

Práctica obligatoria 2

Reconocimiento de objetos

|  |
| --- |
| Adrián García Oller |
| Daniel Lois Nuevo |
| Francisco Robles Castro |

Contenido

[Ejercicio 1. 3](#_Toc7869514)

[Algoritmo. 3](#_Toc7869515)

[Estadísticas obtenidas 4](#_Toc7869516)

[Ejercicio 2. 6](#_Toc7869517)

[2. LBP 6](#_Toc7869518)

[Algoritmo. 6](#_Toc7869519)

[Estadísticas obtenidas. 6](#_Toc7869520)

[3. PCA 9](#_Toc7869521)

[Algoritmo. 9](#_Toc7869522)

[Estadísticas obtenidas. 9](#_Toc7869523)

[4. KNN 11](#_Toc7869524)

[Algoritmo. 11](#_Toc7869525)

[Estadísticas obtenidas. 11](#_Toc7869526)

[Problemas encontrados. 13](#_Toc7869527)

# Ejercicio 1.

## Algoritmo.

|  |  |
| --- | --- |
| Opción: | --classifier LDA-Bayes |

Clases implicadas= CharacteristicsExtractor.py, main.py, ComputedImage.py y SignalRecognizer.py.

Para todas las imágenes de train:

-Lee la imagen.

-Pasa la imagen a niveles de gris.

-Ecualiza los niveles de gris de la imagen.

-Cambia el tamaño de la imagen a 32x32.

-Extrae los vectores de características con HOG, los convierte en ComputedImage y añade ComputedImage a una lista.

Devuelve una lista de computedImage con los atributos belonging\_class, que es la clase a la que pertenece la imagen, y characteristics\_vector, que es el vector de características calculado de la imagen.

Se pasa a la función que va a clasificar la lista de vectores de características de train y las etiquetas de todas las imágenes de train.

Crea el fichero donde se va a escribir el resultado.

Se convierte la lista de vectores de características en una matriz.

LDA se entrena con la matriz de vectores de características y los transforma dejando su dimensionalidad en 42 (número de clases menos uno).

Se entrena el clasificador bayesiano con el resultado del LDA y las etiquetas de las imágenes de entrenamiento. (En este caso el LDA y el clasificador bayesiano son el mismo tipo de objeto ya que sklearn.discriminant\_analysis permite realizar ambas funciones).

Para cada imagen de test:

-Guarda el nombre de la imagen y la etiqueta en una lista cada una.

-Lee la imagen.

-Pasa la imagen a niveles de gris.

-Ecualiza los niveles de gris de la imagen.

-Cambia el tamaño de la imagen a 32x32.

-Extrae el vector de características.

-Lo añade a la lista de vectores de características de test.

Convierte la lista de vectores de características de test en una matriz y reduce con LDA su dimensionalidad.

Se clasifican las imágenes de test con el clasificador bayesiano.

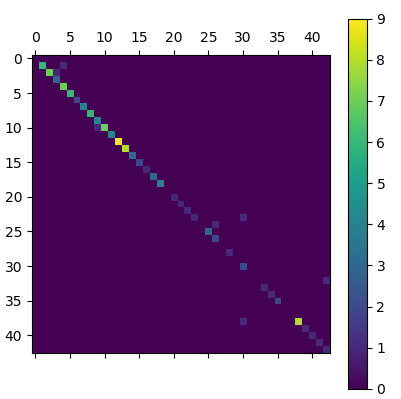
Por último, se escribe en el fichero creado al principio la etiqueta que le ha asignado el clasificador a cada imagen junto con el nombre de la imagen.

## Estadísticas obtenidas.

\* Para estas estadísticas, se han usado la carpeta de entrenamiento y de test subidas por el profesor (una batería de 121 imágenes recortadas). Para la tasa de error se ha utilizado el método de exclusión.

