Interface para modelado de redes neuronales

Roldán Galán Lafarga
rolgalan@gmail.com

Huesca - 19 de noviembre de 2008

Contenido

1. Qué
2. Por qué
   - MLP
   - Entorno R
   - Aplicación
3. Cómo
   - Herramientas
   - Sintetización de la arquitectura
   - Descripción de las clases principales
   - Selección de la mejor red
4. Ampliación

Descripción del proyecto

La aplicación pretende facilitar el entrenamiento de la red Perceptrón Multicapa para problemas de clasificación en el entorno **R** mediante el desarrollo de una interfaz gráfica, cálculos efectuados por medio de computación distribuida y la selección automática del mejor modelo utilizando el valor del área bajo la curva ROC.

Contenido

1 Qué

2 Por qué
- MLP
- Entorno R
- Aplicación

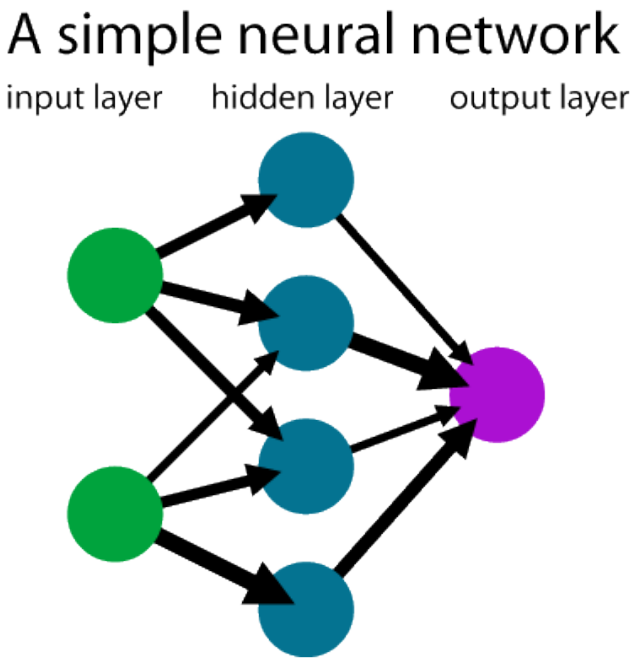
3 Cómo
- Herramientas
- Sintetización de la arquitectura
- Descripción de las clases principales
- Selección de la mejor red

4 Ampliación

Redes Neuronales

Características

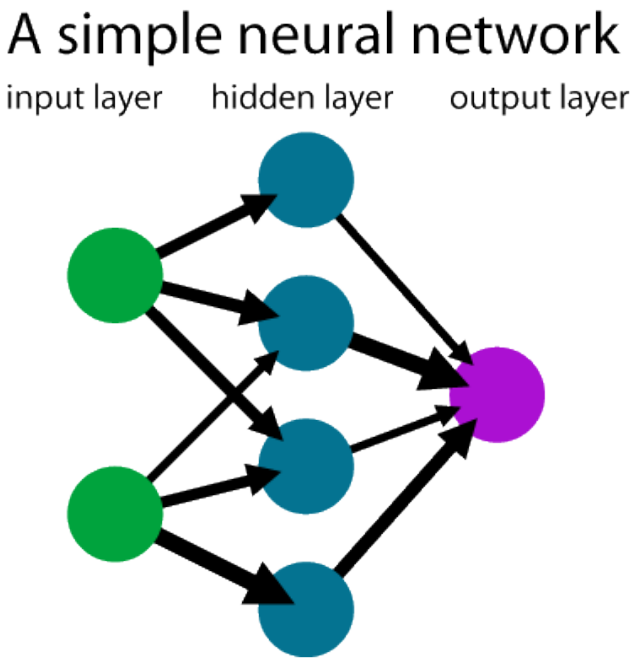
- Su unidad básica es la neurona, compuesta de una serie de entradas con un peso asociado, una función de activación y una salida.
- Todas las neuronas se conectan entre sí para devolver el resultado buscado.
- Pueden considerarse como una serie de funciones no lineales combinadas para optimizar un objetivo.



Redes Neuronales

Características

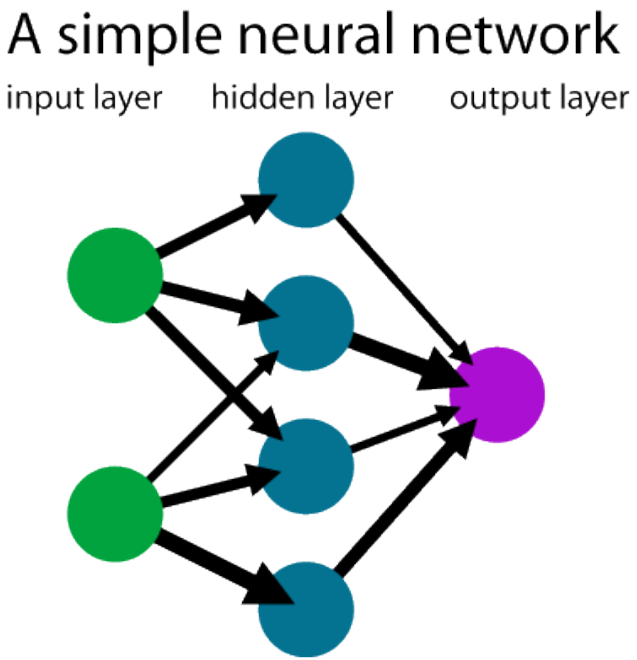
- Su unidad básica es la neurona, compuesta de una serie de entradas con un peso asociado, una función de activación y una salida.
- Todas las neuronas se conectan entre sí para devolver el resultado buscado.
- Pueden considerarse como una serie de funciones no lineales combinadas para optimizar un objetivo.



