



IDENTIFICACIÓN DE TEXTURAS EN UNA IMAGEN DIGITAL

Regiones: Cielo, Bosque y Suelo

Objetivo

Diseñar un clasificador paramétrico bayesiano que identifique tres regiones o áreas en una imagen digital: “Cielo, Bosque y Suelo”



Instituto Politécnico Nacional

Escuela Superior de Cómputo



Unidad de Aprendizaje: Pattern Recognition

Práctica 1: Reporte

Unidad Temática: Diseño de un Clasificador Paramétrico Bayesiano para una distribución normal
(Distancia Euclidiana)

Grupo: 3CM11

Profesora: Cruz Meza María Elena

Fecha de entrega: 5 de mayo de 2021

Integrantes:

Arévalo Andrade Miguel Ángel

Medero Luján Alejandro

Castro Cruces Jorge Eduardo

Franco Ake Alan

Introducción	4
Marco teórico	4
Diseño de un Clasificador Paramétrico Bayesiano para una distribución normal (Distancia Euclidiana)	5
Preprocesamiento	6
Segmentación	7
Multi Umbralización	9
Operación Lógica “OR”	11
Desarrollo	12
Clasificador Paramétrico Bayesiano para una distribución normal (Distancia Euclidiana)	12
Fase de aprendizaje	12
Fase de recuperación	15
Implementación	17
Pruebas y resultados	18
Conclusiones	19
Fuentes consultadas	20

Introducción

En el caso de estudio, que se trata en este documento, se desea resolver un problema diseñando un clasificador basado en la distancia euclídea que permita identificar o clasificar tres tipos de texturas naturales, es decir, tres regiones en una imagen digital (ver fig. 1) son: cielo, tierra, zonas boscosas. En la imagen se distinguen el predominio de áreas verdes y zonas sin Cultivos, en color ocre, a partir de una imagen de entrada, basadas en tres componentes de color (R, G, B)



Fig. 1 imagen $I(x,y)$, con tres tipos de texturas

Marco teórico

La clasificación consiste en asignar clases o etiquetas a objetos. Hay dos tipos básicos de de clasificación:

No supervisada: en este caso las clases son desconocidas, por lo que el problema consiste en dividir un conjunto de objetos en n grupos o clusters, de modo que se asigne una clase a cada grupo diferente. También se le conoce como *clustering*.

Supervisado: las posibles clases o etiquetas se conocen a priori, y el problema consiste en encontrar un función o regla que asigne cada objeto a uno de las clases.

Clasificación probabilística

La clasificación supervisada consiste en asignar a un objeto particular descrito por sus atributos, $A1, A2, \dots, An$, una de las m clases, $C = \{c1, c2, \dots, cm\}$, tal que la probabilidad de la clase dados los atributos se maximiza:

Evaluación del clasificador

Precisión: se refiere a qué tan bien un clasificador predice la clase correcta para ejemplos invisibles (es decir, aquellos no considerados para aprender el clasificador).

Tiempo de clasificación: cuánto tarda la clasificación. proceso para predecir la clase, una vez que el clasificador ha sido entrenado.

Tiempo de formación: cuánto tiempo se necesita para aprender clasificador de datos.

Requisitos de memoria: cuánto espacio en términos de memoria se requiere para almacenar los parámetros del clasificador.

Claridad: si el clasificador es fácilmente comprensible para una persona.

Desequilibrio de clase

En general queremos maximizar la clasificación precisión; sin embargo, esto solo es óptimo si el costo de un la clasificación incorrecta es la misma para todas las clases

Cuando existe un desequilibrio en los costos de clasificación errónea, debemos minimizar el esperado costo (EC). Para dos clases, esto viene dado por:

$$EC = FN \times P(-)C(- | +) + FP \times P(+)C(+ | -)$$

Donde: FN es la tasa de falsos negativos, FP es el falso tasa positiva, $P(+)$ es la probabilidad de positivo, $P(-)$ es la probabilidad de negativo, $C(- | +)$ es el costo de clasificando un positivo como negativo, y $C(+ | -)$ es el costo de clasificar un negativo como positivo

Clasificador Bayesiano

Aplicación de la regla de Bayes:

$$P(C | A1, A2, \dots, An) = \frac{P(C)P(A1, A2, \dots, An | C)}{P(A1, A2, \dots, An)}$$

Que se puede escribir de forma más compacta como:

$$P(C | A) = P(C)P(A | C)/P(A)$$

El problema de clasificación se puede formular como:

$$\text{ArgC}[Max[P(C | A) = P(C)P(A | C)/P(A)]]$$

Equivalente:

- $\text{ArgC}[Max[P(C)P(A | C)]]$
- $\text{ArgC}[Max[\log(P(C)P(A | C))]]$
- $\text{ArgC}[Max[(\log P(C) + \log P(A | C))]]$

Tenga en cuenta que la probabilidad de los atributos, **P(A)**, no puede variar con respecto a la clase, por lo que puede ser considerada como una constante para la maximización.

Complejidad

La aplicación directa de la regla de Bayes da como resultado un problema computacionalmente costoso

El número de parámetros en el término de probabilidad, $P(A_1, A_2, \dots, A_n | C)$, aumenta exponencialmente con el número de atributos.

Una alternativa es considerar ciertas propiedades como en los modelos gráficos, en particular que todos los atributos son independientes dada la clase, lo que resulta en el clasificador bayesiano ingenuo.

El clasificador bayesiano simple o ingenuo (NBC) se basa en el supuesto de que todos los atributos son independiente dada la variable de clase:

Naive Bayes

El clasificador bayesiano simple o ingenuo (NBC) se basa en el supuesto de que todos los atributos son independiente dada la variable de clase:

La fórmula ingenua de Bayes reduce drásticamente la complejidad del clasificador bayesiano, como en este caso solo requieren la probabilidad previa (unidimensional vector) de la clase, y las n probabilidades condicionales de cada atributo dada la clase (bidimensional matrices)

El requisito de espacio se reduce de exponencial a lineal en el número de atributos

El cálculo de la parte posterior se simplifica enormemente, ya que para estimarlo (no normalizado) solo se realizan n multiplicaciones requerido

Modelo gráfico

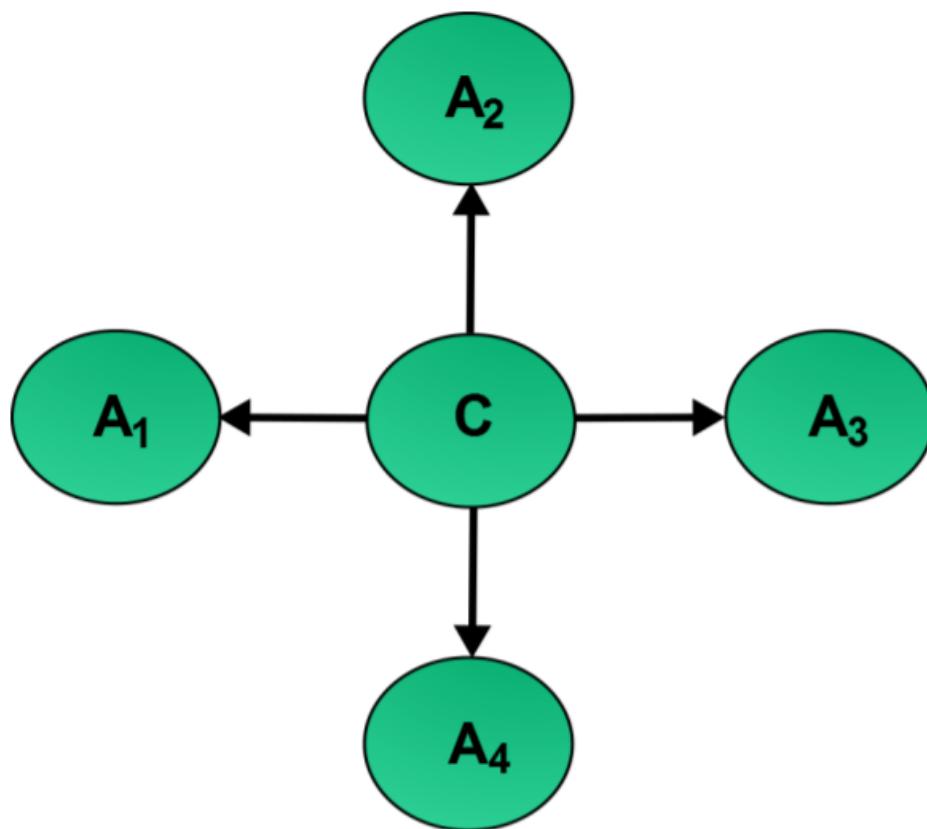


Fig. 2 Modelo gráfico del clasificador bayesiano simple

Aprendizaje de parámetros

Las probabilidades se pueden estimar a partir de datos utilizando, por ejemplo, estimación de máxima verosimilitud.

Se dan las probabilidades previas de la variable de clase C por:

$$P(c_i) \sim N_i/N$$

Las probabilidades condicionales de cada atributo, A_j , pueden ser estimadas como:

$$P(A_{jk} | c_i) \sim N_{jki}/N_i$$

Inferencia

La probabilidad posterior se puede obtener simplemente por multiplicar el anterior por la probabilidad de cada atributo.

