Máster en Big Data Analytics

Curso 2017-2018

Introducción al Aprendizaje Automático

Práctica 6

Kernel Density Estimators frente a k-Nearest Neighbors

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

En las prácticas 2 y 3 hemos aplicado dos técnicas paramétricas de aprendizaje supervisado para clasificación sobre un *dataset*, un clasificador del tipo *Naive Bayes* de tipo *Gaussiano* y un clasificador también basado en *Gaussianas* entrenado mediante *Maximum Likelihood Estimation* (MLE).

En la práctica 4 se ha visto un clasificador también Gaussiano pero cuyo entrenamiento se basa en aprendizaje semi-supervisado, pues una parte se realiza mediante *clustering* para obtener una mejor estimación de la densidad de probabilidad de cada clase conocida. Para cada clase se ha entrenado un GMM con el algoritmo EM.

En la presente práctica vamos a ver dos técnicas no paramétricas de aprendizaje supervisado cuando se necesita utilizarlas para tareas de clasificación. También sirven como técnicas para estimar densidades de probabilidad.

2. Objetivos

2.1. Conceptuales

- Utilizar técnicas no paramétricas de entrenamiento de clasificadores, útiles cuando no se conoce la distribución de probabilidad según la cual se generan las muestras, y sobre todo cuando se sabe que no se distribuyen según una función de densidad de probabilidad conocida.

2.2. En el manejo de utilidades

- Utilizar la clase `sklearn.neighbors.KernelDensity` para estimar densidades de probabilidad para cada una de las clases conocidas, y después clasificar las muestras del subconjunto de test según la clase que les asigna mayor probabilidad.

- Utilizar la clase `sklearn.neighbors.KNeighbors.Classifier` para comprobar su comportamiento en el mismo *dataset*.

- Saber construir los scripts en Python que cargan los *datasets*, importan los módulos necesarios, preprocesan el *dataset* y aplican las dos técnicas de clasificación propuestas en los puntos anteriores.

  Para ello el alumno partirá de los scripts que se han ido construyendo en las prácticas anteriores y añadirá lo correspondiente a las nuevas técnicas.

3. Tareas

3.1. Tarea 1

Descargad el ejemplo Python de `scikit-learn`

`http://scikit-learn.org/stable/auto_examples/neighbors/plot_kde_1d.html`

y ejecutadlo varias veces cambiando el número de muestras N para el caso de obtener los histogramas. En el ejemplo tiene un valor de 20, podéis probar hasta 1000. También podéis aumentar el número de particiones o *bins* de los histogramas, línea

```
bins = np.linspace( -5, 10, B )
```

B por defecto vale 10, podéis aumentarlo hasta 100, aunque mucho más de 50 no tiene sentido. Ojo que N siempre debe ser mayor que B, se recomienda $N > 2 * B$.

3.2. Tarea 2

Desarrollad el programa Python necesario para aplicar KDE sobre el *dataset MNIST* de dígitos manuscritos.

Utilizad la clase `sklearn.neighbors.KernelDensity` para estimar las densidades correspondientes a cada clase, es decir, a cada dígito.

Una vez realizado el paso anterior, utilizad el método `score_samples` de dicha clase para obtener los logaritmos de la probabilidad que se asigna a cada muestra del subconjunto de test y asignad cada muestra a la clase más probable.

Comprobad la precisión de este clasificador al igual como se hizo en las prácticas 2, 3 y 4. También debéis probar el efecto de reducir la dimensionalidad con PCA o con la técnica que se aplicó desde la práctica 2 consistente en acumular intensidad de gris por filas y por columnas. Realizad esta comprobación también en las siguientes tareas.

Fichero de la práctica `kde_sobre_mnist.py`

3.3. Tarea 3

Utilizad la clase Python `MyKernel.py` facilitada para a partir del código anterior probar un clasificador cuyo código tenéis disponible. Básicamente es cambiar uno por otro.

Ficheros de la práctica:

- `kde_with_mykernel_sobre_mnist.py`
- `MyKernel.py`
- `MyKernelClassifier.py`

Ambos clasificadores van lentos porque se trata de un *dataset* bastante grande. Podéis dejar los procesos calculando y probad ambos clasificadores con otros *datasets* bien sintéticos o que os descarguéis.

Ficheros de la práctica:

- `kde_1_with_mykernel.py`
- `kde_2_with_mykernel.py`

3.4. Tarea 4

De manera similar, utilizad uno de los scripts anteriores para en lugar de utilizar estimación de densidades en base a una función *kernel* utilizar un clasificador basado en los k-Nearest Neighbors.

Probad distintos valores del número de vecinos, parámetro `n_neighbors` del clasificador `sklearn.neighbors.KNeighbors.Classifier` para observar cómo puede llegar a cambiar su comportamiento.

Este clasificador también irá muy lento con este *dataset*, dejadlo en funcionamiento y probad la técnica con otros *datasets* más pequeños.

Se debe probar esta técnica con el *dataset MNIST* porque debéis ir rellenando la tabla que desde la práctica 2 estaréis construyendo con objeto de estudiar cómo se comportan las distintas técnicas de *Machine Learning* estudiadas sobre un mismo *dataset*.

4. Conclusiones

En esta práctica se han visto dos técnicas de aprendizaje supervisado pero no paramétrico que podrán ser comparadas con las anteriores técnicas sí paramétricas y con las que trabajaremos en las siguientes clases.

En este punto uno ya debe darse cuenta de que no existe una técnica que sea la mejor en todos los casos. Según cada problema se obtendrán mejores resultados con una u otra. Además, también es muy importante ver el efecto de preprocesar los datos, pues según qué técnicas de clasificación o regresión se utilicen el preproceso puede ayudar a mejorar los resultados.

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].