|  |
| --- |
| VISIÓN POR COMPUTADOR |
| SISTEMA DE RECONOCIMENTO |
|  |

|  |
| --- |
| Fabregat Jaén, Marc |

Índice

Introducción ………………………………………………………………… 1

Extracción de características ……………………………………………. 1

Preprocesamiento, segmentación y extracción ………………. 1

Análisis y selección de características …………………………. 2

Diseño de clasificadores …………………………………………………. 3

Naive Bayes ………………………………………………………….. 3

SVM ……………………………………………………………………. 3

Clustering (K-NN) …………………………………………………… 4

Árboles de decisión ………………………………………………... 4

Análisis de los resultados y conclusiones …………………………… 5

Sistema de reconocimiento de caras ………………………………….. 6

Algoritmo AdaBoost ……………………………………………….. 6

Descriptores de Haar ………………………………………………. 6

ANEXO I – Programas

Extractor de características

Clasificador bayesiano

Clasificador SVM

Clasificador Clustering (K-NN)

Clasificador árboles de decisión

Reconocimiento de caras (Descriptores de Haar)

ANEXO II – Análisis de características

Introducción

El objetivo principal del trabajo es comparar diferentes tipos de clasificadores para ver sus diferencias. De forma indirecta, esto conlleva a diseñar todo un sistema de reconocimiento completo con cada una de sus etapas. En concreto, se diseñarán y programarán las etapas de preprocesamiento, segmentación, extracción de características y 4 clasificadores diferentes.

Se ha elegido una aplicación para el sistema bastante realista: un sistema de reconocimiento y clasificación de frutas y verduras. Las imágenes con las que se va a trabajar se han obtenido de la base de datos de imágenes de la Universidad de Ámsterdam: [ALOI](https://aloi.science.uva.nl/). ALOI tiene 1000 objetos con más de 100 imágenes diferentes, en las que se varía el ángulo del objeto y la iluminación. Así se pueden conseguir situaciones de iluminación o posición realistas que ocurren en la práctica.

Personalmente me interesaba el ámbito del reconocimiento facial, por lo que, a modo de aplicación extra, se ha diseñado un sistema de reconocimiento de caras utilizando Descriptores de Haar.

Para programar e implementar los sistemas se ha utilizado C++ con la librería open source OpenCV, la cual nos facilita enormemente todo al disponer de la gran mayoría de funciones necesarias.

Extracción de características

Preprocesamiento, segmentación y extracción

Se han juntado las 3 etapas de preprocesamiento, segmentación y extracción de características en un solo programa, ya que las imágenes que nos ofrece la Universidad de Ámsterdam apenas necesitan procesado previo.

Para extraer las características de un objeto en una imagen, es necesario segmentarla y quedarse solo con el objeto, aislándolo del fondo u otros estorbos. Cuando la imagen está preparada, se pueden calcular los atributos que definen el objeto matemáticamente mediante varios métodos para luego poder ser importados en el diseño y uso de un clasificador.

El programa diseñado extrae las siguientes características: área, perímetro, centro de gravedad, límites horizontales y verticales, compacidad, excentricidad, eje principal de inercia, longitud, anchura y tono medio. Genera 3 archivos con los datos calculados en diferentes formatos: .txt, .arff (Weka) y .csv.

El código del programa se encuentra en el Anexo I, pero se va a explicar brevemente:

El programa se ha diseñado de forma que calcula las características para una sola clase, es decir, se va a tener que ejecutar tantas veces como clases se tengan, cambiando el nombre de la clase cada vez. Necesita como entrada las imágenes que va a analizar, el nombre de la clase y un archivo imágenes.txt con el nombre de cada imagen en diferentes líneas. Se hace uso de una estructura “características” donde se almacenan todos los atributos de la imagen.

Se hace un bucle para cada línea leída del archivo imágenes.txt y en cada iteración se lee la imagen que corresponde al nombre de la línea leída. Luego, se llama a la función ExtraccionCaracteristicas, la cual se explicará a continuación, y se escribe cada archivo con su formato correspondiente.

