Máster en Big Data Analytics

Curso 2017-2018

Introducción al Aprendizaje Automático

Práctica 1

Ejemplo de regresión para ajustar una función

Jon Ander Gómez Adrián
jon@dsic.upv.es
Departament de Sistemes Informàtics i Computació
Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Índice

1. Motivación

La regresión lineal es una de las técnicas más básicas en estadística y en aprendizaje automático, de las más sencillas, pero que resultan muy útiles en muchos casos.

En esta práctica vamos a utilizar un ejemplo artificial para el cual conocemos con precisión el proceso o función que genera los datos. Este ejemplo figura en la introducción del libro *"Pattern Recognition and Machine Learning"* [4] donde lo podréis encontrar más desarrollado.

La función a ajustar mediante regresión en este ejemplo es una función escalar de una variable escalar

$$sin(2 * \pi * x) \tag{1}$$

en el intervalo $[0, 1]$ (ver Figura 1).

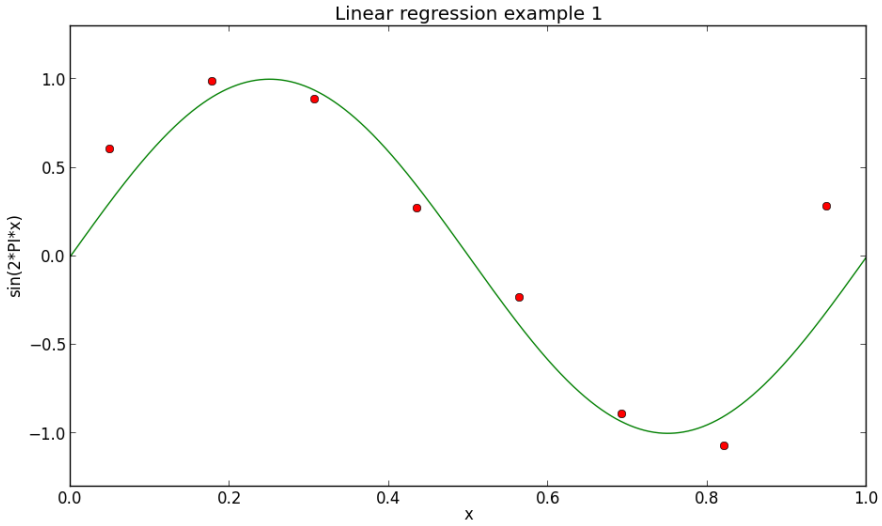

Figura 1: Función $sin(2 * \pi * x)$ para el intervalo $[0, 1]$ en verde. En rojo los puntos de los que se dispone para el aprendizaje, obtenidos artificialmente con ruido aditivo generado mediante una distribución normal de media 0 y desviación típica 0.25 $\mathcal{N}(0{,}0.25^2)$.

Se trata de una función cuya salida es un escalar (un valor real) al que debemos ajustar dado un valor de la variable de entrada x, que también es un valor real. Es el caso más simple porque las muestras son de una dimensión.

El conjunto de muestras de aprendizaje o *training set* es

$$\mathbf{X} = (x_1, x_2, \ldots, x_N)^T$$

$$\mathbf{t} = (t_1, t_2, \ldots, t_N)^T$$

$\mathbf{X}$ es el conjunto de variables de entrada, en este caso representado como un vector columna, y $\mathbf{t}$ es el conjunto de variables de salida, también representado como un vector columna. Cada par (x_i, t_i) se considera una muestra de aprendizaje obtenida mediante la siguiente función

$$t_i = sin(2 * \pi * x_i) + \mathcal{N}(0, \sigma^2) \tag{2}$$

donde la desviación típica σ =0.25, (ver Figura 1).

Como se comentó antes, el objetivo de la regresión es calcular $y(\cdot)$ tal que nos permita obtener el valor mejor aproximado posible $\hat{t}$ al valor correcto t de salida dada una observación $\hat{\mathbf{x}}$

$$\hat{t} = y(\hat{\mathbf{x}}) \tag{3}$$

En este ejemplo conocemos la ecuación (2) que genera los puntos del *training set*, y también la función subyacente a partir de la cual se generaron los datos y que queremos aproximar (1). Pero en un caso real sólo dispondremos del *training set*, es decir, del conjunto de pares de valores a partir del cual debemos ajustar el valor de salida por cada observación de la(s) variable(s) de entrada.