Redes Neuronales

Características

- Su unidad básica es la neurona, compuesta de una serie de entradas con un peso asociado, una función de activación y una salida.
- Todas las neuronas se conectan entre sí para devolver el resultado buscado.
- Pueden considerarse como una serie de funciones no lineales combinadas para optimizar un objetivo.



Perceptrón Multicapa

- Las neuronas se distribuyen en **capas**.
- La capa inicial —de **entrada**— se corresponde a los datos: tendrá tantas neuronas como información distinta de cada elemento a estudiar.
- La capa final —de **salida**— consta de una única neurona, con valor entre 0 y 1: revela bajo qué categoría se ha clasificado el dato analizado.
- El resto de capas intermedias se les denomina capas **ocultas**, y unen la capa de entrada con la de salida.
- La conexión se realiza entre las neuronas de las **capas anterior y posterior**.

Perceptrón Multicapa

- Las neuronas se distribuyen en **capas**.
- La capa inicial —de **entrada**— se corresponde a los datos: tendrá tantas neuronas como información distinta de cada elemento a estudiar.
- La capa final —de **salida**— consta de una única neurona, con valor entre 0 y 1: revela bajo qué categoría se ha clasificado el dato analizado.
- El resto de capas intermedias se les denomina capas **ocultas**, y unen la capa de entrada con la de salida.
- La conexión se realiza entre las neuronas de las **capas anterior y posterior**.

Perceptrón Multicapa

- Las neuronas se distribuyen en **capas**.
- La capa inicial —de **entrada**— se corresponde a los datos: tendrá tantas neuronas como información distinta de cada elemento a estudiar.
- La capa final —de **salida**— consta de una única neurona, con valor entre 0 y 1: revela bajo qué categoría se ha clasificado el dato analizado.
- El resto de capas intermedias se les denomina capas **ocultas**, y unen la capa de entrada con la de salida.
- La conexión se realiza entre las neuronas de las **capas anterior y posterior**.

Perceptrón Multicapa

- Las neuronas se distribuyen en **capas**.
- La capa inicial —de **entrada**— se corresponde a los datos: tendrá tantas neuronas como información distinta de cada elemento a estudiar.
- La capa final —de **salida**— consta de una única neurona, con valor entre 0 y 1: revela bajo qué categoría se ha clasificado el dato analizado.
- El resto de capas intermedias se les denomina capas **ocultas**, y unen la capa de entrada con la de salida.
- La conexión se realiza entre las neuronas de las **capas anterior y posterior**.

Perceptrón Multicapa

- Las neuronas se distribuyen en **capas**.
- La capa inicial —de **entrada**— se corresponde a los datos: tendrá tantas neuronas como información distinta de cada elemento a estudiar.
- La capa final —de **salida**— consta de una única neurona, con valor entre 0 y 1: revela bajo qué categoría se ha clasificado el dato analizado.
- El resto de capas intermedias se les denomina capas **ocultas**, y unen la capa de entrada con la de salida.
- La conexión se realiza entre las neuronas de las **capas anterior y posterior**.

Parámetros

- Número de capas
- Cantidad de neuronas en cada capa
- Pesos iniciales de cada neurona
- Distorsión inicial de cada neurona
- Tasa de aprendizaje del algoritmo de retropropagación
- Número máximo de iteraciones para el entrenamiento
- Máxima diferencia entre la salida deseada y la que se espera obtener
- Funciones de activación de cada capa
  - Función logística
  - Función tangente hiperbólica
  - Función gaussiana
  - Función identidad

Contenido

1. Qué
2. Por qué
   - MLP
   - Entorno R
   - Aplicación
3. Cómo
   - Herramientas
   - Sintetización de la arquitectura
   - Descripción de las clases principales
   - Selección de la mejor red
4. Ampliación

¿Qué es?

- Es un entorno matemático orientado al cálculo estadístico (en el cual se basan fundamentalmente las redes neuronales).
- Cuenta con el paquete *neural* para generar redes MLP (que también soporta basadas funciones de base radial).
- Dispone del paquete *ROCR* para elaborar la curva *Receiver Operating Characteristic*.
- No dispone de interfaz gráfico, se introducen las órdenes mediante código *script*:

  Ventaja permite una gran personalización y mayor funcionalidad.

  Inconveniente tener que escribir muchos comandos distintos puede ser tedioso e incómodo para el usuario.

Entrenamiento de redes bajo R

No es inmediato: tardan un tiempo nada despreciable, que se incrementa con el número de neuronas, de entradas y el total de datos.
Es necesario un comando distinto para cada red de diferentes parámetros que se desee entrenar.

Comando de ejemplo

```
mlptrain(inp,neurons,out,weight=c(),dist=c(),
alfa=0.2,it=200,online=TRUE, permute=TRUE,thresh=0,
dthresh=0.1,actfns=c(),diffact=c(),visual=TRUE)
```

Entrenamiento de redes bajo R

No es inmediato: tardan un tiempo nada despreciable, que se incrementa con el número de neuronas, de entradas y el total de datos.
Es necesario un comando distinto para cada red de diferentes parámetros que se desee entrenar.