Dados los valores de m atributos, a_1, \dots, a_m , para cada clase c_i , el posterior es proporcional a:

$$P(c_i | a_1, \dots, a_m) \sim P(c_i)P(a_1 | c_i) \dots P(a_m | c_i)$$

La clase c_k que maximiza la ecuación anterior debe ser seleccionada.

Diseño de un Clasificador Paramétrico Bayesiano para una distribución normal (Distancia Euclidiana)

Dada una imagen (fig. 1), en ella podemos distinguir tres tipos de regiones o áreas de interés, las cuales pueden describirse por los colores de los píxeles que conforman tales regiones a las que denominaremos clases C_i :

- o Área Cielo, C_1
- o Área Boscosa, C_2
- o Área Suelo, C_3

De las tres clases o regiones notemos que se tiene datos a priori, de antemano sabemos que cada píxel de $f(x,y)$ corresponderá a un patrón x_i . Cada patrón x_i corresponderá tiene una pertenencia a una clase, es decir $x_i \in C_i$, donde $C_i = \{C_1, C_2, C_3\}$. Además, dada la naturaleza de cada píxel, podemos considerar la textura como el rasgo principal de cada región o área a identificar. Con estos datos es posible diseñar un clasificador (fig. 2) que permita identificar a cada una de las clases.

Los elementos que intervienen para este proceso se listan a continuación:

- Preprocesamiento de la imagen
 - Abrir imagen
 - Convertirla a ng
 - (Extracción del canal verde
 - Extraer canales en RGB)
- Segmentación
 - Multi Umbralización
 - Operación puntal OR
 - Generación del descriptor de texturas para cada clase C_i
 - Asociación de cada píxel $f(x,y)$ a C_i o Definición del CMA
- Aprendizaje
 - Vector de entrada (r,g,b)
 - Cálculo del Centroide o vector media Z_i para cada clase C_i
 - Algoritmo BM
 - Pruebas
- Implementación del Clasificador

- Interfaz gráfica
- Extracción de características
- Clasificador

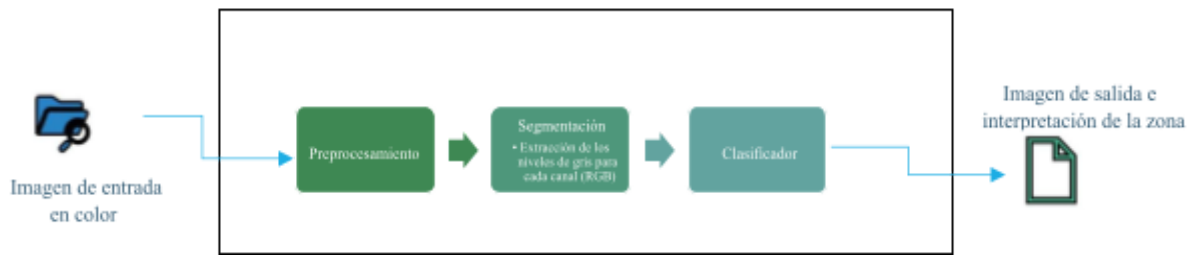


Fig. 3 Esquema general del diseño del Clasificador

Preprocesamiento

Dada la imagen de estudio para resolver el problema, se requiere abrir la imagen para poder extraer los valores de los píxeles, es decir, extraer la textura de cada región o clase ya que se llevarán a cabo las pruebas necesarias para definir si este primer rasgo es el que puede describir adecuadamente a los patrones de cada clase, posteriormente se someterán a la regla de aprendizaje para finalmente llevar a cabo la implementación del clasificador que resuelva el problema de la identificación de las tres regiones o clases.

Especificaciones de la imagen de entrada:

- o Imagen en color (R, G, B), a la que denominaremos $f(x,y)$, donde el rango de valores para cada canal va de $[0-255]$ niveles de color.
- o Píxel (r,g,b), que llamaremos patrón x_i

 C_i o Tamaño de la imagen: $M \times N = 334 \times 218$

Definición del vector de entrada (r,g,b)

La descripción de cada región o clase será mediante el análisis de su textura, por lo que se necesita extraer cada píxel, estos conformarán nuestro vector de entrada para el clasificador.

Píxel (x,y) para cada canal (r,g,b) $\rightarrow x_i = (r, g, b) \rightarrow x_i =$

Para clase C_i , se requiere almacenar los píxeles que conforman ese tipo de textura, por ejemplo, tomemos un píxel cualquiera para cada clase, la representación del vector de rasgos que describe a cualquier patrón para cada una de las clases tendrá la siguiente forma:

- Un patrón de la clase suelo: $x_i \Rightarrow (r, g, b) \rightarrow x_i = \begin{pmatrix} r \\ g \\ b \end{pmatrix} = \begin{pmatrix} 200 \\ 160 \\ 120 \end{pmatrix} \in C_1$
- Para la clase Bosque: $x_i \Rightarrow (r, g, b) \rightarrow x_i = \begin{pmatrix} r \\ g \\ b \end{pmatrix} = \begin{pmatrix} 90 \\ 130 \\ 60 \end{pmatrix} \in C_2$
- Para la clase Cielo $x_i \Rightarrow (r, g, b) \rightarrow x_i = \begin{pmatrix} r \\ g \\ b \end{pmatrix} = \begin{pmatrix} 30 \\ 44 \\ 178 \end{pmatrix} \in C_3$

Segmentación

La segmentación subdivide una imagen en sus partes constituyentes u objetos, con el fin de separar las partes de interés del resto de la imagen, por lo tanto, el nivel al que se lleva a cabo esta subdivisión depende del problema a resolver [2]. La segmentación automática es una de las tareas más difíciles del procesamiento de imágenes, esta etapa determina el eventual éxito o fracaso del análisis, de hecho, rara vez llega a alcanzar una solución satisfactoria, se debe buscar un método alternativo de comprobación para la verificación de los resultados. Usualmente hay dos formas de segmentación de la información, mediante el análisis de dos conceptos aplicados a los píxeles: la similaridad o bien mediante la discontinuidad.

Los algoritmos de segmentación de imágenes monocromáticas generalmente se basan en una de las dos propiedades básicas de los valores del nivel de gris: discontinuidad y similaridad [3]. En la discontinuidad el método consiste en dividir una imagen basándose en los cambios bruscos del nivel de gris [4]. Los temas más importantes en la discontinuidad son:

- a) Detección de puntos aislados,
- b) Detección de líneas
- c) Detección de bordes de una imagen.

En la similaridad, se presenta la regularidad en los valores del nivel de gris, los principales métodos están basados en:

- a) Técnicas basadas en la umbralización,
- b) Crecimiento de región,
- c) División y fusión de regiones.

La segmentación de una imagen basada en la discontinuidad o en la similaridad de los valores del nivel de gris de sus píxeles es aplicable tanto a las imágenes estáticas como a las dinámicas (variantes en el tiempo). En la Figura N.º 2 se muestra un ejemplo de segmentación de imagen.

Hemos dicho que la segmentación subdivide una imagen en sus partes constituyentes u objetos, con el fin de separar las partes de interés del resto de la imagen, por lo tanto el nivel al que se lleva a cabo esta subdivisión depende del problema a resolver. En el proceso de detectar las partes en una imagen se identifican bordes de la imagen, o se segmenta está en regiones, líneas o curvas, etc. Otra definición considera a la segmentación como la clasificación de los puntos de la imagen (píxeles), indicando las clases a la que pertenecen los diferentes píxeles. Los atributos básicos de segmentación de una imagen son: la luminancia en imágenes monocromáticas.

Actualmente hay una gran variedad de técnicas que abordan este tema de interés, por ejemplo, las que se pueden agrupar en tres: técnicas orientadas al píxel, a los bordes y a las regiones. Dentro de ellas se pueden destacar los siguientes métodos: línea divisoria de aguas (watershed), una técnica de morfología matemática que a partir de los mínimos en la imagen se aumenta gradualmente el nivel de gris, como si fuera agua que se vierte en un valle, hasta encontrar sus valles vecinos; detección de bordes de las regiones mediante la búsqueda de máximos en el gradiente de la imagen o cruces por cero en la segunda derivada de la imagen; filtros en los que se optimiza una función de costo que considera la exactitud en la posición del borde y la cantidad de bordes detectados; y detección de regiones mediante agrupación de píxeles vecinos con características similares. Otra técnica interesante es por medio de filtros de mediana adaptados al objeto de inspección, la estimación de una imagen libre de fallas a partir de la imagen tomada del objeto mismo. Mediante simple comparación entre la imagen real y la imagen libre de fallas estimada se segmentan las fallas. Si bien es cierto que con este último método se obtienen excelentes

resultados, es necesario invertir mucho tiempo en el diseño del filtro para conseguir una adaptación al objeto. Típicamente, para el análisis de una imagen es necesario dividirla a priori en cientos de sub imágenes. Cada subimagen posee un filtro morfológico distinto, configurado a partir de las características de la porción del objeto presente en ella

- o Extracción de los niveles de gris para cada canal (RGB), especificando el tamaño de cada clase (cuantos píxeles contiene cada una)

Convertir la imagen en escala de grises. imagen tomando el máximo R, G o Nivel B y usarlo como escala de grises nivel de píxel. Esto creará una imagen en escala de grises brillante y puntos brillantes, como las áreas naranja o violeta en él las imágenes de abajo se convierten en blanco.

