Máster en Big Data Analytics

Curso 2017-2018

Introducción al Aprendizaje Automático

Práctica 2

Aplicación de Naive Bayes a los datasets Iris y MNIST: hand-written digits

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 **2**

2. Objetivos **2**
2.1. Conceptuales 2
2.2. En el manejo de utilidades 2

3. Tareas **3**
3.1. Tarea 1 3
3.2. Tarea 2 3
3.3. Tarea 3 4
3.4. Tarea 4 4
3.5. Tarea 5 5

4. Conclusiones **5**

5. Bibliografía **6**

1. Motivación

Tras ver un poco de teoría de la probabilidad y la idea *Naive Bayes* para implementar algunos clasificadores, vamos a ver cómo aplicar un clasificador del tipo *Naive Bayes* utilizando la distribución normal o de Gauss para estimar la densidad de probabilidad de cada variable de entrada con respecto a cada valor de la variable salida.

$$p(x_i \mid t) = \mathcal{N}(\mu_t, \sigma_t^2) = \frac{1}{\sqrt{2\pi}\sigma_t} e^{-\frac{1}{2}(\frac{x_i - \mu_t}{\sigma_t})^2} \tag{1}$$

donde μ es el valor medio de la variable de entrada x_i cuando la variable de salida toma el valor t y σ su desviación típica. El clasificador que vamos a utilizar estima los valores de μ_t y σ_t por cada variable de entrada con respecto a cada clase t en las que se pueden clasificar las muestras.

Según el *dataset* utilizado (Iris o MNIST) el número de clases a las que puede pertenecer cada muestra es diferente. Este tipo de factores suelen tener gran relevancia en los resultados, pues no es lo mismo distinguir entre tres clases (caso del *dataset* Iris) que entre diez clases como es el caso de los dígitos manuscritos.

2. Objetivos

2.1. Conceptuales

1. Ver que no siempre las simplificaciones tienen buenos resultados. Depende del problema y de cómo estén los datos las premisas del tipo *Naive Bayes* son asumibles o no.

2. Ver que realizando ciertas transformaciones a las muestras (variables de entrada), como puede ser la reducción de la dimensionalidad mediante PCA, podemos obtener mejoras significativas en algunas técnicas de clasificación. En este caso, las mejoras sobre un clasificador del tipo *Naive Bayes* con distribuciones normales son importantes cuando se aplica al *dataset* MNIST de dígitos manuscritos.

3. Entender los conceptos de partición de test y partición de validación, así como la técnica de validación cruzada (*cross-validation*).

2.2. En el manejo de utilidades

1. Aprender a descargar *datasets* disponibles para `scikit-learn`.

2. Saber barajar (shuffle) las muestras de un *dataset*.

3. Ser capaz de realizar distintas particiones en los *datasets* para distinguir entre partición o subcorpus de entrenamiento, de validación y de test.

4. Saber cómo implementar la técnica de la validación cruzada.

3. Tareas

3.1. Tarea 1

Esta tarea es un simple primer paso para ver como cargar un *dataset* y cómo visualizarlo mediante herramientas gráficas tras reducir la dimensionalidad mediante *Principal Component Analysis* (PCA).

Descargad de la Web de `scikit-learn` el ejemplo `plot_iris_dataset.py` y ejecutadlo:

```
$ python plot_iris_dataset.py
```

Después lo mismo con

```
$ python nb_iris_dataset.py
```

pero tomando el fichero modificado para esta práctica disponible en *PoliformaT*.

En este ejemplo se ve cómo crear un objeto de la clase `GaussianNB`, cómo utilizarlo para entrenar a partir del `dataset` y cómo comprobar la precisión de la predicción sobre el mismo *dataset*.

En realidad esto no es una buena práctica, no se puede saber la capacidad del modelo para generalizar. La precisión sobre el subcorpus de entrenamiento no es una buena medida de la calidad, en este caso, del clasificador.

3.2. Tarea 2

Estudiad el código de `load_mnist.py` para ver un ejemplo de cómo descargar y guardar en caché un *dataset* de los que hay disponibles para utilizar en `scikit-learn` y otros toolkits de *Machine Learning*.