Comando de ejemplo

```
mlptrain(inp,neurons,out,weight=c(),dist=c(),
alfa=0.2,it=200,online=TRUE, permute=TRUE,thresh=0,
dthresh=0.1,actfns=c(),diffact=c(),visual=TRUE)
```

Contenido

1. Qué
2. Por qué
   - MLP
   - Entorno R
   - Aplicación
3. Cómo
   - Herramientas
   - Sintetización de la arquitectura
   - Descripción de las clases principales
   - Selección de la mejor red
4. Ampliación

Actividades esenciales

- Interfaz gráfica para facilitar la selección de los parámetros que definirán las redes.
- Generación automática de distintas redes en base a los parámetros.
- Computación distribuida para el entrenamiento de los diferentes modelos deseados.
- Selección automática de la red más eficiente mediante el valor AUC.
- Muestra por pantalla de los parámetros significativos del mejor modelo.

Contenido

1. Qué
2. Por qué
   - MLP
   - Entorno R
   - Aplicación
3. Cómo
   - Herramientas
   - Sintetización de la arquitectura
   - Descripción de las clases principales
   - Selección de la mejor red
4. Ampliación

Programas utilizados

- **Java:** Interfaz gráfica de todas las partes, generación y distribución de comandos.
- **R:** Cálculos de las redes neuronales y del área bajo la curva *ROC* para seleccionar la mejor.
- **JRI:** *Interface* para realizar la comunicación entre **Java** y **R**.

Contenido

1. Qué
2. Por qué
   - MLP
   - Entorno R
   - Aplicación
3. Cómo
   - Herramientas
   - Sintetización de la arquitectura
   - Descripción de las clases principales
   - Selección de la mejor red
4. Ampliación

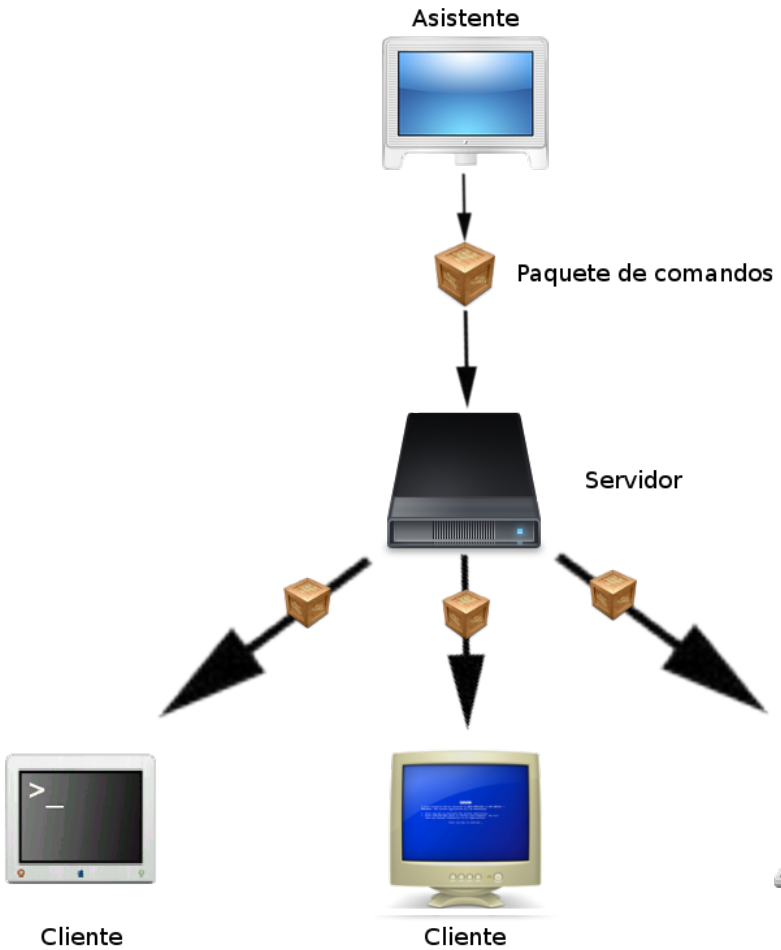
Vistazo General

Asistente/Servidor

Selección/Generación de las redes a entrenar y gestión del proceso de entrenamiento.

Cliente

Cálculo de las redes solicitadas por el Asistente/Servidor.