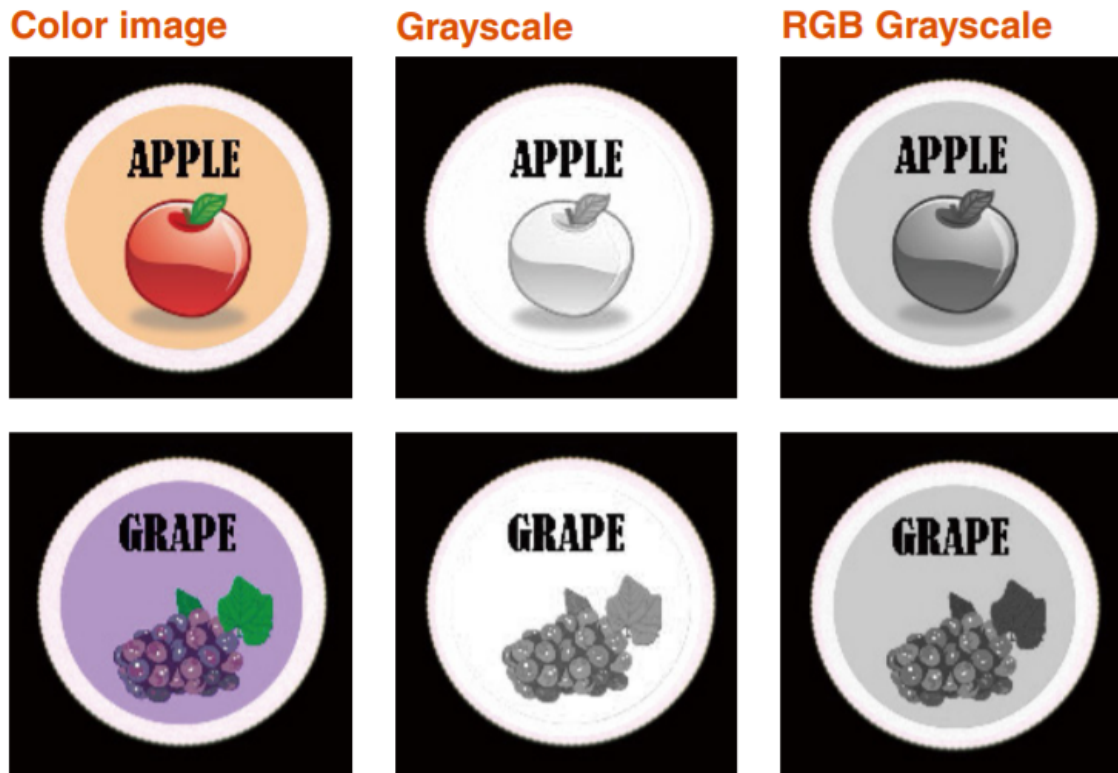


Fig. 4 Imagen ejemplo de la conversión a escala de grises

Convierte la imagen en escala de grises, imagine tomando el promedio de la R, Nivel G y B y usar eso como el nivel de píxeles en escala de grises. Esto creará una imagen en escala de grises cercana a la capturada con una cámara monocromática.

El procesamiento de color a gris convierte una imagen en color en una imagen en escala de grises mediante el uso de un elegido gama de colores como brillo máximo. La el resto de los colores se convierten en una gama de valores de escala de grises basados en los colores seleccionados. Dado que tanto la información de brillo como de color es utilizado, distinción entre colores similares como el oro y la plata se distingue fácilmente como una escala de grises imagen.

Color to gray operation

Image capture

Pre-processing

Image processing

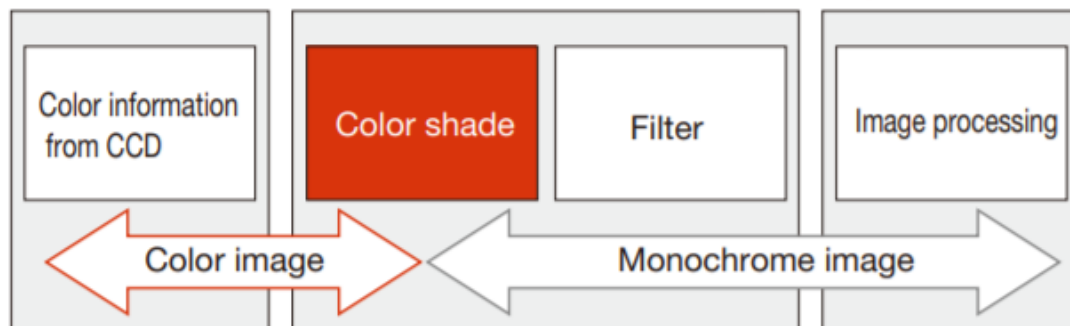


Fig. 5 Diagrama de captura de imagen, preprocesamiento y procesamiento

La etapa de la descripción o extracción de características consiste en extraer características con alguna información cuantitativa de interés o que sean fundamentales para diferenciar una clase de objetos de otra y posteriormente estos datos alimentarán a la fase de reconocimiento, proceso que asignará una etiqueta a un objeto basándose en la información proporcionada por sus descriptores. [1,3] La interpretación implica asignar significado a un conjunto de objetos reconocidos.

Ck	C1					C2					C3				
R	200	210	215	210	198	90	92	87	91	85	30	20	24	28	22
G	160	170	172	165	177	130	138	128	134	123	44	40	42	50	46
B	120	130	133	134	138	60	54	66	60	55	178	180	184	176	181

Fig. 6 Matriz que muestra el CMA

Multi Umbralización

La umbralización es uno de los métodos más importantes de la segmentación de imágenes. Se define el umbral como una función que convierte una imagen con diferentes tonalidades en una imagen en blanco y negro. Si la imagen original es $f(x, y)$, la imagen umbralizada $g(x, y)$ y se fija un umbral U ($0 < U < 255$), la operación de umbralizado se define como:

$$g(x, y) = 255 \text{ si } f(x, y) > \text{Umbral}$$

$$g(x, y) = 0 \text{ si } f(x, y) \leq \text{Umbral}$$

Se selecciona un umbral que permita agrupar los píxeles de una imagen pertenecientes a los diversos objetos de la misma imagen diferenciándolos del fondo [1,2,4]. De esta manera la segmentación basada en el histograma se basa en la elección de uno o varios umbrales que permiten agrupar los puntos de la imagen en regiones de características similares en función de sus niveles de gris. El umbral de una imagen se define como:

$$T = T [x, y, p(x, y), f(x, y)]$$

Donde:

$f(x, y)$ es la intensidad o nivel de gris del punto (x, y) y $p(x, y)$ representa alguna propiedad local medida en un entorno de vecindad de este punto.

La imagen que se obtiene al llevar a cabo un procesamiento de umbral se define como:

$$(x, y) = 1 \text{ si } f(x, y) > T \text{ o si } f(x, y) \leq T$$

De este modo los píxeles marcados con 1 corresponden a objetos, mientras que los píxeles marcados con 0 corresponden al fondo. En la fig. 3 se ha supuesto que la intensidad de los objetos es mayor que la intensidad del fondo. Este proceso es conocido como binarización, por lo que los píxeles por encima del umbral $T = 78$ corresponden al objeto y los que no serán el fondo [1].

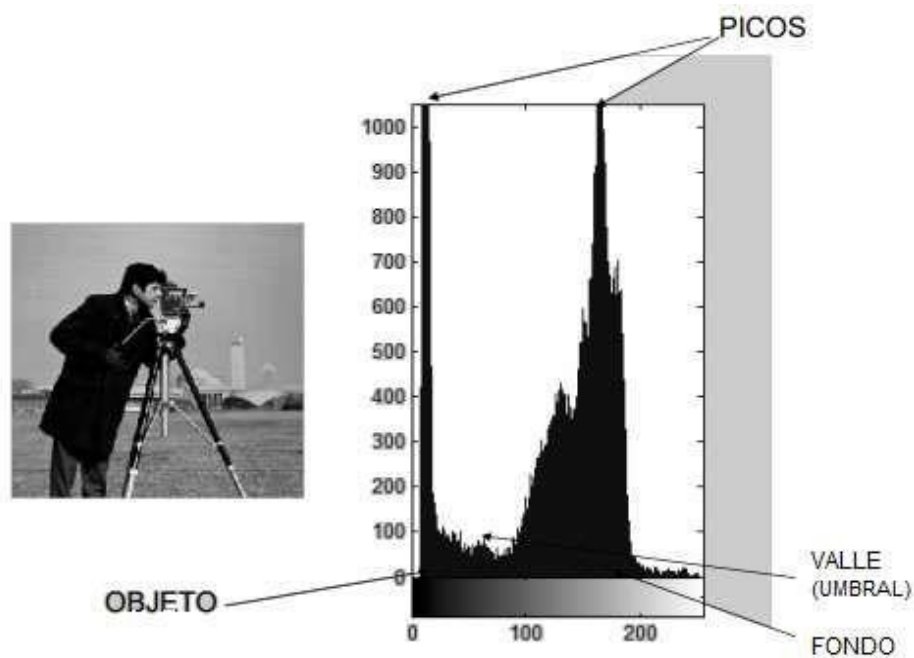


Fig. 7 Imagen “cameraman” en grises y su histograma. Fuente: Gonzalez & Woods

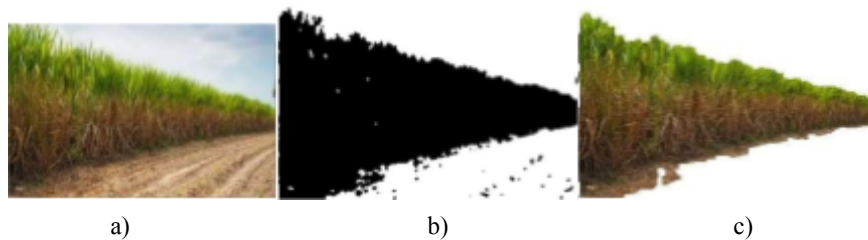


Fig. 8 Imagen con las distintas transformaciones, a) Imagen original, b) Segmentación de región boscosa y c) Extracción de la región boscosa en b)

