Práctica Obligatoria 2

Visión artificial

Nicolae Alexe y Diego Méndez (Grupo N)

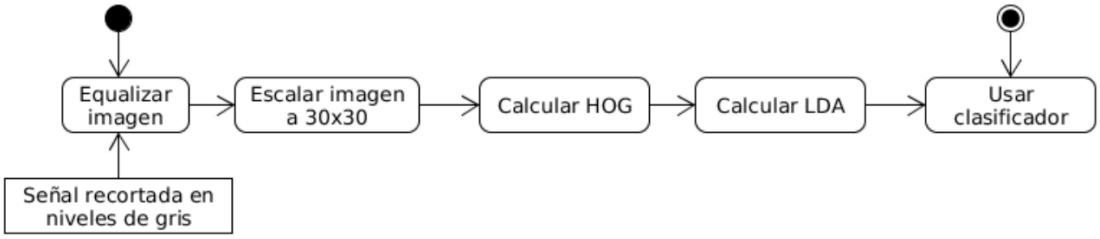
Junio 2019

Propuesta de trabajo

Se desea construir un sistema que permita el reconocimiento de ciertas señales de tráfico en imágenes realistas (tomadas en la calle desde un coche). Siguiendo un enfoque Top-Down, el alumno deberá construir dos funciones:

- Función de aprendizaje: La primera función recibirá el nombre de un directorio que contendrá las imágenes de aprendizaje. La función utilizará dichas imágenes para entrenar un clasificador que posteriormente será usado en la siguiente función.
- Función de reconocimiento: Esta segunda función recibirá un clasificador entrenado y la imagen recortada de una señal. La función devolverá una cadena con el nombre de la clase que predice el clasificador.

Esquema de funcionamiento del sistema de reconocimiento



1. Extracción del vector de características.

Una vez ejecutamos el proyecto llevamos a cabo la primera fase del sistema de reconocimiento, es decir, la extracción del vector de características y de etiquetas de las imágenes. Para ello hemos creado la clase Classifier que recibe cómo parámetro el tipo de clasificador que se desea utilizar (están implementados LDA-BAYES y PCA-BAYES).

La función "start" de esta clase comienza con una llamada a la función "classify" que recorre tanto los directorios de train como los de test preparando cada imagen (funciones "prepareImage" y "equalize" del fichero utils.py. Se realiza la ecualización y escalado de la imagen para proceder después con la extracción de HOG (Histograma de Orientación de Gradientes). Hemos utilizado la clase cv2.HOGDescriptor y el método "compute" cómo se puede ver en el ejemplo implementado por el profesor de la asignatura.

2. Reducción de la dimensionalidad

Una vez hemos obtenido el vector de características de cada imagen y las etiquetas de entrenamiento llevamos a cabo la reducción de la dimensionalidad. Para este proceso hemos implementado la función "reduce_dimensionality" (Classifier), que llama a las función "fit_transform" del clasificador, pasándole los vectores de características y las etiquetas o sólo los vectores dependiendo del clasificador (LDA, PCA respectivamente).
Antes de este proceso llamamos a la función "reshapeList" ya que sklearn requiere datos de forma (n_filas, n_columnas).
Por último devolvemos los datos en float 32 ya que HOG es float 32 y el transform de LDA es float 64.

3. Entrenamiento del clasificador Bayesiano con Gaussianas

Después de reducir la dimensionalidad de los datos procedemos al entrenamiento del clasificador. En el caso de LDA hemos aprovechado que LinearDiscriminanAnalysis de sklearn ofrece la reducción de dimensionalidad de LDA (con los métodos fit y transform) y además proyección LDA + clasificación con un Bayesiano con Gaussianas (fit y predict). Para PCA hemos utilizado el clasificador sklearn.naive_bayes.GaussianNB de sklearn.
En ambos casos utilizamos la función de la clase Classifier "train_classifier" que ejecuta las funciones "fit" y "predict" y obtenemos la precisión del clasificador para los ejemplos de test.