**1.a. Matriz de confusión:**

Para esta opción, hemos obtenido una matriz en la que tenemos 9 elementos que no están en la diagonal, cuyo valor es distinto de 0. Esto quiere decir que, respecto de sus clases reales, se han clasificado **7 resultados de forma incorrecta** de **121 muestras**, por lo que el **% de acierto es de 94.21%**. La **tasa de error es de 5.79%.**



Como podemos observar, el clasificador se confunde en las clases 1, 2, 10, 23, 24, 32 y 38

**1.b. F1 Score:**

Imagen que contiene captura de pantalla

Descripción generada automáticamente

**1.c. Precisión:**

Imagen que contiene captura de pantalla

Descripción generada automáticamente

# Ejercicio 2.

## 2. LBP

|  |  |
| --- | --- |
| Opción: | --classifier LBP-LDA-Bayes |

Clases implicadas= CharacteristicsExtractorLBP.py, main.py, ComputedImage.py y SignalRecognizerLBP.py.

### Algoritmo.

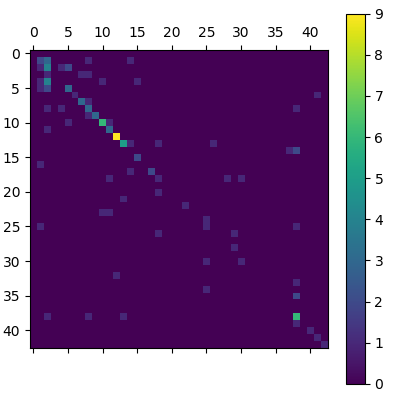
El algoritmo es igual al del ejercicio uno solo que en vez de usar el histograma de orientación de gradientes utiliza LBP como vector de características con 16 vecinos circularmente simétricos y un radio circular de 11.

### Estadísticas obtenidas.

\* Para estas estadísticas, se han usado la carpeta de entrenamiento y de test subidas por el profesor (una batería de 121 imágenes recortadas). Para la tasa de error se ha utilizado el método de exclusión.

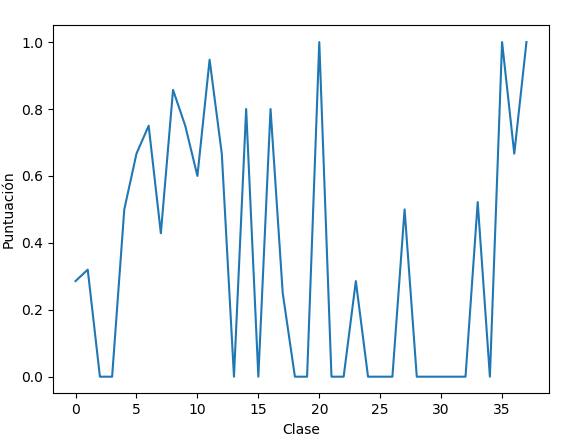
**2.a. Matriz de confusión:**

De la misma forma que en el caso anterior, si hacemos la matriz de confusión habiendo utilizado para conseguir los vectores de características la función de LBP (Local Binary Patterns), obtenemos **62 clasificaciones erróneas**, por lo que nuestra tasa de fallos ha aumentado. El **% de acierto es de 48.77%.** La **tasa de error es de 51.23%.** Al ser la tasa de error superior al azar consideramos que no es una buena opción.

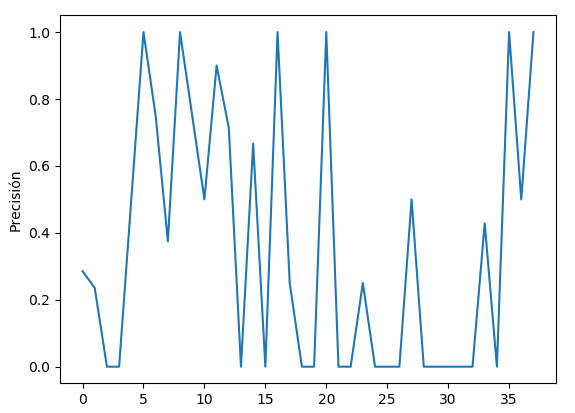


En este caso, el clasificador se confunde en una gran cantidad de clases

**2.b. F1 Score:**



**2.c. Precisión:**

****

## 3. PCA

|  |  |
| --- | --- |
| Opción: | --classifier PCA-Bayes |

Clases implicadas= CharacteristicsExtractor.py, main.py, ComputedImage.py y SignalRecognizerPCA.py.

