Selección de variables

Vamos a ensayar algunos métodos de selección de variables utilizando unos datos de un problema de clasificación. Se trata de predecir si una muestra es cancerígena o no a partir del resultado de espectrometría de masas. Nos dan el espectro caracterizado con 10 000 puntos para cada muestra y una indicación de si es cancerígena o no. Los datos están comprimidos en el archivo adjunto.

Para cargar los datos utilizamos el comando:

```
load arcene;
```

Nos aparecerán las variable `datos`, `salida1`, `prueba` y `salida2`. Los datos son 100 muestras, cada una con sus 10000 puntos de espectro. Puedes intentar una red que trabaje sobre 10 000 variables, pero aquí nos vamos a plantear reducir esa cantidad. Las categorías correspondientes (sí cáncer o no cáncer) están en `salida1`. Cuando creamos que hemos hecho una selección de variables buena, utilizaremos `prueba` y `salida2`, que son otras 100 muestras, y ... ¡a ver qué pasa!

Una primera cosa que podemos hacer es quitar las variables que sean claramente irrelevantes. Desde un punto de vista no lineal, estamos hablando de información mutua. Podemos comparar las informaciones mutuas entre cada variable de entrada y la de salida y ver si hay alguna claramente inferior. En la toolbox de información mutua hay una función que calcula el valor que nos interesa. Sí por ejemplo queremos calcular la información mutua entre la j-ésima variable y la salida en el conjunto de ajuste sería:

```
MutualI1(datos(:,j),salida1)
```

La toolbox de estadística tiene algunos algoritmos para crear conjuntos de variables de dimensión reducida. Aparte usaremos otra toolbox específica: "*Dimensionality Reduction*"; tiene muchos, y alguna función auxiliar facilitadora

¡Ojo, cuidado! Con 100 puntos, sacar conclusiones por encima de 100 dimensiones no tiene sentido.

Proyección en componentes principales

Es un método lineal, pero suele ayudar bastante. Recuerda que sólo estudia la representación de los datos, sin tener en cuenta la variable de salida. Tenemos varios comandos para ello. El de la estadística es el que da más información, pero no es el más fácil de aplicar, así que miraremos la información en uno y después usaremos el otro.

1. Obtengamos los 10 000 ejes ortogonales ordenados por varianza de mayor a menor. En realidad no serán 10 000, porque no puede calcular más de 100 si sólo tiene 100 puntos.
   `[coorprin,nuevaentrada,varianza]=princomp(datos);`
   Esto nos da las coordenadas de los versores principales, las coordenadas de los puntos originales en esos ejes y la varianza en cada eje.
2. Para comparar varianzas mejor, vamos a ponerlas en tanto por uno relativo al total:
   `porunovarianza=varianza/sum(varianza);`
3. La idea es coger unos cuantos que tengan claramente más varianza que los demás y que entre todos sumen una fracción muy alta de la varianza total. Si queremos saber, por ejemplo, cuánta varianza acumulan los siete primeros, sería:

   ```
   sum(porunovarianza(1:7))
   ```
4. Con la fracción que elijamos podemos preparar las variables para una red. Lo que nos interesa es cuántos componentes queremos. Si por ejemplo queremos coger hasta el 15, sería:

   ```
   [entradared,transf]=compute_mapping(datos,'PCA',15);
   ```

```
entradapru=out_of_sample(prueba,transf);
```
Hay que tener en cuenta que estas variables la toolbox de redes las coge habitualmente traspuestas. La trasposición se puede hacer desde dentro de la herramienta gráfica.