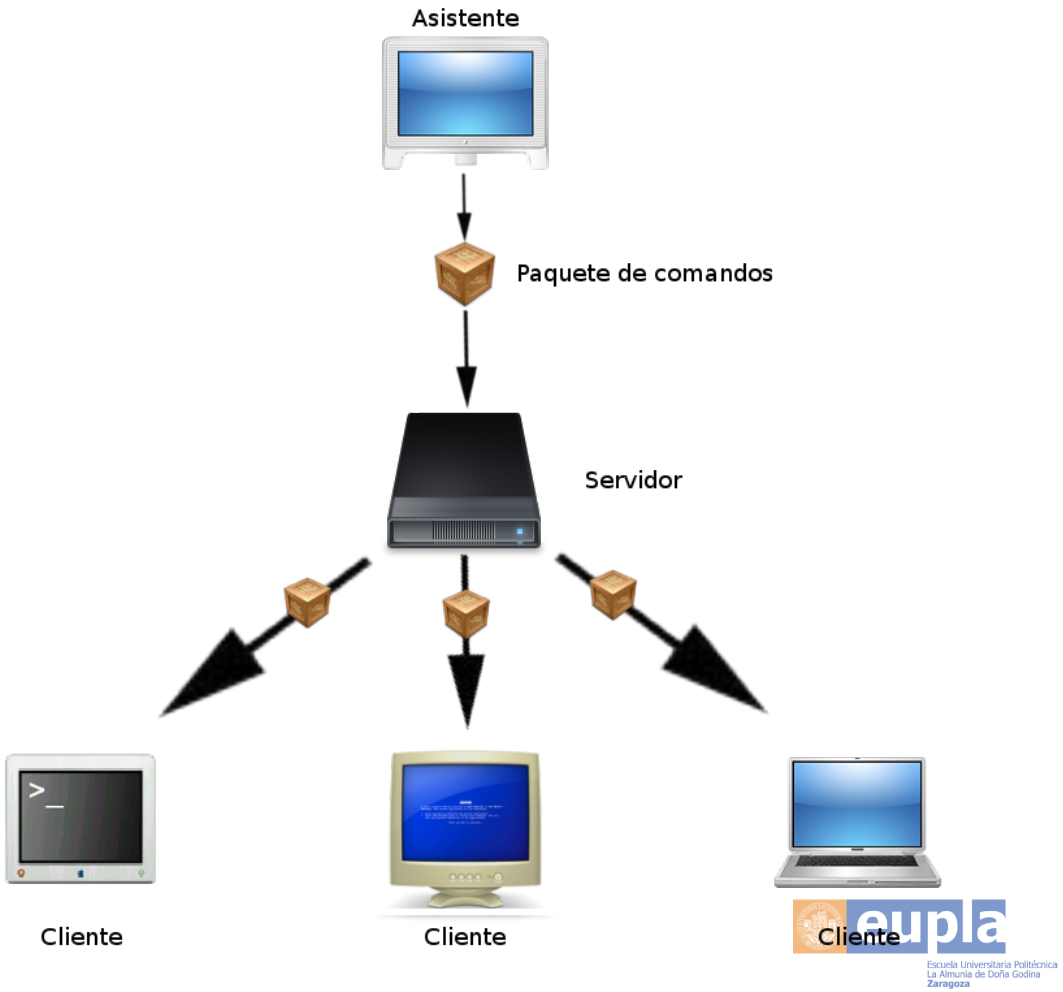
Vistazo General

Asistente/Servidor

Selección/Generación de las redes a entrenar y gestión del proceso de entrenamiento.

Cliente

Cálculo de las redes solicitadas por el Asistente/Servidor.



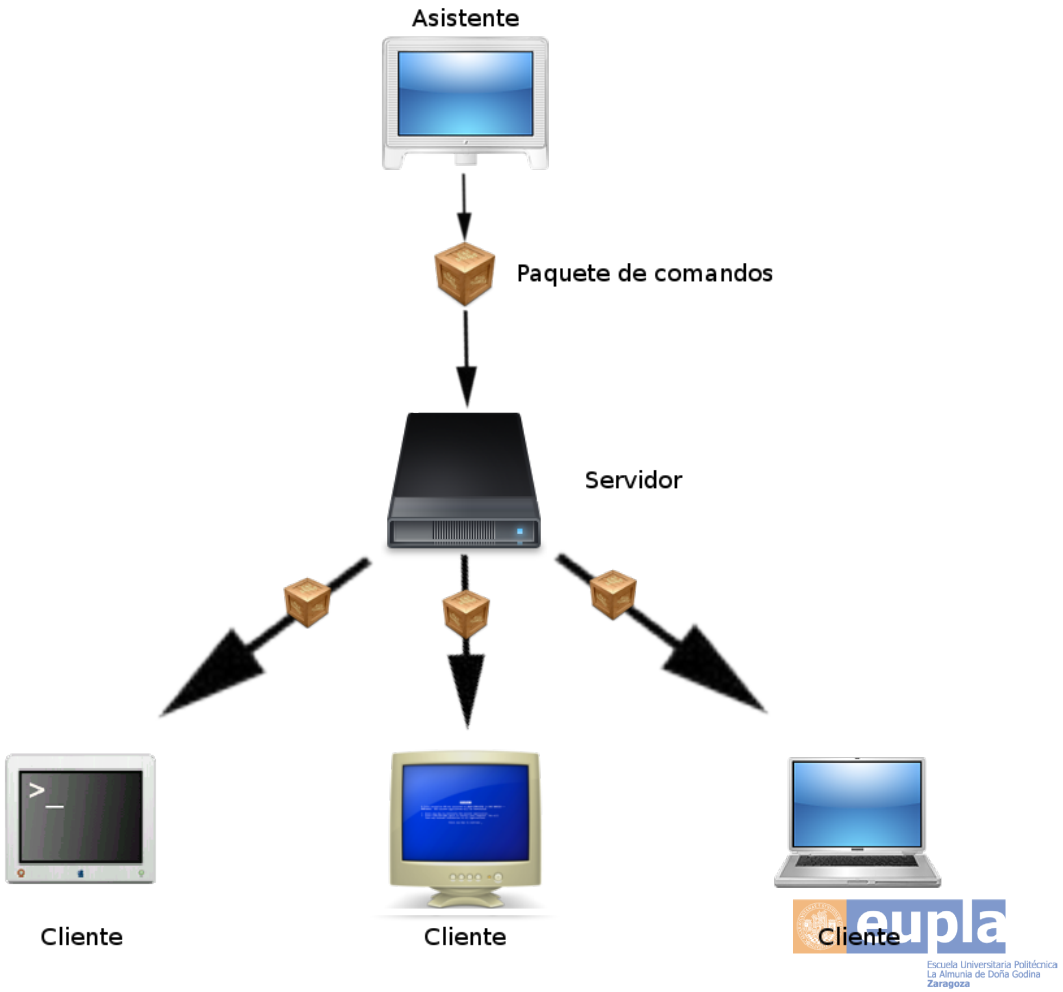
Vistazo General

Asistente/Servidor

Selección/Generación de las redes a entrenar y gestión del proceso de entrenamiento.

