

Práctica 2 Introducción al Aprendizaje Automático

Jose Joaquín Rodríguez García y Araceli Teruel Domenech

Universidad Politécnica de Valencia

Master en Big Data Analytics

Octubre 2017

1. Primera Tarea

1.1. Enunciado

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.

1.2. Resolución

Estudiando el código del primer archivo `plot_iris_dataset.py` vemos cómo carga un dataset, le aplica un PCA con 3 componentes y lo muestra gráficamente. En el segundo archivo `nb_iris_dataset.py` carga un dataset *MNIST original* y le aplica un Gaussian

Naive Bayes con todos los datos cargados

```
mnist = fetch_mldata( 'MNIST_original' )  
gnb = GaussianNB()  
gnb.fit( mnist.data , mnist.target )
```

Y usando los mismos datos cargados, y con los que hemos entrenado, hacemos la predicción. Tras ejecutarlo varias veces, podemos comprobar que los resultados no varían:

```
6 muestras mal clasificadas de 150  
Accuracy = 96.0\%
```

Esto es porque estamos usando la misma muestra tanto para training como para test, lo cual no es una buena práctica.

2. Segunda Tarea

2.1. Enunciado

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
```

2.2. Resolución

Podemos comprobar que se ha generado una carpeta en la carpeta de usuario

 python	10/01/2018 16:41	Carpeta de archivos
 scikit_learn_data	10/01/2018 16:50	Carpeta de archivos

en la que encontramos un dataset

 mnist-original

10/01/2018 16:50

Microsoft Access ...

54.142 KB

3. Tercera Tarea

3.1. Enunciado

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
```

3.2. Resolución

Empezamos ejecutando el archivo `nb_mnist_dataset.py` vemos una predicción usando un clasificador Gaussian Naive Bayes sobre el dataset MNIST pero usando todo el dataset tanto para training como para test. Vemos los resultados que obtenemos y son 30924 muestras mal clasificadas de 70000 con 55.8 % de precisión. Pasando al siguiente archivo `nb_mnist_dataset_2.py` hacemos exactamente la misma predicción pero esta vez usando 60000 muestras iniciales para el entrenamiento y las 10000 restantes para el test

```
X_test = mnist.data[60000:]  
Y_test = mnist.target[60000:]
```

Y mezclamos los datos

```
(X_train, Y_train) = shuffle( mnist.data[:60000], mnist.target[:60000] )
```

Los resultados obtenidos son 4442 muestras mal clasificadas de 10000 con un 55.8% de precisión.

4. Cuarta Tarea

4.1. Enunciado

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 una muestra que es una imagen de 28 x 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 las 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%.

4.2. Resolución

Podemos comprobar que reduciendo dimensión sí que obtenemos una mejora en los resultados: 3003 muestras mal clasificadas de 10000 Accuracy = 70.0 %

Usando Cross Validation a estos datos con dimensiones reducidas vemos que la mejora no es mayor

3003 muestras mal clasificadas de 10000 Accuracy = 70.0 %

5. Quinta Tarea

5.1. Enunciado

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.

5.2. Resolución

Haciendo un PCA a las variables de la siguiente manera:

```
X_train = PCA(n_components=15).fit_transform(X_train)
X_test = PCA(n_components=15).fit_transform(X_test)
```

y ejecutando nuevamente el fichero obtenemos unos resultados bastante peores pero en tiempo de ejecución es bastante mejor

4726 muestras mal clasificadas de 10000

Accuracy = 52.7 %