4. Uso del clasificador

Una vez entrenado el clasificador seleccionado procedemos a preparar los ejemplos de test (función "prepare_test" de la clase Classifier) realizando el reshape de la lista y aplicando el método "transform" del clasificador.
Por último llamamos a la función "predict" y obtenemos los resultados del clasificador, mostrando su precisión para los ejemplos de test.

5. Obtención de gráficos y estadísticas

Con todos los resultados obtenidos pasamos a mostrar diferentes gráficos como el porcentaje de aciertos y fallos del clasificador para el entrenamiento y el test, la matriz de confusión y el F1_Score del test.
Para ello hemos creado la clase Graphics y la hemos utilizado en el fichero main.py. Más adelante visualizaremos ejemplos de su ejecución en la comparativa entre LDA y PCA.

6. Escritura en el fichero de resultados

Escribimos los resultados del clasificador en el fichero resultados.txt con el formato requerido por los profesores de la asignatura mediante la función de utils.py "writeInFile".

```
18-00008.ppm ; 18
07-00018.ppm ; 07
38-00056.ppm ; 38
23-00016.ppm ; 01
10-00004.ppm ; 10
10-00010.ppm ; 10
38-00081.ppm ; 05
13-00025.ppm ; 13
25-00010.ppm ; 25
16-00002.ppm ; 16
12-00052.ppm ; 12
09-00012.ppm ; 09
02-00014.ppm ; 02
01-00020.ppm ; 01
23-00014.ppm ; 01
02-00001.ppm ; 02
07-00026.ppm ; 07
```

LDA VS. PCA

Hemos decidido implementar los clasificadores LDA (Linear Discriminant Analysis) y PCA (Principal Component Analysis).

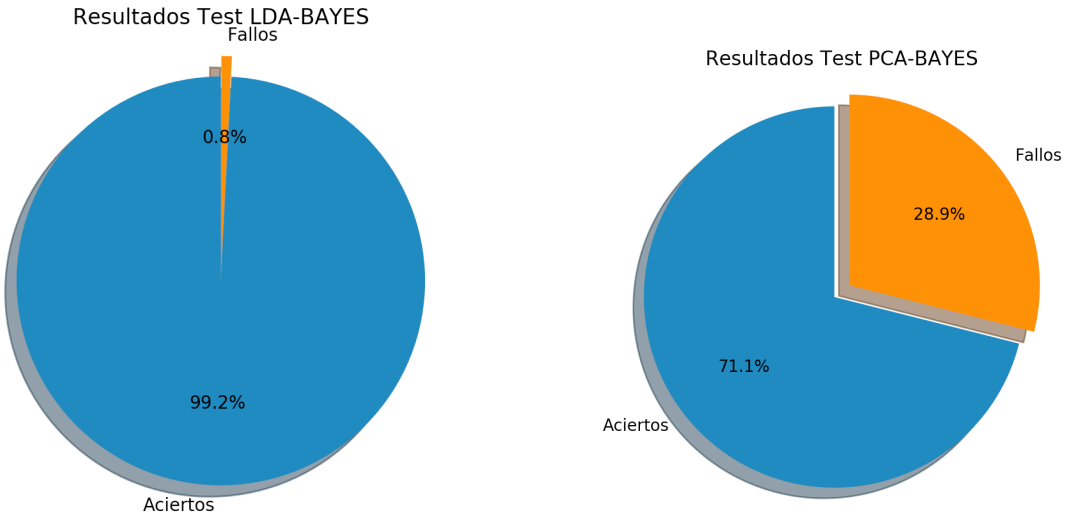
En ambos casos mostramos la precisión de la predicción del entrenamiento y su porcentaje, como la precisión era el test y su F1_Score.