Hay varios tipos de umbrales, los más utilizados:

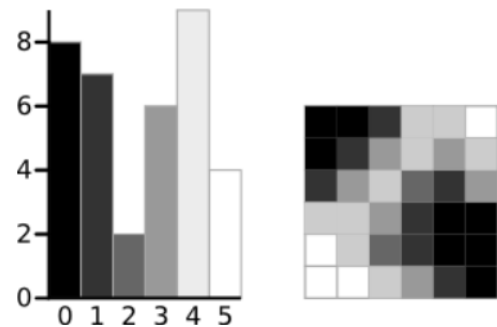
- a) Umbral global, cuando T depende solamente de $f(x, y)$,
- b) Umbral local, si T depende de $f(x, y)$ y de $p(x, y)$,
- c) Umbral dinámico, si T depende de las coordenadas x e y , además de $f(x, y)$ y de $p(x, y)$.
- d) Multi Umbralización

Umbral de Otsu

El método de umbral de Otsu implica iterar a través de todos los posibles valores de umbral y calcular una medida de dispersión para los niveles de píxeles a cada lado del umbral, es decir, los píxeles que caen en primer plano o en segundo plano. El objetivo es encontrar el valor de umbral en el que la suma de los diferenciales de primer plano y de fondo es mínima.

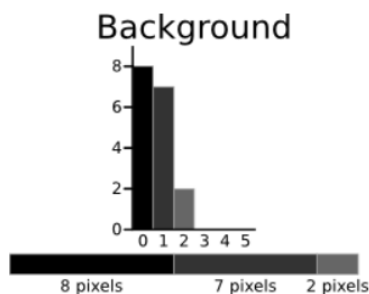
IDENTIFICACIÓN DE TEXTURAS EN UNA IMAGEN DIGITAL

El algoritmo se demostrará utilizando la imagen simple de 6x6 que se muestra a continuación. Junto a él se muestra el histograma de la imagen. Para simplificar la explicación, solo se utilizan 6 niveles de escala de grises.



Una imagen en escala de grises de 6 niveles y su histograma

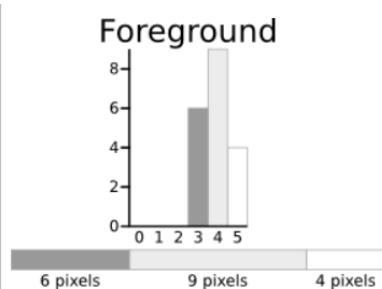
Ahora se muestran los cálculos para encontrar las variaciones de primer plano y de fondo (la medida de la dispersión) para un único umbral. En este caso, el valor umbral es 3.



$$\text{Weight } W_b = \frac{8 + 7 + 2}{36} = 0.4722$$

$$\text{Mean } \mu_b = \frac{(0 \times 8) + (1 \times 7) + (2 \times 2)}{17} = 0.6471$$

$$\begin{aligned} \text{Variance } \sigma_b^2 &= \frac{((0 - 0.6471)^2 \times 8) + ((1 - 0.6471)^2 \times 7) + ((2 - 0.6471)^2 \times 2)}{17} \\ &= \frac{(0.4187 \times 8) + (0.1246 \times 7) + (1.8304 \times 2)}{17} \\ &= 0.4637 \end{aligned}$$



$$\text{Weight } W_f = \frac{6 + 9 + 4}{36} = 0.5278$$

$$\text{Mean } \mu_f = \frac{(3 \times 6) + (4 \times 9) + (5 \times 4)}{19} = 3.8947$$

$$\begin{aligned} \text{Variance } \sigma_f^2 &= \frac{((3 - 3.8947)^2 \times 6) + ((4 - 3.8947)^2 \times 9) + ((5 - 3.8947)^2 \times 4)}{19} \\ &= \frac{(4.8033 \times 6) + (0.0997 \times 9) + (4.8864 \times 4)}{19} \\ &= 0.5152 \end{aligned}$$

El siguiente paso es calcular la 'Varianza dentro de la clase'. Esta es simplemente la suma de las dos variaciones multiplicadas por sus pesos asociados.

$$\begin{aligned} \text{Within Class Variance } \sigma_W^2 &= W_b \sigma_b^2 + W_f \sigma_f^2 = 0.4722 \times 0.4637 + 0.5278 \times 0.5152 \\ &= 0.4909 \end{aligned}$$

Este valor final es la 'suma de las variaciones ponderadas' para el valor de umbral 3. Este mismo cálculo debe realizarse para todos los posibles valores de umbral de 0 a 5. La siguiente tabla

IDENTIFICACIÓN DE TEXTURAS EN UNA IMAGEN DIGITAL

muestra los resultados de estos cálculos. La columna resaltada muestra los valores del umbral calculado anteriormente.

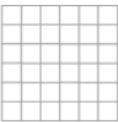
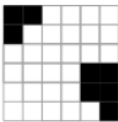
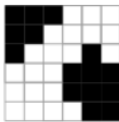
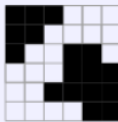
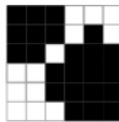

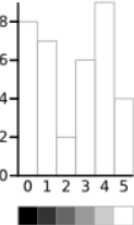
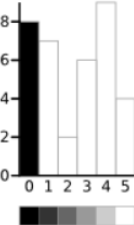
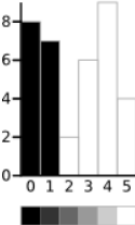
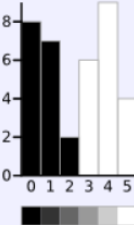
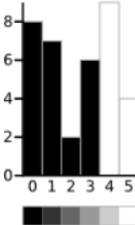
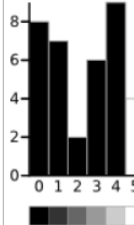
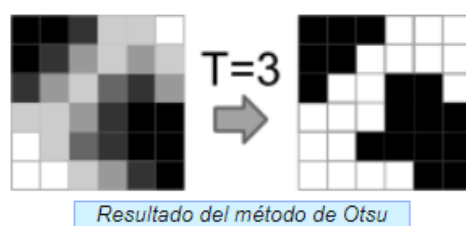
Umbral	T=0	T=1	T=2	T=3	T=4	T=5
						
						
Peso, fondo	$w_b = 0$	$w_b = 0.222$	$w_b = 0.4167$	$w_b = 0.4722$	$w_b = 0.6389$	$w_b = 0.8889$
Media, fondo	$m_b = 0$	$m_b = 0$	$m_b = 0.4667$	$m_b = 0.6471$	$m_b = 1.2609$	$m_b = 2.0313$
Varianza, fondo	$\sigma_b^2 = 0$	$\sigma_b^2 = 0$	$\sigma_b^2 = 0.2489$	$\sigma_b^2 = 0.4637$	$\sigma_b^2 = 1.4102$	$\sigma_b^2 = 2.5303$
Peso, primer plano	$w_f = 1$	$w_f = 0.7778$	$w_f = 0.5833$	$w_f = 0.5278$	$w_f = 0.3611$	$w_f = 0.1111$
Significa, primer plano	$m_f = 2.3611$	$m_f = 3.0357$	$m_f = 3.7143$	$m_f = 3.8947$	$m_f = 4.3077$	$m_f = 5.0000$
Varianza, primer plano	$\sigma_f^2 = 3.1196$	$\sigma_f^2 = 1.9639$	$\sigma_f^2 = 0.7755$	$\sigma_f^2 = 0.5152$	$\sigma_f^2 = 0.2130$	$\sigma_f^2 = 0$
Variación dentro de la clase	$\sigma_W^2 = 3.1196$	$\sigma_W^2 = 1.5268$	$\sigma_W^2 = 0.5561$	$\sigma_W^2 = 0.4909$	$\sigma_W^2 = 0.9779$	$\sigma_W^2 = 2.2491$

Fig 9. Tabla de resultados

Se puede observar que para el umbral igual a 3, además de ser utilizado para el ejemplo, también tiene la menor suma de varianzas ponderadas. Por lo tanto, este es el umbral seleccionado final. Todos los píxeles con un nivel inferior a 3 son de fondo, todos aquellos con un nivel igual o superior a 3 son el primer plano. Como muestran las imágenes de la tabla, este umbral funciona bien.



