posible caer en mínimos locales, ya que en ese cambio tan pequeño no encuentra otra opción “mejor”. En cambien si el valor es muy alto, los saltos que la red da al intentar predecir el valor de salida son tan altos que se vuelve inestable y no llega a converger
- Factor momento: se utiliza para acelerar los procesos de convergencia y tiene en cuenta la dirección del incremento en la dirección anterior. Esto se utiliza para darle más rapidez a la serie.
- Decay: desaceleración de la red neuronal, mediante la modificación de los pesos cuando la red se vaya acercando al final ir desacelerando este cambio. De esto va a depender también el número de ciclos que se realicen en la red.
- Funciones de activación: las funciones de activación de las capas oculta y la de salida se pueden cambiar. Se utilizan funciones fácilmente derivables, las más normal es la sigmoide o la tangente hiperbólica.

3. **Explique cómo realizar una simulación de las Máquinas de Vectores Soporte con un kernel de base radial.**

   Lo primero sería realizar la división de los conjuntos de entrenamiento y validación.

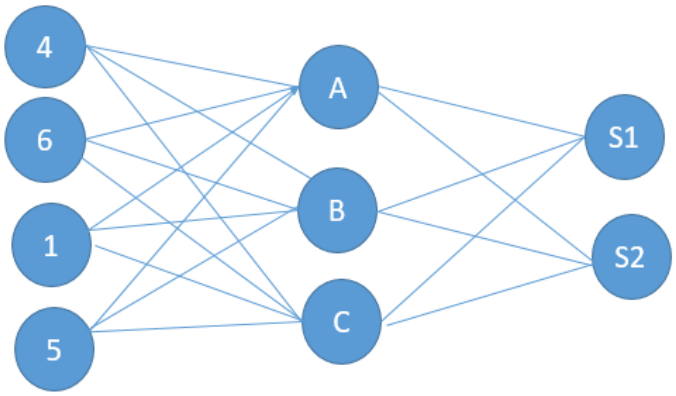
   Tras esto se realiza la etapa de entrenamiento del modelo con dicho conjunto de entrenamiento.

   Primero se seleccionan las opciones de control del entrenamiento, Cross-validation o repeated Cross-Validation y las particiones y repeticiones que se quieran, normalmente entre 5 y 10 particiones y unas 3 repeticiones. Si es un problema de clasificación habrá que seleccionar que nos prediga las probabilidades. Y el tipo de resumen de métricas que queremos obtener. Se selecciona la métrica, que va a depender del problema a resolver. Si es de clasificación serán ROC, Kappa o Accuracy las mas comunes. Si el problema es de regresión se utilizarán MSE, RMSE.

   Para obtener los parámetros óptimos hay que explorar distintos hiperparámetros para que nos proporcionen el mejor resultado posible. Para ello se realiza un expand grid de los parámetros, esto es la creación de una rejilla que combina todos los parámetros que se le indican al problema para probar todas las combinaciones posibles. Para un kernel en base radial se va a necesitar indicar dos parámetros: C y sigma (o lambda, dependiendo de la librería utilizada). C nos da una medida del coste de la mala clasificación y sigma es la varianza del kernel radial, por lo que cuanto mayor más varianza y más holgura habrá.

   Al final se seleccionará el modelo que mejor métrica proporcione, es decir, en el caso de ROC, Kappa o Accuracy, el que valores más altos proporcione. En el caso de regresión el criterio es al contrario pues las métricas nos dan un indicador del error, por tanto, cuanto menor el error mejor el modelo.

4. **Para la siguiente estructura de Perceptrón Multicapa y los datos de la tabla que se muestran a continuación y, sabiendo que la función de transferencia para las neuronas de la capa intermedia es una tangente hiperbólica y para las neuronas de la capa de salida es una sigmoide, calcule los resultados que proporciona para las dos neuronas de salida. Nota: se prescinde del sesgo.**




| **Unidades de Entrada** | | **Pesos entre la capa de entrada y la capa oculta** | | | |
|---|---|---|---|---|---|
| 4 | | 0,1 | 0,2 | 0,5 | |
| 6 | | 0,3 | 0,1 | 0,3 | |
| 1 | | 0,2 | 0,4 | 0,5 | |
| 5 | | 0,1 | 0,5 | 0,2 | |
| | | | | | |
| **Unidades capa ocuta** | | **Pesos entre la capa oculta y la capa de salida** | | | |
| A | | 0,2 | 0,4 | 0,3 | |
| B | | 0,1 | 0,3 | 0,4 | |
| C | | 0,1 | 0,1 | 0,5 | |

El calculo de la red neuronal se ha realizado con Python. Se han escogido como pesos entre la capa oculta y la de salida las dos primeras columnas.