Aquí podemos ver los resultados de ambos (LDA Y PCA respectivamente):

```
Clasificador: LDA-BAYES
Clasifier initialized!
/Users/Diegomendez1997/Coding/practica_va2/venv/lib/python3.7/site
  warnings.warn("Variables are collinear.")
Precisión de la predicción del entrenamiento: 1.0 - 100.0%
Precisión de la predicción del test: 0.9917355371900827 - 99.17%
F1_score test samples: 0.9917355371900827
Reconocimiento finalizado. Resultados escritos en resultados.txt
```

```
Clasificador: PCA-BAYES
Clasifier initialized!
Precisión de la predicción del entrenamiento: 0.9799054373522459 - 97.99054373522459%
Precisión de la predicción del test: 0.7107438016528925 - 71.07%
F1_score test samples: 0.7107438016528925
Reconocimiento finalizado. Resultados escritos en resultados.txt
```

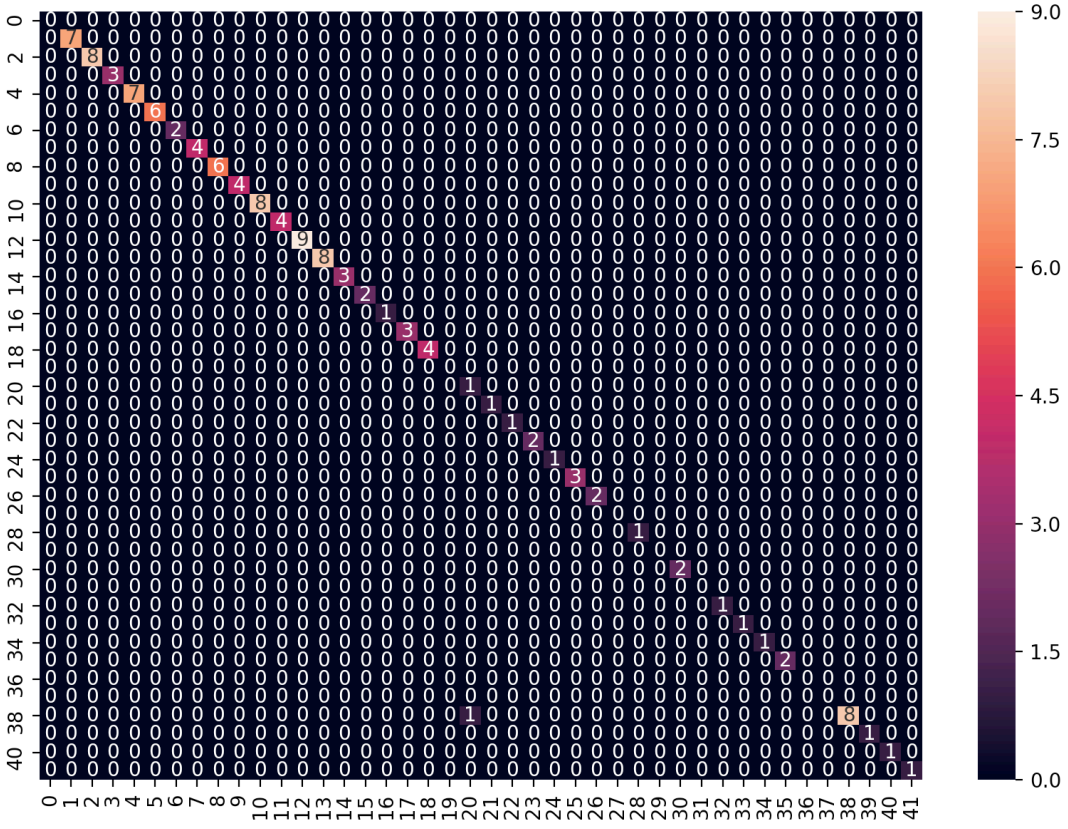
Como hemos dicho anteriormente hemos creado la clase Graphics para mostrar diferentes gráficos y estadísticas del clasificador.
Aquí podemos ver la comparativa en el porcentaje de aciertos y fallos de cada clasificador.