Desarrollo

Propuesta de solución

Clasificador Paramétrico Bayesiano para una distribución normal (Distancia Euclidiana)

Fase de aprendizaje

Recordemos que:

$$x(k) = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \alpha_i \text{ ó bien, } \in R^+$$

Además de que la forma de la función discriminante es:

$$fd(x) = w^T X * m = [w_1 \ w_2 \ w_3] \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}$$

La construcción de la FD lineal debe ser:

$$fd(x) = w_1 x_1 + w_2 x_2 + w_3$$

Donde: w=vector de coeficiente de discriminación o ponderación (el vector medio), en este caso se desconocen la media.

La problemática en este tipo de clasificador radica en la importancia y la necesidad de seleccionar los rasgos que mejor representen a las clases, por lo que debe elegirse el par de rasgos que son útiles en la discriminación. Se proponen los rasgos que tengan mayor representación en un rango de color entre las clases.

Ck	C1					C2					C3				
R	200	210	215	210	198	90	92	87	91	85	30	20	24	28	22
G	160	170	172	165	177	130	138	128	134	123	44	40	42	50	46
B	120	130	133	134	138	60	54	66	60	55	178	180	184	176	181

Fig 10. Tabla de resultados

Diseño del algoritmo con la 1er. propuesta: Elección de los canales r= rojo y g= verde

1. Se elige una muestra de patrones clasificada de antemano con n clases $\{c_1, c_2, \dots, c_n\}$ y la métrica d_E (distancia Euclidiana).

■ El CMA consta de un total de n muestras para cada clase:

$$C_1 = \{X_1, \dots, X_p\} \text{ con } i=1 \text{ hasta } p$$

$$C_2 = \{X_1, \dots, X_p\} \text{ con } j=1 \text{ hasta } p$$

$$C_3 = \{X_1, \dots, X_p\} \text{ con } k=1 \text{ hasta } p$$

Consideremos el CMA indicado, entonces el cálculo del vector medio:

$$m_1 = \frac{1}{5} \sum_{j=1}^2 x_{ij} = \frac{1}{2} \left[\begin{pmatrix} 200 \\ 160 \end{pmatrix} + \begin{pmatrix} 210 \\ 170 \end{pmatrix} + \begin{pmatrix} 215 \\ 172 \end{pmatrix} + \begin{pmatrix} 210 \\ 165 \end{pmatrix} + \begin{pmatrix} 198 \\ 177 \end{pmatrix} \right] = \begin{pmatrix} 206 \\ 168.8 \end{pmatrix}$$

$$m_1 = \begin{pmatrix} 206 \\ 168.8 \end{pmatrix}, \quad m_2 = \begin{pmatrix} 89.0 \\ 130.6 \end{pmatrix}, \quad m_3 = \begin{pmatrix} 24.8 \\ 44.4 \end{pmatrix}$$

2. Generar funciones discriminantes $d_{ij}(x)$ para cada par de clases C_i, C_j , de forma que:

$$\begin{aligned} d_{ij}(x) &= (z_i - z_j)^t x - \frac{1}{2} [(z_i - z_j)^t (z_i + z_j)] \\ d_{ik}(x) &= (z_i - z_k)^t x - \frac{1}{2} [(z_i - z_k)^t (z_i + z_k)] \\ d_{jk}(x) &= (z_j - z_k)^t x - \frac{1}{2} [(z_j - z_k)^t (z_j + z_k)] \end{aligned}$$

$$\text{donde } x = \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}$$

Calculando la primer: Fd_1 , con $m_1 = \begin{pmatrix} 206 \\ 168.8 \end{pmatrix}$ $m_2 = \begin{pmatrix} 89.0 \\ 130.6 \end{pmatrix}$,

$$d_{12}(x) = \left(\begin{pmatrix} 206 \\ 168.8 \end{pmatrix} - \begin{pmatrix} 89.0 \\ 130.6 \end{pmatrix} \right)^t x - \frac{1}{2} \left[\left(\begin{pmatrix} 206 \\ 168.8 \end{pmatrix} - \begin{pmatrix} 89.0 \\ 130.6 \end{pmatrix} \right)^t \left(\begin{pmatrix} 206 \\ 168.8 \end{pmatrix} + \begin{pmatrix} 89.0 \\ 130.6 \end{pmatrix} \right) \right]$$

$$d_{12}(x) = \begin{pmatrix} 117 & 38.2 \end{pmatrix}^t \cdot X - \frac{\begin{pmatrix} 117 & 38.2 \end{pmatrix}^t * \begin{pmatrix} 295 \\ 299.4 \end{pmatrix}}{2}$$

$$= \begin{pmatrix} 117 & 38.2 \end{pmatrix}^t \cdot X - \frac{1}{2} (34515 + 11526.9) = \begin{pmatrix} 117 & 38.2 \end{pmatrix}^t \cdot X - 22983.525$$

$$\text{dado que } x = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}, \text{ entonces:}$$

$$d_{12}(x) = \begin{pmatrix} 117 & 38.2 \end{pmatrix}^t \cdot \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} - 22983.525$$

$$\mathbf{d_{12}(x) = 117X_1 + 38.2 X_2 - 22983.525}$$

Calculando la segunda Fd2 con $m_1 = \begin{pmatrix} 206 \\ 168.8 \end{pmatrix}$, $m_3 = \begin{pmatrix} 24.8 \\ 44.4 \end{pmatrix}$

$$d_{13}(x) = \left(\begin{pmatrix} 206 \\ 168.8 \end{pmatrix} - \begin{pmatrix} 24.8 \\ 44.4 \end{pmatrix} \right)^t x - \frac{1}{2} \left[\left(\begin{pmatrix} 206 \\ 168.8 \end{pmatrix} - \begin{pmatrix} 24.8 \\ 44.4 \end{pmatrix} \right)^t \left(\begin{pmatrix} 206 \\ 168.8 \end{pmatrix} + \begin{pmatrix} 24.8 \\ 44.4 \end{pmatrix} \right) \right]$$

$$m_1 = \begin{pmatrix} 206 \\ 168.8 \end{pmatrix}, \quad m_2 = \begin{pmatrix} 89.0 \\ 130.6 \end{pmatrix}, \quad m_3 = \begin{pmatrix} 24.8 \\ 44.4 \end{pmatrix}$$

$$d_{13}(x) = \left(\begin{pmatrix} 89.0 \\ 130.6 \end{pmatrix} - \begin{pmatrix} 24.8 \\ 44.4 \end{pmatrix} \right)^t x - \frac{1}{2} \left[\left(\begin{pmatrix} 89.0 \\ 130.6 \end{pmatrix} - \begin{pmatrix} 24.8 \\ 44.4 \end{pmatrix} \right)^t \left(\begin{pmatrix} 89.0 \\ 130.6 \end{pmatrix} + \begin{pmatrix} 24.8 \\ 44.4 \end{pmatrix} \right) \right]$$

$$\text{dado que } x = \begin{pmatrix} 89.0 \\ 130.6 \end{pmatrix}, \text{ entonces:} \quad \begin{pmatrix} 24.8 \\ 44.4 \end{pmatrix}$$

$$d_{13}(x) =$$

Probando la función discriminante Fd_1 encontrada. Si se desea conocer por donde pasa la recta solo debemos despejar alguna de las incógnitas (X_1 o X_2).

También, una vez encontradas las ffd 's restantes, se pueden obtener las mediatrices:

$$fd_{12}(x) = fd_1 - fd_2$$

$$fd_{13}(x) = fd_1 - fd_3$$

$$fd_{23}(x) = fd_2 - fd_3$$

Fase de recuperación

3. En el momento de clasificar (recuperación), el patrón x será clasificado en la clase i si cumple lo siguiente:

$$\forall j, k, j \neq i, k \neq i, x \in C_i, \text{ si } dij(x) \geq 0 \text{ y si } dik(x) \geq 0$$

- Probando el clasificador con el patrón desconocido $x? = (208 \quad 170 \quad 135)$

Tomando la fd_1

$$d_{12}(x) = 117X_1 + 38.2 X_2 - 22983.525$$

Queda:

$$d_{12}(x?) = 117 (208) + 38.2 (170) - 22983.525$$

$$d_{12}(x?) = 24336 + 6494 - 22983.525 = 7846.475$$

Verificar que se cumple:

$$\forall j, k, j \neq i, k \neq i, x \in C_i, \text{ si } dij(x) \geq 0 \text{ y si } dik(x) \geq 0$$

Para conocer a que clase pertenecerá $x?$, $x? \in C?$

Implementación

Segmentación

Antes de poder segmentar la imagen en zonas de interés, se convierte en escala de grises con la función `rgb2gray` de OpenCV. Esto con la finalidad de no tener que trabajar con valores de colores RGB en escala de 0 a 255.

```
image = cv2.imread('CMA-x1.png')

print(image.shape)

gray = rgb2gray(image)
h, w = gray.shape
print(gray.shape)
print('width: ', w)
print('height: ', h)
#print('channel:', c)

gray.shape
```

Fig 10. Segmento de código

Una vez obtenida la imagen en escala de grises, se toma la media de los valores de los píxeles y la usaremos como umbral. Si el valor del píxel es mayor que nuestro umbral, podemos decir que pertenece a una zona de interés. Si el valor del píxel es menor que el umbral, se tratará como fondo.

```
gray_r = gray.reshape(gray.shape[0]*gray.shape[1])
for i in range(gray_r.shape[0]):
    if gray_r[i] > gray_r.mean():
        gray_r[i] = 1
    else:
        gray_r[i] = 0
gray = gray_r.reshape(gray.shape[0], gray.shape[1])
```