### Algoritmo.

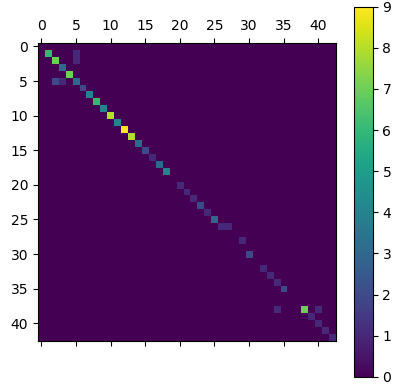
El algoritmo es igual al del ejercicio uno solo que en vez de usar LDA para la reducción de dimensionalidad utiliza PCA con el mismo número de componentes que LDA ,42 (el número de clases menos 1). En PCA se le pasa las etiquetas, pero como es no supervisado las ignora.

### Estadísticas obtenidas.

\* Para estas estadísticas, se han usado la carpeta de entrenamiento y de test subidas por el profesor (una batería de 121 imágenes recortadas). Para la tasa de error se ha utilizado el método de exclusión.

**3.a. Matriz de confusión:**

En este caso, si para la reducción de dimensionalidad usamos PCA (Principal Component Análisis), obtenemos **9 clasificaciones erróneas,** peor que con LDA. Por lo tanto, su **% de acierto es del 66.12%.** La **tasa de error es de 7.43%.**



Las clases que confunde son la 1, 2, 5 (tres veces), 26, 28 y 38 (dos veces). Alguna de ellas las confunde más de una vez

**3.b. F1 Score:**

Imagen que contiene captura de pantalla

Descripción generada automáticamente

**3.c. Precisión:**

Imagen que contiene captura de pantalla

Descripción generada automáticamente

## 4. KNN

|  |  |
| --- | --- |
| Opción: | --classifier LDA-KNN |

### Algoritmo.

Clases implicadas= CharacteristicsExtractor.py, main.py, ComputedImage.py y SignalRecognizerKNN.py.

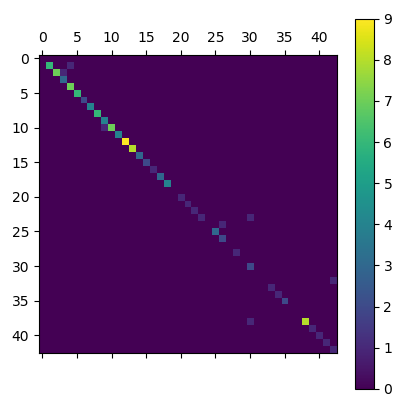
El algoritmo es igual al del ejercicio uno solo que en vez de usar un clasificador bayesiano se utiliza uno de k vecinos con k=6 después de haber probado con varios k siendo este el que mejor resultado daba.

### Estadísticas obtenidas.

\* Para estas estadísticas, se han usado la carpeta de entrenamiento y de test subidas por el profesor (una batería de 121 imágenes recortadas). Para la tasa de error se ha utilizado el método de exclusión.

**4.a. Matriz de confusión:**

Utilizando otro clasificador supervisado, como el K-Neighbors, también se obtienen buenos resultados. De 121 imágenes, solo **ha fallado clasificando 7** de ellas. El **% de acierto** en este conjunto de imágenes **es de 94.21%**. La **tasa de error es de 5.79%.**

****

Podemos ver que se confunde en las clases 1, 2, 10, 23, 24, 32 y 38

**4.b. F1 Score:**

Imagen que contiene captura de pantalla

Descripción generada automáticamente

**4.c. Precisión:**

Imagen que contiene captura de pantalla

Descripción generada automáticamente

# Problemas encontrados.

Entre algunos de los problemas que nos hemos encontrado, uno ha sido con vectores de características. No estábamos usando bien el HOG, y nos devolvía vectores que tenían distinta longitud, por lo que a la hora usar estos vectores, nos daban múltiples errores.

Otro problema que nos surgió fue a la hora de usar LBP para obtener los vectores de características. Tuvimos que probar variando los parámetros P y R(radio) hasta que dimos con uno que mejoraba un poco sobre el resto. Aun así, no es uno de los mejores resultados.

Para entrenar y reducir la dimensión, tuvimos que dedicar tiempo para entender lo que hacía exactamente, ya que no estábamos seguros del siguiente paso. No sabíamos que hacer exactamente con las imágenes de test, si teníamos que reducir sus vectores de características también, si había que hacer fit o no… Al final entendimos en clase con la aclaración del profesor.