Cliente

Cálculo de las redes solicitadas por el Asistente/Servidor.



Asistente

Interfaz gráfico mediante la que se seleccionan los parámetros que definirán las redes MLP a entrenar.

1. Dos procesos distintos para generar las redes: **básico** y **asistido**.
2. Asignación del fichero de datos necesario para el entrenamiento.
3. *GeneradorRedesMLP* extrae toda información de la vista y genera los comandos de **R** para entrenar las redes y seleccionar la mejor.
4. Todo este paquete de comandos se envía al Servidor.
5. Muestra la mejor red que le devuelve el Servidor después de todo el entrenamiento.

Asistente

Interfaz gráfico mediante la que se seleccionan los parámetros que definirán las redes MLP a entrenar.

1. Dos procesos distintos para generar las redes: **básico** y **asistido**.
2. Asignación del fichero de datos necesario para el entrenamiento.
3. *GeneradorRedesMLP* extrae toda información de la vista y genera los comandos de **R** para entrenar las redes y seleccionar la mejor.
4. Todo este paquete de comandos se envía al Servidor.
5. Muestra la mejor red que le devuelve el Servidor después de todo el entrenamiento.

Asistente

Interfaz gráfico mediante la que se seleccionan los parámetros que definirán las redes MLP a entrenar.

1. Dos procesos distintos para generar las redes: **básico** y **asistido**.
2. Asignación del fichero de datos necesario para el entrenamiento.
3. *GeneradorRedesMLP* extrae toda información de la vista y genera los comandos de **R** para entrenar las redes y seleccionar la mejor.
4. Todo este paquete de comandos se envía al Servidor.
5. Muestra la mejor red que le devuelve el Servidor después de todo el entrenamiento.

Asistente

Interfaz gráfico mediante la que se seleccionan los parámetros que definirán las redes MLP a entrenar.

1. Dos procesos distintos para generar las redes: **básico** y **asistido**.
2. Asignación del fichero de datos necesario para el entrenamiento.
3. *GeneradorRedesMLP* extrae toda información de la vista y genera los comandos de **R** para entrenar las redes y seleccionar la mejor.
4. Todo este paquete de comandos se envía al Servidor.
5. Muestra la mejor red que le devuelve el Servidor después de todo el entrenamiento.

Asistente

Interfaz gráfico mediante la que se seleccionan los parámetros que definirán las redes MLP a entrenar.

1. Dos procesos distintos para generar las redes: **básico** y **asistido**.
2. Asignación del fichero de datos necesario para el entrenamiento.
3. *GeneradorRedesMLP* extrae toda información de la vista y genera los comandos de **R** para entrenar las redes y seleccionar la mejor.
4. Todo este paquete de comandos se envía al Servidor.
5. Muestra la mejor red que le devuelve el Servidor después de todo el entrenamiento.

Servidor

Realiza, mediante sockets, la conexión con todos los clientes y gestiona que se realice el trabajo.

- Comunicación con los clientes enviando objetos de la clase ObjetoEnvio que encapsulan la información que se quieren transmitir.
- Busca clientes que no estén realizando cálculos y les envía el paquete hasta que se calculan todos los modelos solicitados
- Recibe de los Clientes la mejor red de cada paquete, compara todas las obtenidas y devuelve la mejor al Asistente, que la muestra por pantalla.

Servidor

Realiza, mediante sockets, la conexión con todos los clientes y gestiona que se realice el trabajo.

- Comunicación con los clientes enviando objetos de la clase ObjetoEnvio que encapsulan la información que se quieren transmitir.
- Busca clientes que no estén realizando cálculos y les envía el paquete hasta que se calculan todos los modelos solicitados
- Recibe de los Clientes la mejor red de cada paquete, compara todas las obtenidas y devuelve la mejor al Asistente, que la muestra por pantalla.

Cliente

Unidad individual de cálculo.

- Se conecta por sockets con el Servidor y está a la escucha permanente de envíos del Servidor.
- Cuando recibe comandos para entrenar redes instancia un hilo para realizar los cálculos mientras sigue escuchando por si el Servidor envía algo más.
- Una vez termina el entrenamiento, utiliza una clase ExtraeRed para tomar la información relevante de la variable de **R** a la que se ha asignado la red más eficiente.
- Encapsula esta red en un ObjetoEnvio y se lo devuelve al servidor, junto a la información del resto de áreas entrenadas.