Fig 11. Segmento de código

Se le aplicó 3 filtros máximos y 3 mínimos a la imagen para obtener la zona boscosa en blanco y negro, se hizo un 'or' con la imagen original para obtenerla

con sus colores originales, se le aplicó un 'not' a la imagen de la zona boscosa en B/N para obtener las otras 2 clases y aplicar un 'or' para obtenerla en color original.

```
maxBoxFilter(3, filename)
minBoxFilter(5, filename)

filename2 = 'boscoso.jpg'
filename3 = 'cieloSuelo.jpg'

image2 = cv2.imread("BN.jpg")

OR = cv2.bitwise_or(image, image2)
cv2.imshow('OR', OR)
cv2.imwrite(filename2, OR)

mask_inv = cv2.bitwise_not(image2)
cv2.imshow('NOT', mask_inv)

OR2 = cv2.bitwise_or(image, mask_inv)
cv2.imshow('OR2', OR2)
cv2.imwrite(filename3, OR2)
```

Fig 12. Segmento de código

Algoritmo

Se insertan los pixeles del CMA a su respectiva clase.

```
f = open('cieloRGB.txt','r')
for i in range(4515):
    mensaje = f.readline()
    clase1.append([int(i) for i in mensaje[:-1].split(' ')])
f.close()

f = open('boscosoRGB.txt','r')
for i in range(35700):
    mensaje = f.readline()
    clase2.append([int(i) for i in mensaje[:-1].split(' ')])
f.close()

f = open('sueloRGB.txt','r')
for i in range(9350):
    mensaje = f.readline()
    clase3.append([int(i) for i in mensaje[:-1].split(' ')])
f.close()
```

Fig 13. Segmento de código

```
media1 = calcularMedia(clase1)
media2 = calcularMedia(clase2)
media3 = calcularMedia(clase3)
```

Fig 14. Segmento de código

Mediante la media se sacan los centroides de cada clase.

```
def calcularMedia(clase):
    media = [0,0,0]
    for i in range(len(clase)):
        for j in range(3):
            media[j] += clase[i][j]

    for i in range(3):
        media[i] /= len(clase)

    return media
```

Fig 15. Segmento de código

```
d12 = funcionDiscriminante(media1,media2)
d13 = funcionDiscriminante(media1,media3)
d23 = funcionDiscriminante(media2,media3)
```

Fig 16. Segmento de código

Se utilizaron los canales rojo y verde como las características, se calcula la función discriminante.

```
def funcionDiscriminante(claseA,claseB):
    A = [0,0,0]
    for i in range(3):
        if i != 2:
            A[i] = claseA[i] - claseB[i]

    B = [0,0,0]
    for i in range(3):
        if i != 2:
            B[i] = claseA[i] + claseB[i]

    C = [0,0,0]
    for i in range(3):
        if i != 2:
            C[i] = A[i] * B[i]

    D = 0
    for i in range(3):
        if i != 2:
            D += C[i]
    D /= -2

    F = [A[0],A[1],D]

    return F
```

Fig 17. Segmento de código

```
d1 = probarFD(patronDesconocido,d12)
d2 = probarFD(patronDesconocido,d13)
d3 = probarFD(patronDesconocido,d23)
```

Fig 18. Segmento de código

Se prueba la función discriminante con el patrón desconocido

```
def probarFD(patron, d):
    res = 0
    for i in range(3):
        if i != 2:
            res += d[i] * patron[i]
    res += d[-1]

    return res
```

Fig 19. Segmento de código

La regla de aprendizaje utilizada fue la siguiente

```
clase = ""
if d1 >= 0 and d2 >= 0:
    clase = "Cielo"
elif d1 >= 0 and d3 >= 0:
    clase = "Bosque"
elif d2 >= 0 and d3 >= 0:
    clase = "Suelo"
elif d3 >= 0:
    clase = "Bosque"
elif d1 >= 0:
    clase = "Suelo"
```

Fig 20 . Segmento de código

Se usó la regla aprendizaje propuesta en clase y además se tuvo que poner 2 condiciones adicionales, nos dimos cuenta que al hacer clic en el suelo los valores que a veces nos arrojaba eran de d1 y d2 negativos y d3 positivo, y en caso del bosque, a veces nos daba d2 y d3 negativos y d1 positivo, así que se agregaron esas condiciones

Pruebas y resultados

A continuación se presentan las pruebas y los resultados de la implementación del Clasificador Paramétrico Bayesiano para una distribución normal (Distancia Euclidiana):

Primeramente, se convierte la imagen original a escala de grises y se binariza

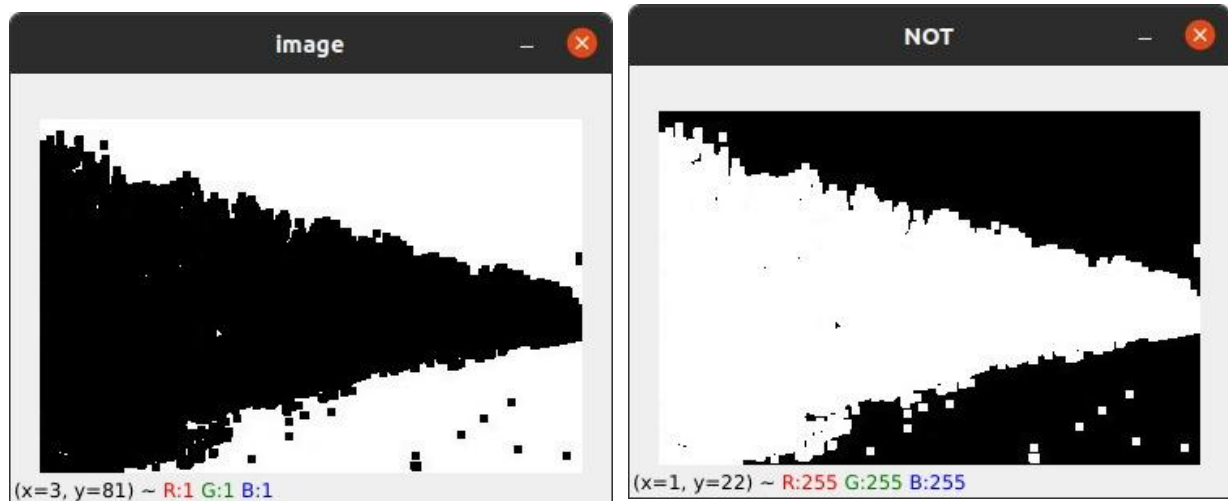


Fig 21 . Imagen binarizada e invertida

Después, se aplica una dilatación con un filtro de 3x3 a la imagen anterior.

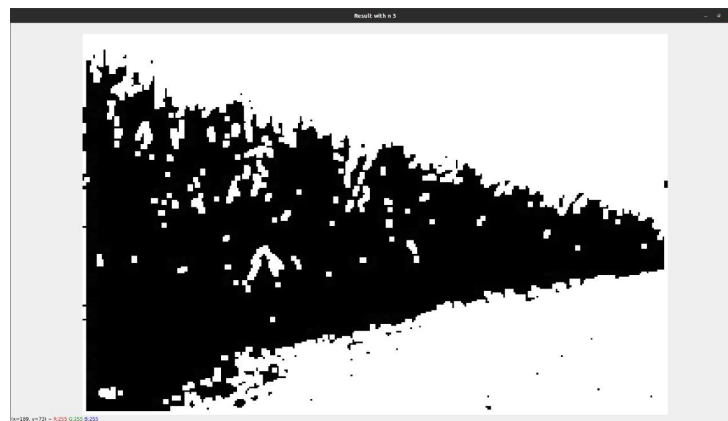


Fig 22. Imagen binarizada e Imagen dilatada con kernel de 3x3

Después, se aplica una erosión con un filtro de 5x5 a la imagen anterior

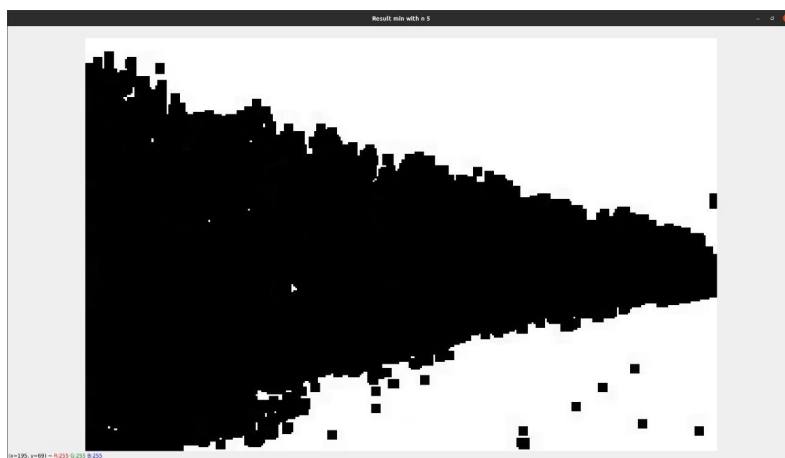


Fig 23. Imagen erosionada con kernel de 5x5

Después, se aplicó la operación lógica OR entre la imagen original y la imagen binarizada