La función ExtraccionCaracteristicas llama inmediatamente a la función PreprocesadoSegmentacion. Esta segunda función convierte la imagen RGB (BGR en OpenCV) en HSV y hace una umbralización del canal Value con un umbral 25. Es decir, pone a 0 los valores más oscuros y a 255 los que están por encima de 25. Esto se hace porque las imágenes que usamos siempre están en un fondo negro, lo cual facilita enormemente la segmentación. Luego, se realiza un doble opening con una máscara circular de 5x5 para eliminar las “salidas” que aparecen en los bordes y el posible ruido. Sin embargo, en algunas imágenes quedaban huecos negros (0) dentro del objeto demasiado grandes para cerrarlos con un closing sin deformar en exceso el objeto, por lo que se recurre a calcular el contorno para posteriormente rellenarlo de blanco (255).

El resto de la función ExtraccionCaracteristicas utiliza fórmulas y funciones implementadas en OpenCV para calcular los atributos de la imagen.

Análisis y selección de las características

El extractor diseñado calcula 14 características diferentes para cada imagen. Si se utilizasen las 14 características para diseñar el clasificador, se obtendría un clasificador demasiado complejo, con un tiempo de computación mayor y menos eficiente.

Como solución, lo aconsejable es seleccionar las características que mejor separen los datos. Se pueden seguir varios métodos para determinar la selección de atributos óptima, como el test χ2 o el análisis mediante el ratio de Fisher. En este caso se ha utilizado el Coeficiente de Separación entre Clases para cada Característica, el cual nos indica si se produce una superposición de las distribuciones gaussianas (cuando es negativo) o la separación entre ellas (cuando es positivo).

Para la aplicación elegida, se han calculado las 76 imágenes de entrenamiento para cada una de las 8 clases de frutas y verduras. Esto hace un total de 608 imágenes a las que se les ha extraído las características del objeto. En el anexo II se muestra el Coeficiente de Separación entre Clases para cada Característica para cada posible pareja de clases.

Para esta aplicación en concreto se han seleccionado 2 características: el área y la anchura. Si se eligen más características, los clasificadores tienen una precisión del 100%, haciendo imposible su comparación entre ellos, que es el objetivo de este proyecto. Es decir, se han elegido solo 2 características, pero se debe tener en cuenta que no es el número óptimo de características a escoger.

Se han elegido estos 2 atributos para los 4 clasificadores, aunque no se mencionará explícitamente cuando se estén explicando.

Diseño de clasificadores

Naive Bayes

El primer clasificador diseñado es el clasificador de Naive Bayes, el cual está fundamentado en el teorema de Bayes. Es un clasificador probabilístico, que se basa en la distribución gaussiana de cada característica y cada clase para clasificar la muestra.

Conociendo la probabilidad a priori de cada clase y el valor x de cada característica de la muestra, el clasificador calculará la probabilidad de x condicionado a cada clase. Luego usando la expresión siguiente, calculará la probabilidad de que la muestra pertenezca a la clase condicionado a x:

Se hará el cálculo para cada clase y el clasificador elegirá como resultado la clase para la que la probabilidad sea mayor.

El código del programa se puede ver en el Anexo I, pero se explicará aquí:

Primero, se entrena al clasificador con los datos de entrenamiento que se han obtenido con el extractor de características ya mostrado. Se han utilizado 608 imágenes para entrenar al clasificador.

Luego, se llama a la función ClasificarImagenes, la cual lee las características de las imágenes clase por clase y da un resultado para cada una de ellas.

Para terminar, se llama a la función MatrizConfusion para mostrar los resultados en forma de matriz confusión y sus estadísticas de validación (TP rate, FP rate, precision, recall y F1-score). La función recorre la matriz de resultados y va sumando un contador dependiendo de la clase real de la muestra y la clasificación de esta. Luego, se calculan las estadísticas de validación del clasificador con fórmulas matemáticas.

SVM (Máquinas de Vector Soporte)

Es un clasificador basado en funciones de decisión. A partir de una hipótesis, se ajusta el modelo minimizando la función de coste, haciendo una aproximación lineal en dos tramos. La función calcula el hiperplano óptimo que separa las clases para cada característica. Esto lo hace mediante un kernel, que permite proyectar cada vector de los datos de entrenamiento sobre el espacio de características, dando lugar a los vectores soporte.