Los datos se descargan desde `http://mldata.org` y se guardan en un directorio por defecto a partir del `HOME` del usuario. Se puede comprobar tras ejecutar la descarga:

```
$ python load_mnist.py
$ ls -lR ~/scikit_learn_data
```

3.3. Tarea 3

En esta tarea también vamos a utilizar un clasificador *Gaussian Naive Bayes* sobre el *dataset* MNIST. En primer lugar ejecutaremos

```
$ python nb_mnist_dataset.py
```

para comprobar cómo se comporta dicho clasificador en este *dataset*. Pero claro, no es concluyente trabajar con todo el corpus para entrenamiento y para test. En el siguiente ejemplo se utilizan las primeras 60000 muestras para entrenamiento y las últimas 10000 para test.

```
$ python nb_mnist_dataset_2.py
```

Ambos ejemplos demuestran que un clasificador del tipo *Naive Bayes* no es una buena opción cuando se trata de imágenes con valores enteros para indicar la intensidad de los píxeles. Con el siguiente ejemplo se puede visualizar un dígito aleatorio y hacernos una idea de que como es cada muestra.

```
$ python show_mnist.py
```

3.4. Tarea 4

Realizemos algunas transformaciones a los datos de entrada para reducir la dimensionalidad y ver si podemos conseguir algún efecto positivo.

```
$ python nb_mnist_dataset_3.py
```

Este programa realiza una simple transformación al mismo tiempo que realiza una reducción de la dimensionalidad. Esta reducción consiste en transformar un muestra que es una imagen de 28×28 a un vector de 56 dimensiones. Las primeras 28 componentes de cada nuevo vector contienen la suma de las intensidades fila por fila, y las siguientes 28 la suma de la intensidades columna por columna. Es una representación alternativa de la imagen.

¿Qué resultados obtenemos ahora? ¿Mejores o peores que antes? ¿Se nos ocurre alguna otra operación para mejorar los resultados?

Mientras se nos ocurre algo para mejorar estudiemos la idea de la validación cruzada (*cross-validation*) para seleccionar el modelo que mejor resultados nos da en la validación para comprobar su validez con el de test.

```
$ python nb_mnist_dataset_3_cv.py
```

Tras ejecutar este último programa Python vemos como, además de barajar las muestras al principio y apartar una partición o subcorpus para test, ahora utilizamos otra partición para validación. De manera que una parte de la partición de entrenamiento no se utiliza para aprender o estimar el modelo, se utiliza para validarlo, es decir, para comprobar como se comporta. Si este proceso lo repetimos varias veces (10 en el ejemplo) y seleccionamos aquél modelo que mejor resultados ha dado con la validación, podemos esperar que será el que mejor resultados nos dará con la partición de test, aunque no es seguro al 100 %.

3.5. Tarea 5

Aplicad *Principal Component Analysis* (PCA) para reducir la dimensionalidad.

Tomando como ejemplo lo visto en la tarea 1, modificad el código de `nb_mnist_dataset_3_cv.py` y realizad una transformación a la variable multidimensional de entrada. No a la variable salida o `target`.

4. Conclusiones

Con esta práctica se debe haber aprendido a realizar particiones en los *datasets* para comprobar sobre ellos el funcionamiento de los algoritmos de aprendizaje, sea para clasificación o para regresión. Es muy importante no calcular la precisión de un algoritmo con los mismos datos utilizados para entrenamiento. Es una mala práctica que principalmente sirve para engañarse a uno mismo.

También se ha visto la técnica de la validación cruzada, es una idea muy simple que consiste en separar una parte de las muestras de entrenamiento para comprobar la precisión del modelo estimado utilizando el resto de muestras. Si esto se repite varias veces podemos obtener la precisión del modelo en término medio, o bien seleccionar el que mejor resultado consigue, como se ha visto en esta práctica. Nótese como el resultado del mejor modelo no es exactamente el mismo para la partición de test que para la de validación, el porcentaje de aciertos variará un poco.

En otras técnicas de aprendizaje basadas en el descenso por gradiente la partición de validación juega un papel muy importante para controlar el sobre aprendizaje (*over-fitting*). Cuando el error sobre la partición de validación comienza a subir o se estabiliza ya no es necesario seguir iterando.

En esta práctica también se ha observado como la idea *Naive Bayes* no siempre obtiene buenos resultados. Pero dado que comparada con otras técnicas es mucho más rápida, en algunos problemas quizás compense sacrificar algo de precisión en aras de conseguir tiempo real o tiempos de respuesta aceptables para la tarea en la que estemos trabajando. Este tipo de decisiones las debe tomar el experto en técnicas de aprendizaje automático.

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].