UNIVERSIDAD
COMPLUTENSE
MADRID

Desarrollo de Aplicaciones y Sistemas Inteligentes (DASI)

Regresión logística

Juan Pavón Mestras
Dep. Ingeniería del Software e Inteligencia Artificial UCM

DASI

Regresión logística

Regresión logística

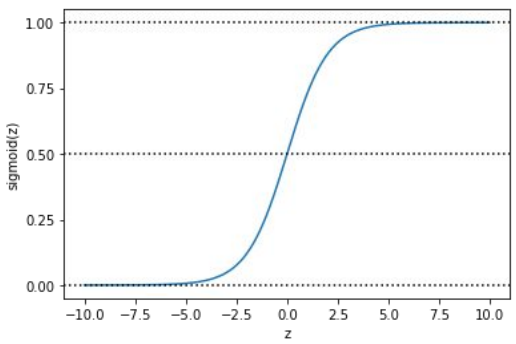
- La **regresión logística** o **modelo *logit***, es un clasificador probabilístico que permite predecir una variable cualitativa Y a partir de un conjunto de variables cualitativas X_i utilizando una **función logística**
- Según los valores que pueda tomar la variable Y puede ser
  - Regresión binaria: dos valores ⇒ función logística (***sigmoid***)
  - Regresión multinomial: más de dos valores ⇒ función ***softmax***
- Es un método eficiente y muy escalable

Función logística

- También conocida como **función *sigmoid***

$$y(z) = \frac{1}{1 + exp(-z)}$$

  - Obsérvese que la función tiene la forma de una S, y que todas las entradas se transforman en valores dentro del rango de 0 a 1
    - Para entradas positivas, un valor mayor da lugar a una salida más cercana a 1
    - Para entradas negativas, un valor menor genera una salida más cercana a 0
    - Cuando la entrada es 0, la salida es el punto medio, 0,5



Regresión logística

- En regresión logística se aplica la función logística a la suma ponderada de los valores de las características x_i:

$$z = w_1x_1 + w_2x_2 + \ldots + w_nx_n = w^Tx$$

  - Normalmente se le añade un sesgo (bias) w_0, y quedaría:

$$z = w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n = w^Tx$$

- La predicción Y será la probabilidad de la clase positiva (1), usando la función logística:

$$\hat{y} = P(y = 1 \mid x) = \frac{1}{1 + exp(-w^Tx)}$$

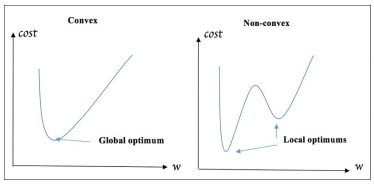
⇒ El **modelo de regresión logística** es el vector de pesos ***w*** que se aprende utilizando el conjunto de datos de entrenamiento

Regresión logística - Función de coste

- Al entrenar el modelo se procura minimizar el **error cuadrático medio** (en inglés, *mean squared error, **MSE***), la media de los cuadrados de las diferencias entre el valor real y el predicho
  - En el caso de clasificador binario (clase positiva=1, negativa=0), la **función de coste** `J(w)` será:

$$J(w) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(\hat{y}(x^{(i)}) - y^{(i)})^2$$

- Esta función de coste no es convergente (puede haber muchos óptimos locales), y por ello se suele utilizar esta otra variante, **pérdida logarítmica** (***log loss***):



$$J(w) = \frac{1}{m}\sum_{i=1}^{m} -[y^{(i)}log(\hat{y}(x^{(i)})) + (1 - y^{(i)})log(1 - \hat{y}(x^{(i)}))]$$

Regresión logística - Descenso del gradiente

- Para minimizar la función objetivo utilizaremos la técnica conocida como **descenso del gradiente**
  - Procedimiento de minimización de una función objetivo mediante optimización iterativa de primer orden
  - En cada iteración, se desplaza un paso que es proporcional a la derivada negativa de la función objetivo en el punto actual
    ⇒ **tasa de aprendizaje** o tamaño del paso (μ)

    $$w = w - \mu\Delta w$$

    que operando sale:

    $$w := w + \eta \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}(z^{(i)}))x^{(i)}$$

  - El punto a optimizar se desplaza iterativamente cuesta abajo hacia el valor mínimo de la función objetivo, dando lugar a lo que buscamos:

$$y' = \frac{1}{1 + \exp(-\boldsymbol{w}^T \boldsymbol{x}')}$$

$$\begin{cases} 1, if\ y' \geq 0.5 \\ 0, if\ y' < 0.5 \end{cases}$$

El umbral 0,5 se puede cambiar, por ejemplo para un sistema de predicción de incendios, sería más bajo (p.ej. 0,3), o más alto si se quiere asegurar la probabilidad de acierto (p.ej. el éxito de ventas de un producto)

Ejemplo

- Código de implementación:

  https://colab.research.google.com/drive/1sGWNUo0z-Kic1OCoLP47PiPoy9pGdd4i?usp=sharing