Cliente

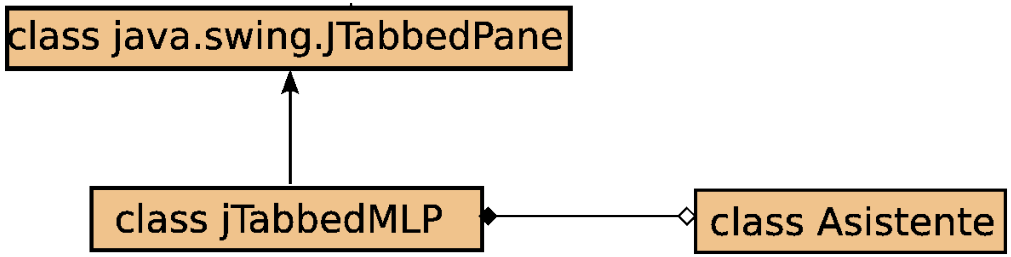
Unidad individual de cálculo.

- Se conecta por sockets con el Servidor y está a la escucha permanente de envíos del Servidor.
- Cuando recibe comandos para entrenar redes instancia un hilo para realizar los cálculos mientras sigue escuchando por si el Servidor envía algo más.
- Una vez termina el entrenamiento, utiliza una clase ExtraeRed para tomar la información relevante de la variable de **R** a la que se ha asignado la red más eficiente.
- Encapsula esta red en un ObjetoEnvio y se lo devuelve al servidor, junto a la información del resto de áreas entrenadas.

Contenido

1. Qué
2. Por qué
   - MLP
   - Entorno R
   - Aplicación
3. Cómo
   - Herramientas
   - Sintetización de la arquitectura
   - Descripción de las clases principales
   - Selección de la mejor red
4. Ampliación

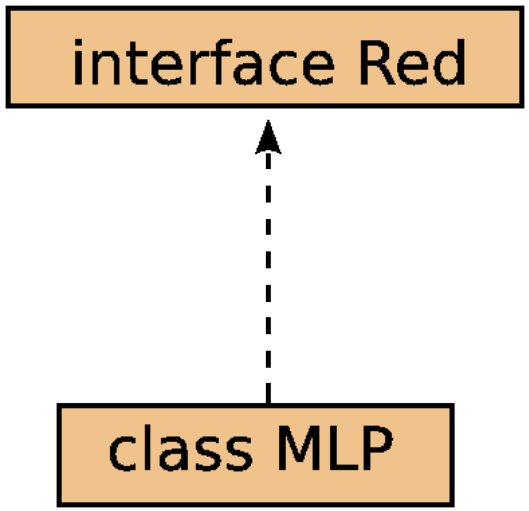
Asistente



- El programa se lanza mediante la clase *Asistente*, que es un *JFrame* con menús para seleccionar algunas opciones y un *JTabbedPane* ocupando el resto de la ventana.
- Cuando seleccionamos que queremos crear redes **MLP**, se asigna un *JTabbedMLP* con toda la interfaz gráfica necesaria para el desarrollo de redes **MLP**.

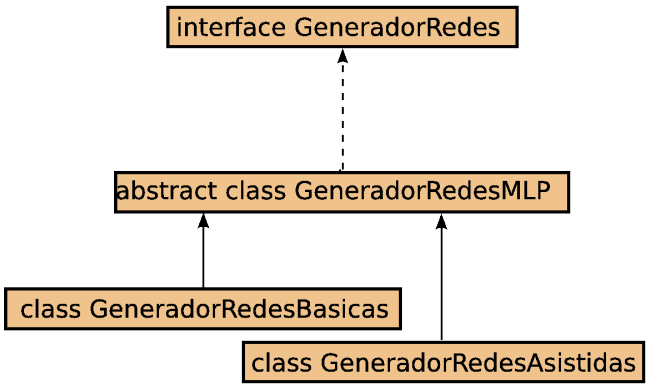
Red

- El modelo de dato con el que trabaja la aplicación son clases que implementan la interface **Red**.
- En este objeto se encapsulan los campos que conforman una red en el entorno **R**.
- Mayoría de parámetros compartidos, por lo que este objeto representa tanto las redes que se van a entrenar, como las entrenadas



Por ahora la aplicación sólo soporta las redes **MLP**

Generadores de Red

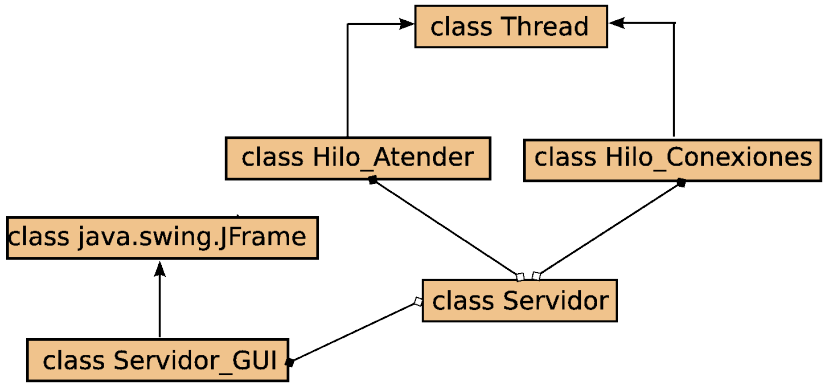


La interface **GeneradorRedes** es el controlador del Asistente. Extrae la información que el usuario introduce en la Vista y la convierte en objetos **Red**.

- **GeneradorRedesMLP** genera objetos **MLP** a partir de la información del **JTabbedMLP**. Crea un objeto **FicheroCSV** que encapsula el fichero de datos.
- GeneradorRedesMLP contiene métodos comunes que utilizan sus dos clases hijas **GeneradorRedesBasicas** y **GeneradorRedesAsistidas**, necesarias por los dos modos de diseño de redes implementados.