Como supuestamente desconocemos el proceso mediante el cual se generaron los datos, vamos a utilizar una función polinómica para realizar la regresión.

$$y(x, \mathbf{w}) = w_0 + w_1 * x + w_2 * x^2 + \cdots + w_M * x^M = \sum_{j=0}^{M} w_j * x^j \tag{4}$$

M es el grado máximo del polinomio y $\mathbf{w} \in \mathbb{R}^M$ es un vector de pesos que la técnica de aprendizaje automático debe estimar.

Si $M = 0$ entonces la solución será aproximar la función mediante un único valor (w_0), si $M = 1$ será regresión líneal, se estimarán w_0 y w_1. Ambos son casos particulares de la regresión polinómica. Sin embargo, en este ejemplo utilizaremos siempre regresión lineal. Lo haremos de manera encubierta, es decir, aplicaremos regresión lineal a muestras con más componentes, como si fuesen de más dimensiones. Por tanto, para valores de $M > 1$ generaremos

muestras $\mathbf{x}$ artificialmente tal que cada muestra $\mathbf{x_i}$ será igual a un vector $\{x_i^0, x_i^1, x_i^2, \ldots, x_i^M\}$, en este caso se trabajará con varias variables de entrada, concretamente con $M + 1$ variables, pero que son la original x_i en una dimensión elevada a distintas potencias.

La función de error debe ser siempre **no negativa**. Será igual a cero si la función $y(\cdot)$ obtiene el valor exacto para la función objetivo.

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n, \mathbf{w}) - t_n)^2 \tag{5}$$

Esta función es la suma del cuadrado de la diferencia entre la estimación y el valor a predecir. Se conoce como *sum-of-squares error*.

El problema consiste encontrar el valor de $\mathbf{w}$ para el cual $E(\mathbf{w})$ es lo más pequeño posible.

2. Objetivos

2.1. Conceptuales

1. Conocer el fenómeno del *over-fitting* y qué opciones existen para mitigarlo.

2. Conocer la importancia de que el *training set* sea representativo.

3. Error de la estimación (o predicción) como función objetivo a minimizar.

2.2. En el manejo de utilidades

1. Manejar las distintas funciones del paquete `scikit-learn` de Python para regresión lineal.

2. Ser capaz de crear funciones para utilizarlas en otras partes del programa y de generar ruido aleatorio para sumarlo a otras funciones y así conseguir muestras con cierto nivel de inexactitud.

3. Saber utilizar el paquete `matplotlib` para representar distintas funciones o valores.

3. Tareas

En la presente práctica vamos a ver la evolución del error cuadrático medio en su variante normalizada y conocida como *Root-Mean-Square*

$$E_{RMS} = \sqrt{\frac{2}{N}\sum_{i=1}^{N}(y_i - predict_i)^2} \tag{6}$$

para regresión líneal y regresión polinómica.

3.1. Tarea 1

Cambiad las siguientes líneas del fichero `linear_regression.py` para ver las gráficas y los valores estimados para el vector de pesos **w** según convenga:

```
verbose=False
intermediate_graphics=False
```

Ejecutad varias veces el código Python para ver el efecto de que las muestras se generan aleatoriamente:

```
$ python linear_regression.py
```

Estudiad el código correspondiente a la transformación de la variable de entrada a un vector que contiene el valor de la variable elevado a distintas potencias.

```
poly = PolynomialFeatures( degree = degree )
X_ = poly.fit_transform(X)
```

Para ello se puede mostrar por pantalla el contenido de la variable $X\_$ y sus dimensiones: $X\_.shape$ añadiendo temporalmente

```
print( X.shape )
print( X_.shape )
print( X_ )
```

3.2. Tarea 2

Variad los siguientes parámetros modificando el código Python `linear_regression.py`

```
num_samples=8
max_degree=1+12
```

num_samples en el rango $[4, 100]$, podríamos poner más de 100 pero no se notarán mejoras significativas.

max_degree poniendo 10 o 12 se ve claramente que para la función $sin(2\pi x)$ en el intervalo $[0, 1]$ no se mejora con polinomios de grado mayor a 8.

