Práctica 3 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

Utilizar la versión de código Python de la práctica anterior y modificarlo para que se utilice un objeto de la clase MyGaussian en lugar de un objeto de la clase GaussianNB. En el código *MyGaussian.py* se dispone de una implementación de un clasificador basado en Gaussianas, es decir, cada clase se modela mediante una distribución normal o de Gauss, y los parámetros de cada Gaussiana (μ y Σ) se estiman por máxima verosimilitud. Revisad bien el código de MyGaussian.py para entender cómo se realiza la estimación por máxima verosimilitud. Ejecutad repetidas veces el clasificador sobre el dataset MNIST variando el preproceso a los datos para estudiar el efecto que sobre este nuevo clasificador tienen las transformaciones a las muestras. ¿Qué preproceso es el más idóneo para este clasificador? ¿Coincide en algunos casos con el comportamiento del clasificador anterior?

1.2. Resolución

Hemos realizado el estudio con diferentes PCA (Entre 20 y 40 número de componentes) y hemos llegado a la conclusión de que el mejor resultado obtenido para realizar una predicción usando un objeto de la clase MyGaussian los mejores resultados los obtenemos con 25 número de componentes principales, obteniendo así una precisión del 95.1 % y 492 muestras mal clasificadas de 10000.
Comparando con los resultados anteriores, podemos asegurar que el proceso más idóneo para este clasificador es que utiliza un objeto de la clase MyGaussian en lugar de un

objeto de la clase GaussianNB. De cualquier forma, obtenemos por lo general mejores resultados.

2. Segunda Tarea

2.1. Enunciado

Se propone trabajar con normalización de los datos de entrada, es decir, que cada componente de los vectores de características o muestras esté normalizado de manera que siga una distribución normal o de Gauss con media cero y varianza 1: $\mathcal{N}(0, 1)$. Para ello debe importarse la classe StandardScaler de Scikit-Learn:

from sklearn.preprocessing import StandardScaler

y se utiliza como sigue para transformar un conjunto de muestras:

norm = StandardScaler()
new_X = norm.fit_transform(X)

Se propone también probar distintas variantes del preproceso. Llegados a este punto, se aconseja prepararse una hoja de cálculo e ir apuntando los resultados obtenidos por cada variante de clasificador, de preproceso, etc.Será de mucha utilidad para hacer el estudio o exploración de combinaciones de variantes para responder a las preguntas que figuran al final, cuyas respuestas servirán como trabajo para evaluar una parte de esta asignatura. Otra transformación sugerida, que podría combinarse con otras es Principal Component Analysis:

from sklearn.decomposition import PCA

y se utiliza como sigue para transformar un conjunto de muestras:

pca = PCA(n_components=30)
new_X = pca.fit_transform(X)

2.2. Resolución

Al normalizar la variable obtenemos unos resultados parecidos a los que teníamos, si que es cierto que en cuestión de memoria, al normalizarlo, usa menos memoria que si no se normaliza

```
396 muestras mal clasificadas de 10000
Accuracy = 96.0\%
```

3. Tercera Tarea

3.1. Enunciado

Utilizad el código de MyGaussian.py para trabajar con la matriz de varianzas-covarianzas diagonal. El código MyGaussian.py ya está preparado para este caso. Puede verse que en el constructor de la clase se puede especificar tipo de matriz de varianzas-covarianzas. Ejemplos de cómo invocar al constructor:

$$classificador = MyGaussian(\ covar\_type='full'\ )$$
$$classificador = MyGaussian(\ covar\_type='diag'\ )$$

Como en la tarea anterior, los resultados obtenidos aquí deben servir para responder algunas de las preguntas del trabajo a evaluar.

3.2. Resolución

No será necesario usar $classificador = MyGaussian(\ covar\_type='full'\ )$ dado que por defecto, usa este $covar\_type$ por lo que los resultados son los mismos que los obtenidos en el ejercicio anterior, sin embargo, usando $covar\_type='diag'$ podemos observar que obtenemos peores resultados, tanto es así que el mejor resultado obtenido ha sido con 37 componentes principales obteniendo una predicción del 86 % con 1396 muestras mal clasificadas de 10000.

4. Cuarta Tarea

4.1. Enunciado

Utilizad el código de MyGaussian.py para trabajar con la matriz de varianzas-covarianzas diagonal o completa pero compartida entre todas las clases. En este caso la media se sigue estimando por cada clase, pero la matriz de varianzas-covarianzas se calcula a partir de la media global de todas las muestras.

$$classificador = MyGaussian(\ covar\_type='tied'\ )$$
$$classificador = MyGaussian(\ covar\_type='tied\_diag'\ )$$

Los resultados obtenidos aquí también deben servir para responder algunas de las preguntas del trabajo a evaluar.

4.2. Resolución

Siguiendo con las pruebas, y usando *covar_type='tied'* podemos comprobar que con el mismo número de componentes, todavía empeora más los resultados, obteniendo como máximo un 82.1 % de predicción. Hemos hecho la prueba de aumentar el número de componentes principales y los resultados son prácticamente parecidos, hemos conseguido en estse caso un 85.3 % de precisión usando 42 componentes principales, pero el tiempo que tarda en hacer la predicción se eleva de manera considerada.

5. Conclusiones

Como hemos podido comprobar para este *dataset* la mejor opción es utilizar una matriz de varianzas-covarianzas completa estimándose una diferente para cada clase. Si ahora usásemos distintos *datasets*