Las salidas de las neuronas de la capa oculta serian 0.598 y 0.689.

A continuación, se copia el código utilizado y las salidas obtenidas

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Jun 26 17:07:54 2020
@author: Barbara
"""
##Red neuronal

import numpy as np
import math
entrada=[4,6,1,5]

#inicializacion de vairables
oculta=np.zeros(3)
salida=np.zeros(2)
y_oculta=np.zeros(3)

#pesos de la capa oculta
w_e_o=np.array([[0.1,0.3,0.2,0.1],
       [0.2,0.1,0.4,0.5],
       [0.5,0.3,0.5,0.2]])

#pesos de la capa de salida
w_o_s=np.array([[0.2,0.1,0.1],
       [0.4,0.3,0.1]])

#Función sigmoide
def sigmoid(x):
  return 1 / (1 + math.exp(-x))



for j in range(len(oculta)): #se itera sobre las neuronas de la capa oculta
    for i in range(len(entrada)): #se itera sobre las neuronas de la capa de
    #entrada
        oculta[j]+=entrada[i]*w_e_o[j,i] #se suma el cociente de la entrada
        #por el peso al de la iteracion anterior para obtener el sumatorio
    y_oculta[j]=math.tanh(oculta[j]) #salida de la capa oculta, con la funcion
    #de activación hiperbolica
    for k in range(len(salida)): #se itera sobre la capa de salida
        salida[k]+=y_oculta[j]*w_o_s[k,j] #se suma el cociente de la salida de
        #La capa oculta por el peso a la iteracion anterior para obtener el
        #sumatorio

sg=np.vectorize(sigmoid) #se vectoriza la función
y=sg(salida) #se calcula la función de activacion de la capa de salida

print(y)
```

| Name | Type | Size | Value |
|---|---|---|---|
| entrada | list | 4 | [4, 6, 1, 5] |
| i | int | 1 | 3 |
| j | int | 1 | 2 |
| k | int | 1 | 1 |
| oculta | Array of float64 | (3,) | [2.9 4.3 5.3] |
| salida | Array of float64 | (2,) | [0.39875084 0.79746984] |
| w_e_o | Array of float64 | (3, 4) | [[0.1 0.3 0.2 0.1]<br>[0.2 0.1 0.4 0.5]] |
| w_o_s | Array of float64 | (2, 3) | [[0.2 0.1 0.1]<br>[0.4 0.3 0.1]] |
| y | Array of float64 | (2,) | [0.5983875 0.689433 ] |
| y_oculta | Array of float64 | (3,) | [0.99396317 0.99963186 0.99995017] |

5. **En la tabla siguiente se especifican los productos (A, B, C y D) que han comprado (marcados con una x) cinco personas. Se pide calcular el soporte, la confianza y el lift para la combinación de productos AB, CD y ABC.**

| | PRODUCTOS | | | |
|---|---|---|---|---|
| | A | B | C | D |
| 1 | | x | x | x |
| 2 | x | | x | |
| 3 | | x | x | x |
| 4 | x | | x | x |
| 5 | x | x | x | x |

No entiendo muy bien si se pide la confianza de A=>B o de AB=>C y AB=D. Entonces voy a hacer las combinaciones posibles con cada uno de las reglas.

Soportes:

Soporte (AB) = P(A∩B)=1/5=20%
Soporte (CD) = P(C∩D)=4/5=80%
Soporte (ABC) = P(A∩B)=1/5=20%

Confianza

C(A=>B)= sop(AB)/Sop(A) *100=33.33
C(B=>A)= sop(AB)/Sop(B) *100=33.33
C(C=>D)= sop(CD)/Sop (C) *100=100
C(D=>C)= sop (CD)/Sop (D) *100=100
C(AB=>C)= sop(ABC)/Sop (AB) *100=100 == C(BA=>C)= sop (ABC)/Sop (BA)
C(AC=>B)= sop(ABC)/Sop (AC) *100=33.33 == C(CA=>B)= sop(ABC)/Sop (CA)
C(BC=>A)= sop(ABC)/Sop (BC) *100=33.33== C(CB=>A)= sop(ABC)/Sop(CB)
Lift

lift(A=>B)= sop(AB)/(Sop(A)*Sop(B)) = 0.56 = Lift(B=>A)
lift(C=>D)= sop(CD)/(Sop(C)*Sop(C)) = 1 = lift(D=>C)
lift(AB=>C)= sop(ABC)/(Sop(AB)*Sop(C)) = 1
lift(AC=>B)= sop(ABC)/(Sop(AC)*Sop(B)) = 0.56
lift(BC=>A)= sop (ABC)/(Sop(BC)*Sop(A)) = 0.56