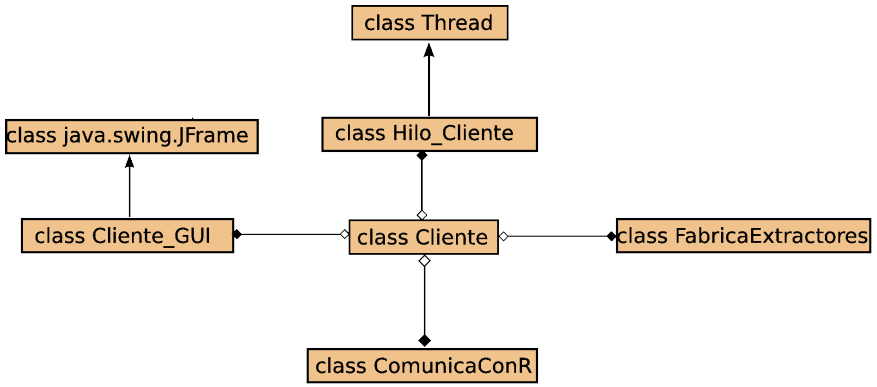
Servidor

La clase en la que se delegan las comunicaciones es **Servidor**, cuya *Vista* es la clase **Servidor_Gui**.



- Al Servidor se le pasa un objeto **GeneradorRedes**, del cual extrae los comandos y el fichero que debe enviar a los Clientes.
- Para estar escuchando constantemente cualquier petición de conexión, utiliza el **Hilo_Conexiones**.
- Cada vez que se acepta una conexión de un *Cliente*, se instancia un **Hilo_Atender**, para realizar la comunciación individual.

Cliente



- Quién realiza la conexión del lado de las máquinas de cálculo, es la clase **Cliente**
- Su interfaz gráfica recae en la clase **Cliente_Gui**.
- Para atender los mensajes enviados por el *Servidor*, se instancia un **Hilo_Cliente**.

Contenido

1. Qué
2. Por qué
   - MLP
   - Entorno R
   - Aplicación
3. Cómo
   - Herramientas
   - Sintetización de la arquitectura
   - Descripción de las clases principales
   - Selección de la mejor red
4. Ampliación

Criterio de calidad

- Entrenar más de una red requiere definir después cuál obtiene los mejores resultados.
- Al realizarse un entrenamiento masivo es necesario automatizar la selección de la mejor red.
- El paquete *neural* de **R** no proporciona ninguna información sobre la calidad de las salidas.
- Hay varios métodos para medir la precisión de la red. En este caso se ha escogido el área bajo la curva **ROC**.

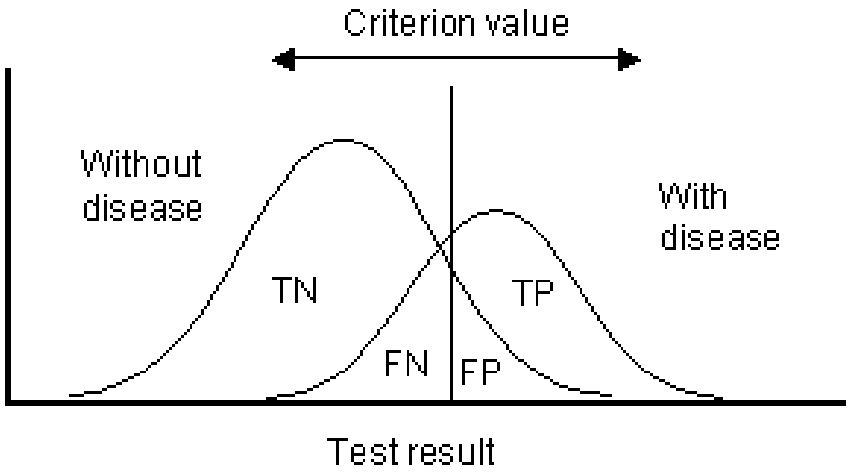
¿Qué es la Curva ROC?

Receiver Operating Characteristic Curve

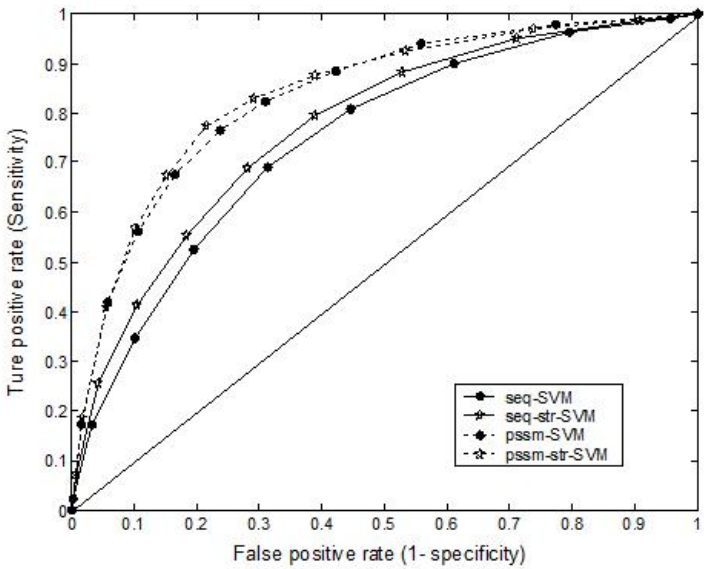
- Método estadístico muy eficiente para evaluar la eficiencia del punto de corte en una clasificación dicotómica.
- Ha sido aplicada ampliamente en medicina por radiólogos para determinar la posibilidad de enfermedad en un paciente, en un caso muy similar a nuestra búsqueda de clasificar datos en dos categorías (enfermo o sano).
- En la actualidad está empezando a introducirse, también, en áreas informáticas como la Inteligencia Artificial o la Minería de Datos.