Fig 24. Imagen resultante de la operación OR

IDENTIFICACIÓN DE TEXTURAS EN UNA IMAGEN DIGITAL

Después, se aplicó la operación lógica OR entre la imagen original y la imagen binarizada invertida



Fig 25. Imagen resultante de la operación OR

Acto seguido, se procede a la extracción de CMA, y dicha información se almacena en archivos .txt

boscosoRGB.txt	cieloRGB.txt	sueloRGB.txt
1 164 175 107	1 219 231 221	1 170 187 117
2 129 143 55	2 217 228 220	2 98 114 52
3 139 154 69	3 215 216 220	3 123 136 90
4 136 149 79	4 221 222 224	4 178 189 155
5 146 155 110	5 212 213 217	5 204 213 192
6 159 176 82	6 205 206 211	6 164 173 156
7 118 139 18	7 220 220 228	7 171 178 160
8 127 147 34	8 227 227 237	8 231 239 215
9 136 153 61	9 226 227 232	9 190 197 164
10 127 138 80	10 220 221 226	10 89 99 47
11 162 182 71	11 225 224 238	11 140 153 74
12 118 142 6	12 223 222 238	12 151 166 85
13 116 138 12	13 214 216 229	13 158 169 113
14 126 145 40	14 222 224 236	14 206 217 174
15 146 159 90	15 228 232 241	15 217 229 191
16 162 174 110	16 226 231 237	16 195 209 174
17 145 161 72	17 223 228 232	17 203 218 189
18 126 144 45	18 222 227 230	18 215 230 207
19 130 147 55	19 217 225 227	19 223 234 220
20 150 163 93	20 214 223 222	20 219 230 216
21 144 164 49	21 214 223 220	21 195 202 194
22 121 144 14	22 216 225 222	22 144 152 139
23 99 120 0	23 215 225 224	23 142 152 127
24 115 133 31	24 213 223 224	24 195 206 172

Fig 26. Conjunto de Muestras de Aprendizaje

IDENTIFICACIÓN DE TEXTURAS EN UNA IMAGEN DIGITAL

A continuación se muestran algunas pruebas de escritorio y sus respectivos resultados