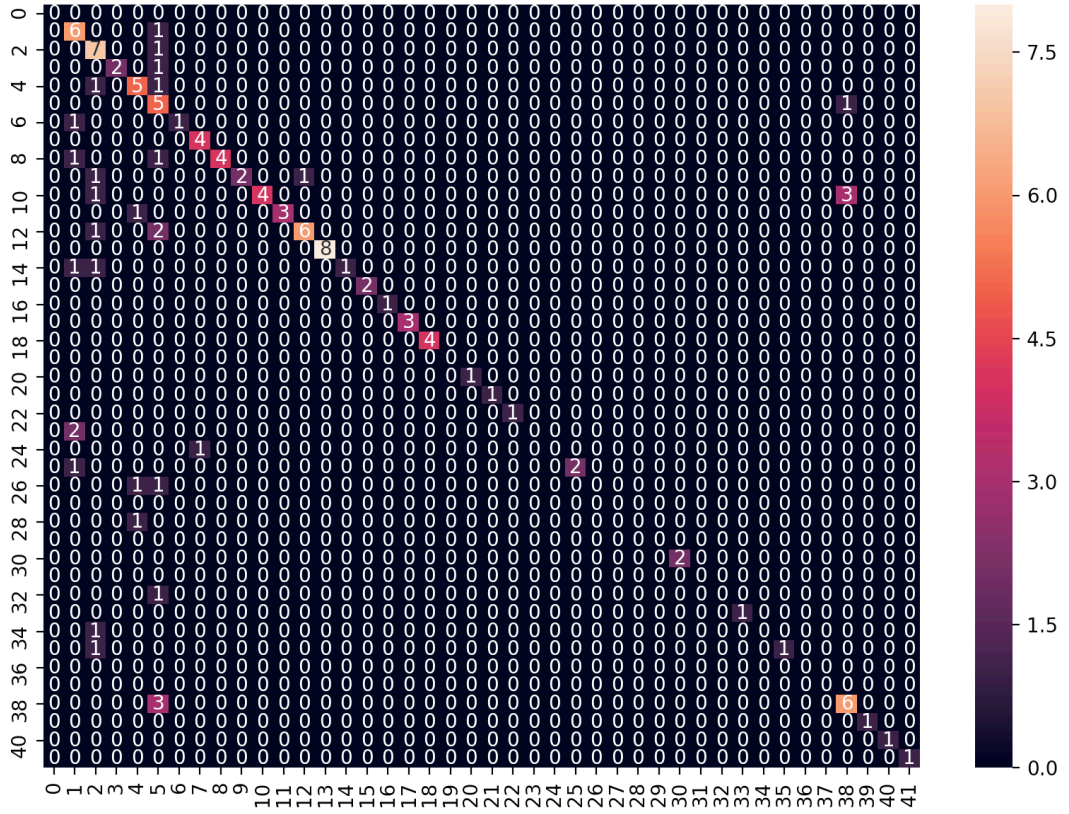


Y también sus matrices de confusión, siempre LDA Y PCA respectivamente.

La matriz de confusión es una herramienta que permite la visualización del desempeño de un algoritmo que se emplea en aprendizaje supervisado.

Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real. Uno de los beneficios de las matrices de confusión es que facilitan ver si el sistema está confundiendo dos clases.





Con todos estos datos podemos afirmar que obtenemos mejor resultado con LDA.

INTEGRACIÓN CON LA PRACTICA 1

Para poder aplicar el clasificador a escenas reales, se ha integrado el detector de señales con el algoritmo de clasificación, de manera que el detector nos devuelve una lista con las imagenes correspondientes a señales.

A partir de este punto no teníamos muy claro como proceder a la clasificacion, y tras probar diferentes alternativas sin mucho éxito decidimos dejar esta funcionalidad para el futuro, por lo que actualmente solo esta disponible la funcionalidad de detección integrada en el main.py.

Para que esta detección se ejecute debemos pasarle al script main.py el parámetro img_path con la ruta de la imagen.

EJECUCIÓN

Podemos ejecutar el programa sin ningún tipo de parámetro debido a que estos tienen valores por defecto para facilitar una prueba rápida del código.

En caso de que queramos cambiar los parámetros, podemos suministrar 4:

- train_path: Ruta de la carpeta con los archivos de entrenamiento
- test_path: Ruta de la carpeta con los archivos de test
- -classifier: Tipo de clasificador (2 opciones LDA-BAYES y PCA-BAYES)
- img_path: Ruta de la imagen que queramos que analice aparte de los archivos de test.

BIBLIOGRAFÍA

- Sklearn
- LDA
- PCA
- HOGDescriptor
- Pandas
- Seaborn