¿Por qué es útil?

- Un modelo de clasificación como el MLP asigna un real entre 0 y 1 a cada salida.
- Otros criterios requieren tomar un valor de corte significativo que permita separar ambos resultados de la forma más conveniente posible.
- Esto no es trivial puesto que, por lo general, ambas poblaciones se solapan.



¿Cómo se construye?



La construcción de la curva **ROC** se basa en los conceptos de:

Sensibilidad probabilidad de catalogar correctamente un dato positivo.

Especificidad probabilidad de clasificar correctamente un dato negativo (1 – fracción de falsos positivos).

Área bajo la curva

- El modelo que discrimina perfectamente todos los datos es aquel con *sensibilidad* $= 1 =$ *especificidad*.
- El área por debajo de esta curva será igual a uno.
- Si la curva sube en diagonal, tendremos un *area* $= 0{,}5$, lo cual significará que la cantidad de aciertos y fallos es la misma.
- Así pues, la calidad de la red es proporcional al valor de la curva.
- Si el área es menor de 0.5 se debe cambiar el objetivo principal.

Escalabilidad

Se ha intentado, en todo lo posible, conseguir un programa fácilmente ampliable, que permita escalabilidad de forma sencilla. Esto se logra mediante la programación a partir de *interfaces* y clases abstractas.

Versatilidad

De hecho la *interface* **Red** podría representar, en realidad, cualquier objeto de **R**; y mediante la clase abstracta **ExtraeRedes** se podría toma la información necesaria. Todo ello sin modificar la estructura del programa.

Escalabilidad

Se ha intentado, en todo lo posible, conseguir un programa fácilmente ampliable, que permita escalabilidad de forma sencilla. Esto se logra mediante la programación a partir de *interfaces* y clases abstractas.

Versatilidad

De hecho la *interface* **Red** podría representar, en realidad, cualquier objeto de **R**; y mediante la clase abstracta **ExtraeRedes** se podría toma la información necesaria. Todo ello sin modificar la estructura del programa.

Modelo de datos

La *interface* **redes.Red** se comporta como modelo de datos. Deberá incluir todos los campos que definan el tipo de redes con el que se pretenda trabajar. Habrá que añadir a la *interface* **Red** un campo entero asignando un número distinto al siguiente modelo de red a crear.

El programa implementa esta interface con la clase **redes.redMLP.MLP**.

Interfaz Gráfica

La clase *asistente.Asistente* que envuelve la parte gráfica de la aplicación contiene un *javax.swing.JTabbedPane*. Se puede extender una clase de éste para ser la Vista de un nuevo modelo de redes.

Actualmente sólo existe **redes.redMLP.vista.JTabbedMLP** ejerciendo esta labor.

La interface **redes.GeneradorRedes** extrae la información introducida por el usuario en la vista y genera los comandos que entrenan las redes y calculan la mejor. Lo inserta en un array de *String* que devolverá cuando se solicite el método *paqueteComandos()*.

Al **Servidor** se le envía el objeto **GeneradorRedes**, del cual tomará el paquete de comandos para enviárselo a tantos clientes como se indique.

La clase abstracta **redes.redMLP.GeneradorRedesMLP** implementa esta interface con algunos métodos necesarios. De ella extienden **GeneradorRedesBasicas** y **GeneradorRedesAsistidas** para los dos modos de creación de redes.

La interface **redes.GeneradorRedes** extrae la información introducida por el usuario en la vista y genera los comandos que entrenan las redes y calculan la mejor. Lo inserta en un array de *String* que devolverá cuando se solicite el método *paqueteComandos()*.

Al **Servidor** se le envía el objeto **GeneradorRedes**, del cual tomará el paquete de comandos para enviárselo a tantos clientes como se indique.

La clase abstracta **redes.redMLP.GeneradorRedesMLP** implementa esta interface con algunos métodos necesarios. De ella extienden **GeneradorRedesBasicas** y **GeneradorRedesAsistidas** para los dos modos de creación de redes.

La interface **redes.GeneradorRedes** extrae la información introducida por el usuario en la vista y genera los comandos que entrenan las redes y calculan la mejor. Lo inserta en un array de *String* que devolverá cuando se solicite el método *paqueteComandos()*.

Al **Servidor** se le envía el objeto **GeneradorRedes**, del cual tomará el paquete de comandos para enviárselo a tantos clientes como se indique.

La clase abstracta **redes.redMLP.GeneradorRedesMLP** implementa esta interface con algunos métodos necesarios. De ella extienden **GeneradorRedesBasicas** y **GeneradorRedesAsistidas** para los dos modos de creación de redes.

Cliente

Se debe extender la clase abstracta **comR.ExtractorRed** para tomar los datos que conforman la red en **R** y convertirla en el objeto **Red** que la representa en Java.
La clase **comR.FabricaExtractores** devuelve el **ExtractorRed** correspondiente a la Red solicitada: se deben añadir las nuevas clases que extienden de ExtractorRed.

En nuestro modelo etendemos un **redes.redMLP.ExtractorRedMLP** para realizar esta tarea.