Estos vectores y el hiperplano calculado llevan asociados un margen, el cual idealmente separa completamente las clases. Pero esto sabemos que en la práctica no ocurre, no siempre se puede calcular un hiperplano que separe completamente las clases. Para solucionarlo se utiliza un coeficiente de regulación, el cual es una constante que intentará corregir el error provocado por la distancia del margen a la muestra que no está en su zona.

El código del programa se puede ver en el Anexo I, pero se explicará aquí:

El funcionamiento del programa es muy parecido al del clasificador bayesiano. La diferencia principal es que antes de entrenar al clasificador, este requiere unos parámetros de ajuste. Se selecciona el tipo de clasificador, C\_SVC, el cual es un clasificador SVM que permite establecer un coeficiente de regularización C, su valor es 0.1 para el clasificador diseñado. También es necesario especificar qué tipo de kernel se va a usar para proyectar los vectores soporte, en este caso se ha usado un kernel de intersección de histograma al ser bastante rápido y preciso.

El resto del código es similar al del clasificador bayesiano. Se llama a la función ClasificarImagenes para obtener los resultados y a MatrizConfusion para mostrar los resultados obtenidos. Ambas funciones son iguales a las del clasificador bayesiano.

Clustering (K-NN)

El clasificador de k-nearest neighbors o k-vecinos más cercanos está basado en la técnica de clustering (análisis de la agrupación). Explicado de una manera sencilla y a grandes rasgos, a partir de cada nueva muestra, el algoritmo selecciona las k muestras más cercanas a la nueva, de entre todas las que se han usado para entrenarlo. Luego, se hace un recuento de las clases a las que pertenecen las muestras seleccionadas y la clase más repetida es el resultado de la clasificación de la nueva muestra.

El código del programa se puede ver en el Anexo I, pero se explicará aquí:

El funcionamiento del programa es muy parecido al de los otros clasificadores. La mayor diferencia es que el clasificador K-NN requiere que se especifique el valor de K, 13 para el clasificador diseñado. Un valor de K grande nos permite reducir el ruido que pueda haber en la clasificación.

El resto del código es similar al de los otros clasificadores. Se llama a la función ClasificarImagenes para obtener los resultados y a MatrizConfusion para mostrar los resultados obtenidos. Ambas funciones son iguales a las de los otros clasificadores.

Árboles de decisión

El último clasificador que se diseñará está basado en árboles de decisión. Es un método que hace uso de funciones lógicas para clasificar la muestra. Para explicarlo se utiliza la metáfora de un árbol.

El algoritmo empieza en el nodo raíz. En cada nodo se hace una operación lógica (mayor que, menor que, igual...) con una característica y se pasa al siguiente nodo siguiendo una rama u otra dependiendo del resultado de la operación. Esto se repite de forma recursiva, siguiendo las ramas hasta llegar a una hoja, que representa la clase escogida como resultado de la clasificación.

El código del programa se puede ver en el Anexo I, pero se explicará aquí:

El funcionamiento del programa es muy parecido al de los otros clasificadores. Sin embargo, en este clasificador es necesario especificar si se “podará” al final. Para esta aplicación, al ser muy sencilla, con solo 2 características, se optará por no “podar” el árbol.

El resto del código es similar al de los otros clasificadores. Se llama a la función ClasificarImagenes para obtener los resultados y a MatrizConfusion para mostrar los resultados obtenidos. Ambas funciones son iguales a las de los otros clasificadores.

Análisis de los resultados y conclusiones

Se han puesto a prueba los 4 clasificadores diseñados con 80 imágenes de test, 10 imágenes de prueba por cada fruta o verdura. Los resultados se pueden ver en el Anexo III. En general, los resultados han sido buenos teniendo en cuenta que solo se han seleccionado 2 características, sin ser el número óptimo.

Para validar y comparar los clasificadores se van a tener en cuenta 3 aspectos:

* Precisión total: es el total de predicciones correctas hecha por el clasificador entre el número total de predicciones.
* F1-score total: es la media de las F1-score de todas las clases. Esta puntuación es la media armónica de la precision y el recall por clase.
* Tiempo de ejecución: el tiempo que tarda el programa en ejecutarse. Se ha tomado haciendo la media de 10 valores diferentes.