  Para el ejemplo de predicción de clicks:

  https://colab.research.google.com/drive/1sRaM5as2MVtnyWovXX49Ha2Tli2nqpRF?usp=sharing

DASI

Regresión logística con descenso del gradiente estocástico

Descenso del gradiente estocástico

- El método del descenso del gradiente, al tener que actualizar todos los pesos en cada iteración es poco escalable
- El método de **descenso de gradiente estocástico** (**SGD**, de *stochastic gradient descent*) lo intenta resolver:
  - En cada actualización solo se utiliza un ejemplo de entrenamiento en vez de todo el conjunto
  - Cada iteración será la actualización uno por uno de todos los ejemplos de entrenamiento
- Ver código en https://colab.research.google.com/drive/1sRaM5as2MVtnyWovXX49Ha2Tli2nqpRF?usp=sharing

DASI

Regularización

Regularización

- Es un término que se añade a la función de coste:

$$J(w) = \frac{1}{m}\sum_{i=1}^{m} -[y^{(i)} log(\hat{y}(x^{(i)})) + (1 - y^{(i)}) log(1 - \hat{y}(x^{(i)}))] + \alpha ||w||^q$$

  - α es una constante que multiplica el término de regularización
    - Si es demasiado grande el modelo puede generalizar demasiado, dando lugar a falta de ajuste
    - Si es demasiado pequeño, no tendría mucho efecto la regularización
  - Dependiendo del valor de q:
    - Regularización L1: q = 1 $\Rightarrow ||w||^1 = \sum_{j=1}^{n} |w_j|$
    - Regularización L2: q = 2

- Introduciendo este término en la función de coste se consigue penalizar que haya algunos pesos w_i , w_h … que tengan valores muy altos y que generarían overfitting

DASI

Aprendizaje online

Aprendizaje online

- En vez de entrenar el modelo de golpe con un conjunto de datos, se puede plantear hacerlo incrementalmente
  - Se gana escalabilidad
  - Es útil cuando van apareciendo nuevos datos en el tiempo
    - Datos de mercados financieros, meteorológicos, etc.

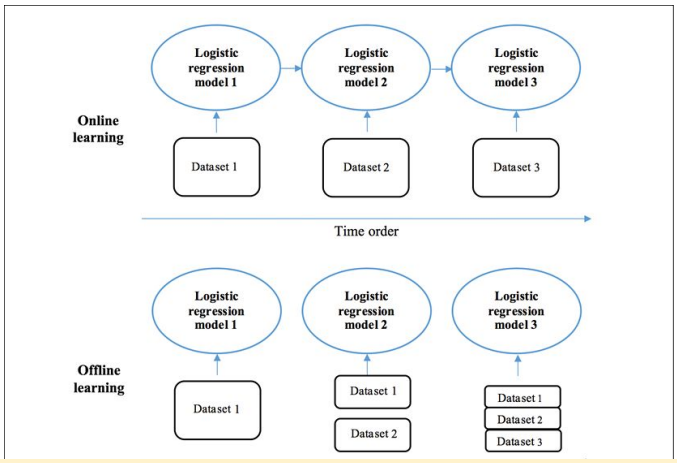


Figure 5.7: Online versus offline learning, del libro Yuxi (Hayden) Liu: *Python Machine Learning By Example, Third Edition*. Packt Publishing, 2020

Aprendizaje online

- El módulo `SGDClassifier` de scikit-learn implementa el aprendizaje online con el método **`partial_fit`**

Ejercicio

- Utilizando la clase **`SGDClassifier`** del módulo `sklearn.linear_model`, implementar el predictor de clicks
  - Tomar los datos del fichero **train300mil.csv**, y usar **10.000** ejemplos, aplicando **OneHotEncoder** para las variables categóricas
  - Una primera versión con los siguientes parámetros para el entrenamiento:
    - **loss='log_loss'** como función de coste
    - **penalty=None** es la regularización para reducir el overfitting (que aquí no se usa y veremos luego el efecto que tiene)
    - **max_iter=10** es el número de iteraciones
    - **eta0=0.01** es la tasa de aprendizaje. Probar primero con 0.01 y con **learning_rate='constant'** para indicar que no se cambia durante el entrenamiento (por defecto es 'optimal', que significa que va decreciendo a medida que se van haciendo actualizaciones).
  - Introducir **regularización L1** para comparar con el modelo anterior
    - **alpha=0.0001**
  - Aplicar aprendizaje online (**partial_fit**) usando todos los datos del fichero, con lotes de 10.000: 90% de los datos para entrenamiento y 10% para prueba

DASI

Bibliografía

Bibliografía

- Sridhar Alla & Suman Kalyan Adari: *Beginning MLOps with MLFlow. Deploy Models in AWS SageMaker, Google Cloud, and Microsoft Azure*. Apress, 2021.
- Yuxi (Hayden) Liu: *Python Machine Learning By Example, Third Edition*. Packt Publishing, 2020.
- Documentación en línea de las distintas herramientas