5. Con esto podemos componer la red neuronal y probar, como hemos hecho en otras prácticas, salvo el detalle de que esta vez estamos haciendo clasificación, no regresión, con lo cual algunos ajustes son mejores de otra forma.
    - En Matlab ir a `nnstart`, elegir "*pattern recognition*" y proceder a crear y probar una red. Si queremos, podemos poner a 0 la fracción de prueba, porque ya tenemos otro conjunto aparte para eso. Cuando esté ajustada, en la pantalla siguiente podemos elegir las variables que tenemos preparadas y probar con ellas la selección y la red. Podemos sacar la matriz de confusión: ¿qué tal sale?
    - En Octave conviene que la capa de salida sea logística ('logistic'), en vez de lineal

Proyección de Sammon sólo con las entradas

Esta es una proyección no lineal y originalmente no supervisada. El uso más fácil es con la toolbox especializada en reducción de dimensión.

En esta proyección se buscan directamente los puntos proyectados, no la transformación, así que para aplicarla a los puntos de prueba hay que hacer una aproximación, que es suponerla lineal localmente. Esta toolbox ya trae un comando para ello.

1. Tenemos que elegir en cuántas dimensiones proyectamos. Hay tres opciones:
    - Ir probando. Puede resultar pesado.
    - Elegir la misma cantidad que ha salido en el análisis anterior.
    - Elegir con base a una estimación de dimensión independiente.

      Si queremos esto último, el comando es (con uno de los métodos que trae la toolbox):

```
ndims = round(intrinsic_dim(datos, 'MLE'));
```
      No tiene mucho sentido aceptar dimensiones por encima del número de puntos y además puede liar a los algoritmos. En vez de MLE puedes probar GMST
2. Hacemos la proyección:
```
nuevasent2 = compute_mapping(datos, 'Sammon', ndims);
```
3. Estimamos las coordenadas del conjunto de prueba:
```
nuevaspru2=out_of_sample_est(prueba,datos,nuevasent2);
```
4. Nos construimos la red, la ajustamos con `nuevasent2` y la probamos con `nuevaspru2` ¿...y?

Proyección SNE

Este es otro caso de proyección no lineal basada exclusivamente en los datos. Se parece bastante al anterior, pero podemos probarlo. Sólo está en la toolbox de reducción de dimensión. Es todo igual que en el caso anterior pero cambiando *Sammon* por *t-SNE*

Proyección Sammon orientada a la tarea

Hay una opción para hacer la proyección Sammon cogiendo las distancias entre puntos, en lugar de las coordenadas o valores de las variables. La ventaja es que podemos incorporar información de

las salidas que pretendemos obtener. No podemos usar sólo ésta última, porque se supone que en el futuro no la sabremos y que para estimar las nuevas coordenadas tendremos sólo las variables de entrada, así que tenemos que dejarlas un papel preponderante.

1. Empezamos obteniendo las distancias entre las entradas:

```
dif=pdist(datos);
```

2. Para poder trabajar con una escala uniforme, las normalizamos dividiendo por la máxima:

```
dif=dif/max(dif);
```

3. Obtenemos la matriz de distancias entre las salidas:

```
clasefilas=repmat(salida1',100,1);
clasecols=repmat(salida1,1,100);
difclase=abs(clasefilas-clasecols);
```

4. Tenemos que pensar en una ponderación de la distancia en salidas respecto a las entradas, para generar la matriz de distancia total. No puede ser un factor alto, pero si es muy bajo es como si estuviéramos en el caso anterior. Suponiendo que fuese 0.15 por ejemplo, sería:

```
dist=squareform(dif)+0.15*difclase;
```

5. Elegimos el número de dimensiones y aplicamos. Nuevamente, la transformación en el conjunto de prueba la hacemos por estimación:

```
opc=struct('Input','distance','Display','on',...
'Initialisation','random','MaxIter',300,'MaxHalves',20,...
'TolFun',1e-9);
nuevasent4 = compute_mapping(dist, 'Sammon', ndims,opc);
nuevaspru4=out_of_sample_est(prueba,datos,nuevasent4);
```

6. Y a pasar la red. Quizá quieras iterar cambiando el ratio entre las distancias de entradas y de salida.