{Texto informe}

Redes Neuronales - TP1 - MLP

Alvarez - Bordón - Santos

Septiembre 2016

Resumen

Se diseñaron modelos de perceptrón multicapa (MLP) para los dos ejercicios propuestos. En ambos casos se utilizaron redes con una única capa oculta, y el algoritmo de retropropagación para su entrenamiento. Distintas configuraciones de parámetros y neuronas fueron testeados. Se presentan los resultados obtenidos para los distintos casos.

1 . Modelos MLP aplicados al diagnóstico de cáncer de mama

{texto diego}

2. Modelos MLP aplicados al análisis de eficiencia energética en edificios

{Voy a separar las secciones de manera idéntica a la de diego para ser consistentes

1.1 introducción

1.2 análisis de la red (((o “análisis en primera instancia” o “primera arquitectura” o similar)))

1.3 preprocesamiento de datos

1.4 arquitectura de la red

1.5 implementación de la red

1.6 experimentación y resultados

1.7 conclusiones

}

2.1 Introducción

El objetivo en este caso consistió en predecir el valor de carga energética necesaria para la calefacción y refrigeración de edificios a partir de ciertas características de los mismos. El conjunto de datos contaba con valores sobre 8 características distintas a tener en cuenta y se debió entrenar al sistema a partir de 500 resultados dados tanto para calefacción como para refrigeración.

2.2 Análisis de la red

En primera instancia utilizamos una arquitectura similar a la implementada en la primera parte del trabajo. Una capa de entrada de 8 + 1 neuronas, una única capa oculta de número de neuronas variables entre 5 y 8, función de activación sigmoide de tipo bipolar (tanh(βX)) y en este caso 2 neuronas en la capa de salida de salida, una para la carga de refrigeración y otra para la carga de calefacción. Contrario a lo que esperábamos la red no lograba aprender, obteniendo valores de salida para las cargas de refrigeración y calentamiento poco correlacionados con los de los datos de entrenamiento, no logrando bajar el error de entrenamiento. Debido a esto, lo siguiente fue empezar a cambiar la arquitectura de la red, variando los parámetros de activación, taza de aprendizaje y número de neuronas, sin obtener mejoras significativas.

Pasamos a cambiar la función de activación, optando por una lineal (f(X)=x), obteniendo valores de capa de salida más acorde a lo esperado, con lo que se utilizó este tipo de función para la arquitectura final de la res (justificación de por qué sucede eso?)

2.3 Procesamiento de datos

De manera análoga al caso anterior, en vista de que las medias y varianzas de unos y otros atributos difieren significativamente y hacen que ciertos dominen en detrimento de otros, pasamos a normalizar los datos para uniformarlos. De esta forma los atributos normalizados tienen media cero y varianza 1, moviéndose en un rango de valores similar. A los datos asociados a las cargas de refrigeración y calefacciones no se les aplicó la normalización.

2.4 Arquitectura de la red

Con los datos preprocesados y fijando una función de activación lineal, comenzamos con las pruebas para obtener una arquitectura óptima, variando el número de capas, taza de aprendizaje, cantidad de iteraciones y tolerancia del error

Luego de realizar los experimentos detectamos que la arquitectura que presenta mejor adaptacion al aprendizaje es xxxxxx

De esta forma, la arquitectura definitiva propuesta consiste en una capa de entrada de 8 + 1 neuronas, capa oculta de xxx neuronas y capa de salida de dos neuronas. Al igual que en el caso anterior, el entrenamiento lo hacemos con el 80% del dataset, dejando los restantes para validar.

2.5 Implementación de la red

A continuación detallaremos como implementamos la red.

----------------------chequear esta estructura---------------------

Implementamos una clase \textbf{Perceptron} con la siguiente estructura

\begin{lstlisting}

struct perceptron {

learning\_rate

tolerancia\_error

cantidad\_repeticiones

cantidad\_mezclas

input\_file

output\_file

tamano\_capa

tamano\_entrada

tamano\_salida

w1

w2

}

\end{lstlisting}

\begin{itemize}

\item \textbf{learning rate} = coeficiente de aprendizaje.

\item \textbf{tolerancia-error} = tolerancia de error.

\item \textbf{cantidad repeticiones} = cantidad de epocas.

\item \textbf{cantidad mezclas} = cantidad de veces que se ejecutará.

\item \textbf{input file} = archivo de entrada.

\item \textbf{output file} = archivo de salida.

\item \textbf{tamano capa} = tamaño de capa interna.

\item \textbf{tamano entrada} = tamaño de capa de entrada.

\item \textbf{tamano salida} = tamaño de capa de salida.

\item \textbf{w1} = vector de pesos de la primer capa.

\item \textbf{w2} = vector de pesos de la segunda capa.

\end{itemize}

Definida la estructura principal del \textbf{Perceptron} presentamos

las funciones principales que utilizá la \emph{Red Neuronal} para

la clasificación de datos.

\begin{itemize}

\item \textbf{entrenar} = entreniento de la red, realiza el preprocesamiento

del dataset, y para la cantidad seteada de mezclas realiza un entrenamiento, tomando

como cota la cantidad de iteraciones y la tolerancia del error. Dentro del ciclo principal

calcula la activación, corrección y adaptación de la red. Luego realiza cross-validation para

verificar los resultados de cada época.

\item \textbf{testing} = toma una red entrenada y calcula la tasa de aciertos.

\item \textbf{funcion activacion} = funcion de activación.

\item \textbf{funcion activacion derivada} = funcion de activación derivada.

\end{itemize}

-------------------------------------------

2.6 Experimentación y resultados

Definida la red procedemos a realizar un entrenamiento de la misma variando el tamaño de la capa interna, entrenamos el suficiente tiempo para encontrar el punto donde empieza a converger el ecm

la primera configuración es la siguiente

2.7 Conclusiones