El ejercicio de ejecutar varias veces el programa `linear_regression.py` cambiando los valores de algunos parámetros nos permite comparar distintos modelos, en este caso distintos grados del polinomio, y seleccionar aquél modelo que mejor se ajusta en función de los datos disponibles.

Dada la función utilizada, los grados $M = 0$ y $M = 1$ son claramente muy pobres, también se observa que a partir de cierto grado del polinomio el error empeora.

Para grados altos se observa claramente el fenónemo del *over-fitting*: error bajo para el *training set* y alto para el *test set*.

¿Qué efecto tiene disponer de más muestras de aprendizaje?

3.3. Tarea 3

Se propone como ejercicio modificar el código Python para trabajar con una función diferente y con diferentes rangos, por ejemplo una polinómica de grado 2, después probad otra de grado mayor. Para ello debe modificarse la siguiente función dentro del código

```
def f(x):
    return numpy.sin( 2 * numpy.pi * x )
```

por algo similar a

```
def f(x):
    return x*x-10*x+3
```

También será necesario cambiar otras líneas de código con objeto ver un mayor ámbito de la nueva función

```
zx = numpy.linspace(0.0,1.0,1000)
zy = f(zx)
```

sustituyendo los valores 0,0 y 1,0 para generar un mayor rango de la variable de entrada y también de la función. Otra línea de código a modificar es

```
X_train = numpy.linspace(0.05,0.95,num_samples)
```

variando los valores 0,05 y 0,95, para generar muestras dentro del rango que corresponda según los valores que se hayan especificado en `numpy.linspace()`.

¿Qué conclusiones podemos sacar por el hecho de utilizar una función de aproximación que sí corresponde a la función desconocida a partir de la cual se generaron los datos?

3.4. Tarea 4

Si la función de error a minimizar se define de la siguiente manera, podemos ver cómo los valores de los pesos no son extremadamente grandes en valor absoluto, esto afecta positivamente a la generalización. Gracias a esta técnica se consiguen mejores estimaciones con relativamente pocos datos.

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(y(x_n, \mathbf{w}) - t_n)^2 + \frac{\alpha}{2}\|\mathbf{w}\|^2 \tag{7}$$

Se conoce como *Ridge Regression* [8] o *weight decay*. Además de conseguir mejor generalización evita inestabilidades numéricas en los cálculos de coma flotante porque los valores de los pesos se ajustan dentro de rangos acotados.

Para ello cambiaremos la línea de código Python:

```
linear_regressor = linear_model.LinearRegression()
```

por

```
linear_regressor = linear_model.Ridge(alpha=0.001)
```

y repetiremos la ejecución del programa varias veces para la función seno y para las polinómicas que hayamos probado. Importante poner `verbose` a *True* para visualizar los valores de los coeficientes obtenidos utilizando *Rigde* y sin utilizarlo.

4. Conclusiones

En este ejemplo el efecto de utilizar *Ridge Regression* no se percibe claramente en los resultados porque no estamos trabajando con alta dimensionalidad. Pero es una técnica a tener en cuenta por lo comentado sobre precisión

en los cálculos y sobre generalización, pues ayuda a mitigar el efecto del *over-fitting*. Aunque se ha visto que la mejor receta para mitigar el *over-fitting* es disponer de mayor número de muestras de aprendizaje, es decir, de un *training set* representativo.

Respecto del manejo de utilidades dentro de Python, y de programación en general, es importante conocer el paquete `matplotlib` que nos permite visualizar de manera sencilla funciones y conjuntos de valores. También es importante conocer la mecánica de las técnicas de regresión: se crea un objeto correspondiente a la técnica de aprendizaje que se vaya a utilizar, se entrena mediante la función `fit()` y después se utiliza el mismo objeto para predecir.

5. Bibliografía

[1] S. authors. Deep Learning. `http://www.deeplearning.net`, 2015. [Online: accessed June 2015].

[2] Y. Bengio, I. J. Goodfellow, and A. Courville. Deep learning. Book in preparation for MIT Press, 2015.

[3] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[4] C. M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006.

[5] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Addison-Wesley, second edition, 2000.

[6] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):2–42, 2006.

[7] G. Hinton. A practical guide to training restricted boltzmann machines, 2010.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, JMLR*, 12:2825–2830, 2011.

[9] Wikipedia. Machine Learning. `http://en.wikipedia.org/wiki/Machine_learning`, 2015. [Online: accessed April 2015].