Para 80 muestras de test se ha obtenido:

|  |  |  |  |
| --- | --- | --- | --- |
| Clasificador | Precisión total | F1-score total | Tiempo de ejecución |
| Naive Bayes | 98.75% | 0.9875 | 24.44 ms |
| SVM | 97.5% | 0.975 | 31.59 ms |
| Clustering (K-NN) | 80% | 0.775 | 19.71 ms |
| Árboles de decisión | 98.75% | 0.9875 | 17.11 ms |

Como cabría esperar, los clasificadores más rápidos han sido el árboles de decisión, al tener solo 2 características con las que trabajar se trata de un árbol bastante sencillo, y el método de k vecinos más cercanos, ya que no necesita construir un modelo previo, si no que las clasificaciones se hacen comparando y accediendo directamente a los datos de entrenamiento. Sin embargo, de estos 2 clasificadores hay que descartar el clasificador basado en el método de clustering, ya que su precisión es muy baja respecto al resto.

Los otros 3 clasificadores han resultado ser muy efectivos y precisos, siendo los 2 mejores el clasificador bayesiano y el de árboles de decisión, ambos con una precisión del 98.75%, o en otras palabras, de las 80 frutas y verduras solo han clasificado incorrectamente 1.

Objetivamente, el mejor clasificador para esta aplicación es el de árboles de decisión. Ha obtenido la máxima precisión en el mínimo tiempo de ejecución.

Sistema de reconocimiento de caras

A modo de proyecto complementario extra se ha realizado un sistema de reconocimiento de caras. El sistema permite reconocer caras de personas y elementos faciales (ojos y boca). Se puede utilizar con imágenes o en tiempo real, ya sea con una cámara USB o un vídeo.

Para la aplicación de reconocimiento de caras, el clasificador que más se suele utilizar está basado en el algoritmo AdaBoost mediante cascadas de Descriptores de Haar. Es el que se ha implantado para este sistema. Se puede consultar el código del programa en el Anexo I. El programa necesita los archivos .xml que contengan los descriptores, los carga y detecta las coincidencias para cada descriptor.

Algoritmo AdaBoost

Es un proceso iterativo, en el cual se combinan clasificadores débiles para conseguir uno fuerte. Se hacen iteraciones, entrenando a los clasificadores débiles y ponderando la siguiente clasificación dependiendo de los resultados que se obtengan, se le da un mayor peso a los ejemplos mal clasificados y un menor peso a los correctos. Al final, se combinan los clasificadores débiles ponderados en función del error que hayan cometido, dando lugar a un clasificador fuerte.

Este algoritmo, junto con Descriptores de Haar, genera un esquema de clasificación en cascada muy efectivo para el reconocimiento de caras.

Descriptores de Haar

Es un tipo de descriptor que nos permite representar matemáticamente la imagen. Es el resultado de aplicar varios filtros direccionales a una imagen para cada escala y cada posición. Los filtros aplicados se pueden representar como combinaciones de rectángulos blancos y negros. Siendo el negro las zonas de contribución positiva al filtro y el blanco las zonas de contribución negativa.

El problema de usar los Descriptores de Haar como clasificadores es la generación de estos. Es un proceso computacionalmente muy exigente, ya que para una imagen de 24x24 píxeles, la cual es minúscula en comparación a los estándares modernos, y usando 5 filtros se generan 162336 características. Esto, combinado a que para poder ser usados como clasificadores efectivos es necesario obtener los descriptores de miles de imágenes, hacen que el proceso de generación pueda llevar muchas horas.

Para poder usar los Descriptores de Haar en un clasificador, es necesario emplear una cascada de descriptores para cada elemento que queramos reconocer (cara, ojos, nariz, etc…). Otro aspecto que cabe tener en cuenta es que estos clasificadores pueden ser lentos. Por ejemplo, para un reconocimiento en tiempo real, si se está trabajando con resoluciones grandes y el procesador usado no es lo suficientemente rápido, el algoritmo no es capaz de seguir el ritmo a los fotogramas por segundo del vídeo, lo que obliga a reescalar el vídeo a una resolución más pequeña