Ajuste y generalización

1. Distintos métodos de ajuste

Vamos a retomar nuestro problema de estimación de precios de casas y vamos a probar distintos métodos de ajuste. La herramienta `nftool` tiene tres opciones de métodos y siempre usa un criterio de parada basado en que aumente el error en un conjunto de prueba, pero hay más posibilidades.

1. Da al comando `nntool` para entrar en un entorno más general de trabajo con redes
2. En la ventana que se abre darle al botón `Import`
3. Elegir la variable `p` como `Input data` y darle al botón `Import`
4. Elegir la variable `t` como `Target data` y darle al botón `Import`. Después dar al botón `Close`
5. Dar al botón `New`
6. Cambiar en la ventana que se abre `Input data` a `p` y `Target data` a `t`. En `Properties` elegir `Layer 1` y poner entre 3 y 5 procesadores. Después dar al botón `Create`
7. Esto nos crea una primera red, con el algoritmo por defecto (Levenberg-Marquardt). Vamos a cambiar cosas y cada vez que le demos al botón `Create` nos añadirá otra red con esas variaciones. Aquí vamos a jugar con el algoritmo de ajuste (`Training function`) y con la función de error (`Performance function`). Como variantes del algoritmo de ajuste, considera a TRAINBFG (cuasi-Newton), TRAINSCG (gradiente conjugado), TRAINBR (regresión bayesiana), TRAINGDX (descenso gradiente adaptativo), TRAINRP (paso fijo en dirección gradiente); y como variante para la función de error, a MSEREG (error cuadrático más suma cuadrada de pesos). Crea por lo menos 5 redes. Después dar a `Close`
8. En el área de redes marcar `network1` y dar al botón `Open`
9. En la ventana que se abre elegir la pestaña `Train` y poner `Inputs` a `p` y `Targets` a `t`. Luego darle a `Train network`. En este caso el ajuste está con sus opciones por defecto. Siempre analiza, con todos los gráficos que te ofrecen herramienta, el resultado de la red y su fiabilidad.

10. El resultado alcanzado puede depender del punto de partida aleatorio tomado. Elegir la pestaña `Reinitialize weights` y darle al botón `Initialize Weights`. Volver luego a la pestaña `Train` y volver a dar a `Train network`

11. Vamos a probar modificando las opciones del algoritmo (subpestaña `Training parameters`). Recuerda que cada vez tienes que inicializar los pesos. Podemos variar el límite de iteraciones (`epochs`), el error admisible (`goal`) o los incrementos de error admitidos en el conjunto de validación (`max_fail`), que si coincide con el número de iteraciones, no actuará como criterio. Haz por lo menos un par de pruebas.

12. Haz lo mismo con las redes que utilizan los otros algoritmos de ajuste.

2. Generalización

Estos apartados los repites variando la red.

2.1. Estimación de error con bootstrap

La idea es crear datos artificiales, con muestreo con reemplazo de los datos originales y luego verificar el error en la muestra original completa.

Una primera opción, para ti, experto en Matlab, es que lo programes ¡Adelante!

Otra segunda opción, es utilizar las siguientes pistas (puedes ir haciéndolo por partes, o mejor, construir un programa con ellas):

1. Para dividirlo conjuntamente, conviene juntar datos y resultados, por ejemplo:
   `muestra=[p;t];`

2. Una forma de hacer el remuestreo, suponiendo que tienes en `casos` el número de casos, y que quieras hacer `n` extracciones:
   `remues=muestra(:,ceil(rand(1,casos,n)*casos));`

3. Así han quedado columnas sucesivas pegadas de cada extracción. Quizá sea cómodo organizarlo en tres índices, de forma que cada extracción se corresponda con un índice de la tercera dimensión:
   `rem3=reshape(remues,[vars casos n]);`
   donde he supuesto que el número total de variables está en `vars`

4. Ahora se trata de obtener el resultado en cada extracción. Si lo programas, a lo mejor es útil irlo poniendo en un vector. Ten en cuenta que la validación ya la estamos haciendo aparte, así que deja la fracción de prueba a 0. Para programarlo o lo haces tú directamente, o modificas lo que exporta `nftool` al final. En cualquier caso, lo que interesa es obtener una medida de error en la muestra completa, como por ejemplo:
   `msegen=mse(real-salidamodelo);`
   o en Octave:
   `msegen=meansq(real-salidamodelo);`

5. Cuando hayas rellenado u obtenido el vector de errores cuadráticos, hay que compararlos. Podemos obtener un histograma con el comando `hist`

2.2. Estimación de error con validación cruzada

La idea es parecida: se parte la muestra en N trozos y se deja cada vez uno de ellos fuera del ajuste, midiéndose el error en ese trozo que se ha quedado fuera.

Para los que necesitéis pistas de cómo implementarlo, aquí van:

1. Es conveniente barajear la muestra. Una reordenación posible:
   ```
   reord=randperm(casos);
   muesbaraj=muestra(:,reord);
   ```

2. Si queremos hacer un tensor tridimensional con la parte que se queda fuera cada vez (¡atención!: el número de veces debe ser divisor del número de casos), siendo la tercera dimensión cada una de esas veces, propongo:
   ```
   fuera=reshape(muesbaraj,vars,[],N);
   ```

3. Si quieres tener también por separado los bloques que sí se usan para el ajuste, te puede valer algo como:
   ```
   reordfuera=reshape(reord,N,[]);
   ajuste=zeros(vars,casos-size(fuera,2),N);
   for vez=1:N
   quedan=setxor(reord,reordfuera(vez,:));
   ajuste(:,:,vez)=muestra(:,quedan);
   end
   ```

4. Eso es lo que debes usar cada vez, ajustarlo con la parte de ajuste y medir el error con la parte de fuera. Esos errores los vas metiendo en un vector.

5. Podemos ver el histograma, como antes.