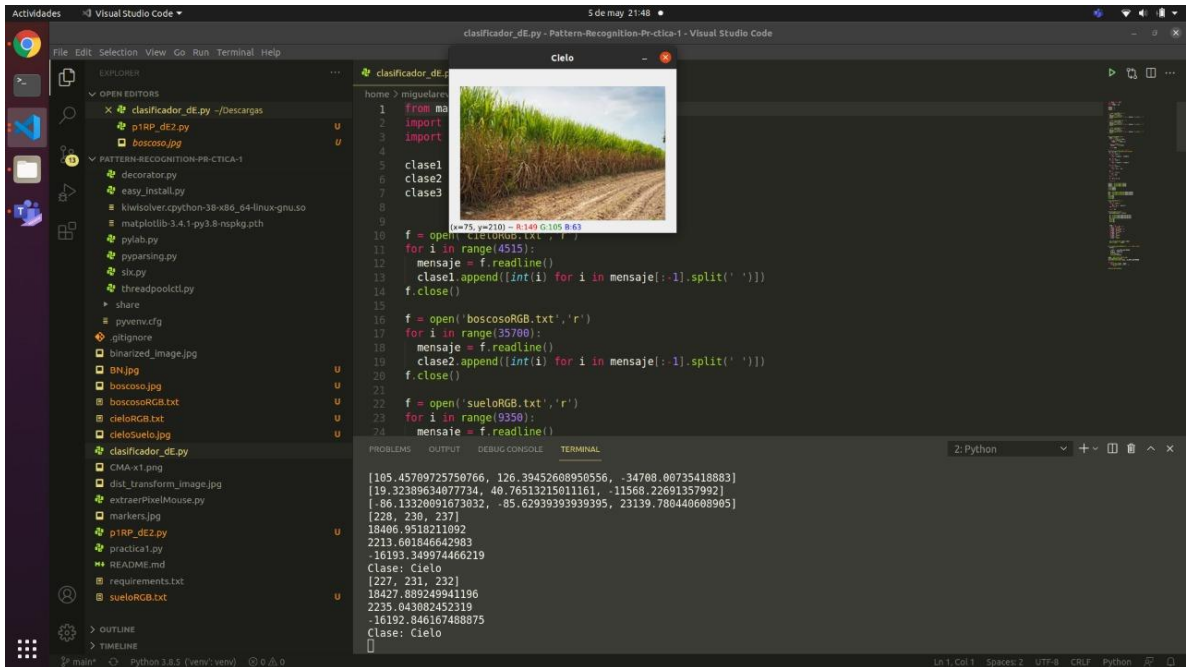


Fig 27. Primer prueba de escritorio

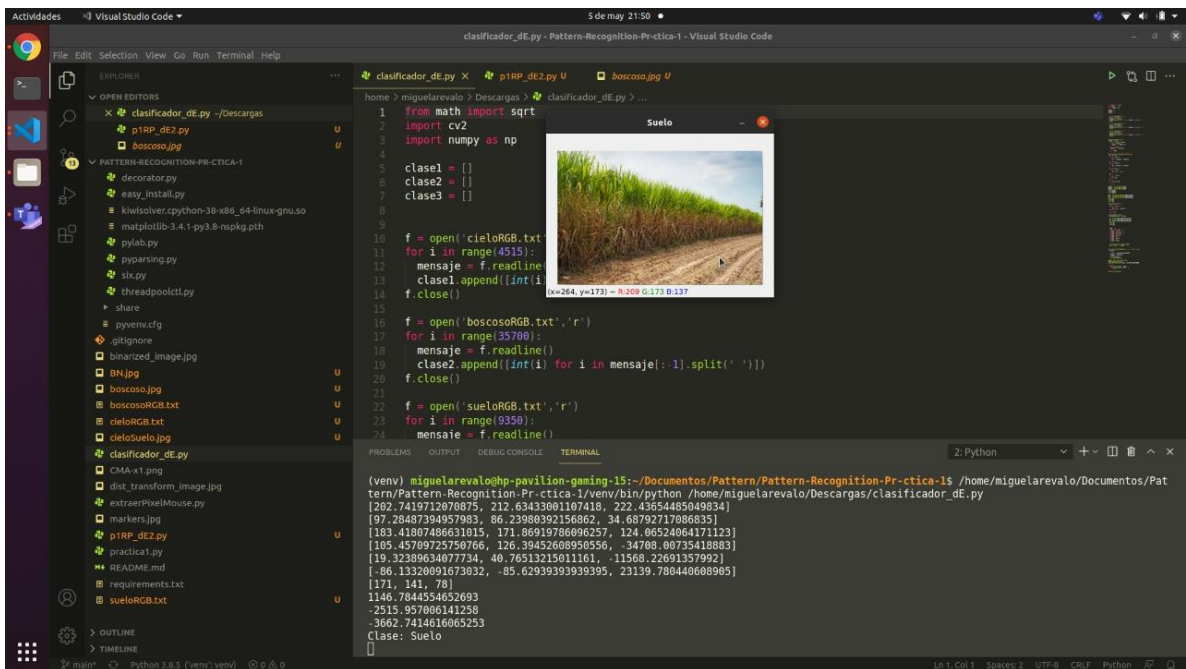


Fig 28. Segunda prueba de escritorio

IDENTIFICACIÓN DE TEXTURAS EN UNA IMAGEN DIGITAL

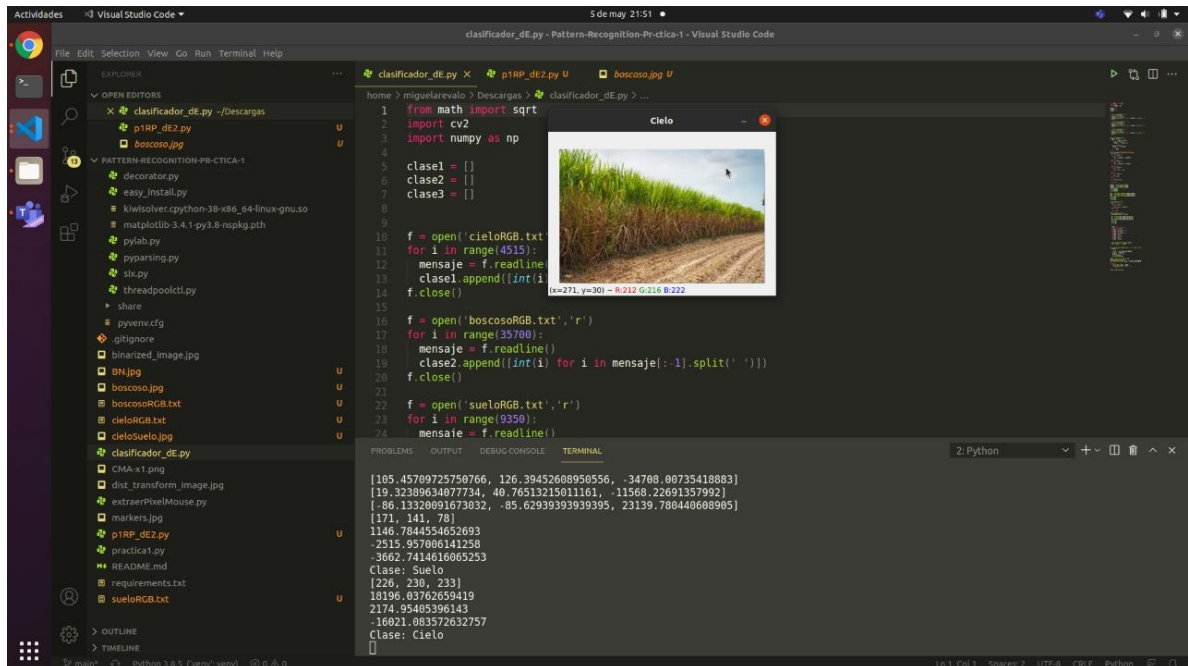


Fig 29. Tercer prueba de escritorio

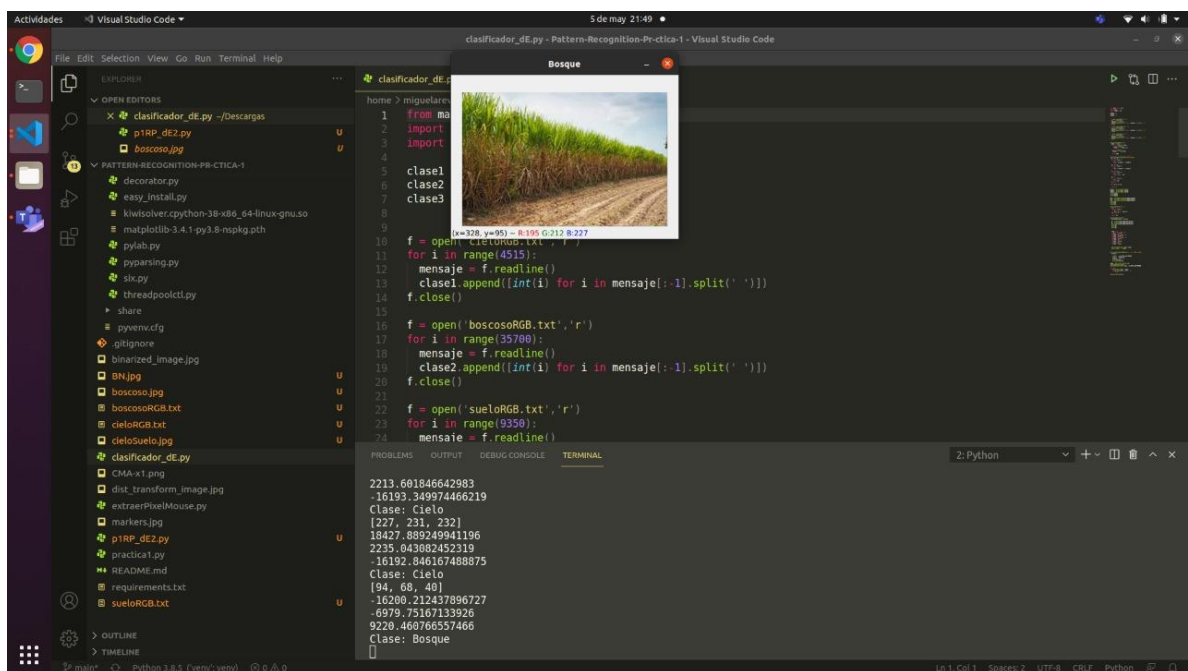


Fig 30. Cuarta prueba de escritorio

IDENTIFICACIÓN DE TEXTURAS EN UNA IMAGEN DIGITAL

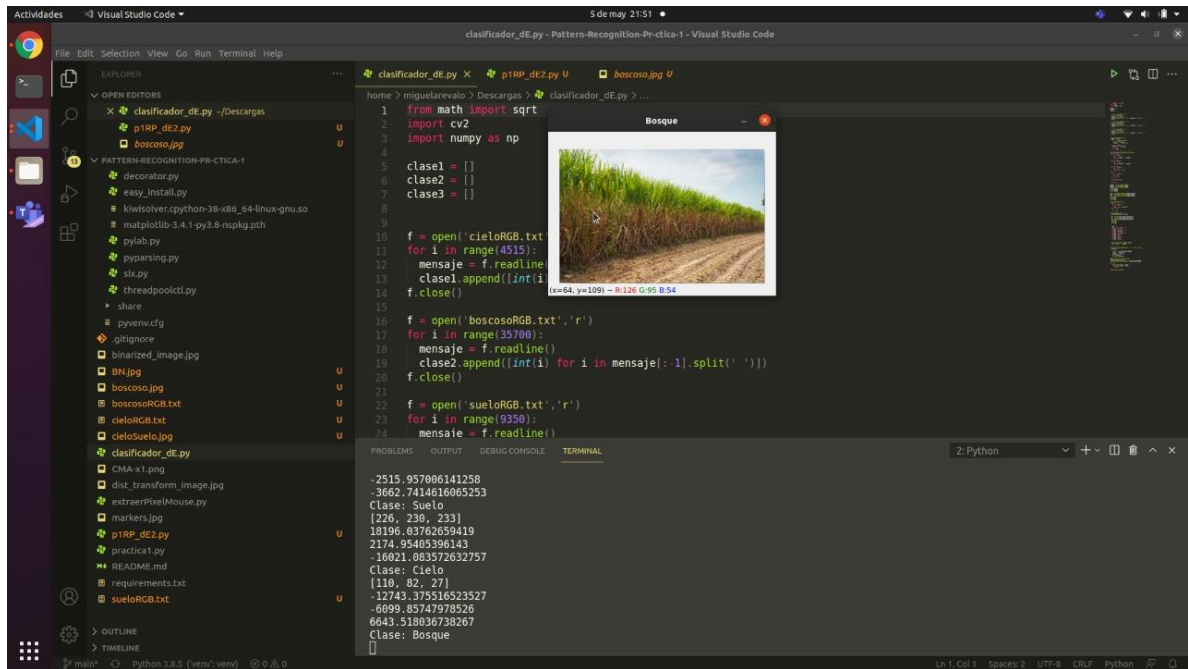


Fig 31. Quinta prueba de escritorio

Conclusiones

Objetivo General:

- Diseñar un algoritmo basado en el enfoque estadístico probabilístico o basado en una norma con aprendizaje supervisado que permita identificar una zona o área de interés en una imagen digital.

Objetivos Particulares:

- Se deben identificar los parámetros requeridos que describan a las clases de las áreas boscosas, tierra y cielo presentes en una imagen digital.
- Analizar el tipo de aprendizaje supervisado que puede resolver el problema de la identificación para los casos de:
 - Clasificación basada en una norma, o bien,
 - Para los casos de aproximación paramétrica o no paramétrica,
 - Para el diseño de FD para clases con Distribución Normal o Gaussiana, o mediante,
 - La ventana de Parzen.

Retomando los objetivos previos a la realización de la práctica, podemos concluir lo siguiente:

- Primeramente, nosotros implementamos un Clasificador Paramétrico Bayesiano para una distribución normal (Distancia Euclidiana).
- Se logró procesar la imagen deseada para su:
 - Separación de canales
 - Conversión a escala de grises
 - Binarización aplicando multi-umbralización
 - Segmentación
- Se extrajo el Conjunto de Muestras de Aprendizaje de la imagen previamente procesada.
- Se aplicó la metodología vista en clase para la clasificación de vectores desconocidos usando el distancia Euclídea.
- Se muestra el resultado de la clasificación en consola.

En conclusión, se lograron satisfactoriamente los objetivos previamente establecidos.

Fuentes consultadas

- Bielza, C., Li, G., Larranaga, P.: Multi-Dimensional ~ Classification with Bayesian Networks. International Journal of Approximate Reasoning, (2011)
- Borchani, H., Bielza, C., Toro, C., Larranaga, P.: ~ Predicting human immunodeficiency virus inhibitors using multi-dimensional Bayesian network classifiers. Artificial Intelligence in Medicine 57, 219–229 (2013)
- Cheng, J., Greiner, R.: Comparing Bayesian Network Classifiers. Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, 101–108 (1999)
- Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian Network Classifiers. Machine Learning 29, 131–163 (1997)
- Gonzálo Pajares Martinsanz & Jesús M. de la Cruz García. Visión por computador. Ed. Alfaomega Ra-MA. Edición. Primera Edición. Abril 2008
- Silverman, B.W. (1986) Density Estimation for Statistics and Data Analysis. Chapman and Hall, London.
- Wand, M. P. and Jones, M. C. (1995). Kernel Smoothing. Chapman and Hall, London.
- Bowman, A.W. and Azzalini, A. (1997). Applied Smoothing Techniques for Data Analysis: the Kernel Approach with S-Plus Illustrations. Oxford University Press, Oxford



Instituto Politécnico Nacional

Escuela Superior de Cómputo



Unidad de Aprendizaje: Pattern Recognition

Práctica 2: Reporte

Unidad Temática: Implementación de un PMC

Grupo: 3CM11

Profesora: Cruz Meza María Elena

Fecha de entrega: 19 de junio de 2021

Integrantes:

Arévalo Andrade Miguel Ángel

Medero Luján Alejandro

Castro Cruces Jorge Eduardo

Franco Ake Alan

Índice

- Portada (
 - identidad politécnica,
 - nombre del curso,
 - grupo,
 - nombre de la tarea,
 - nombre de los integrantes,
 - fecha de entrega)
- Índice o contenido
- Introducción breve
- Objetivo del proyecto (recordemos que en este caso el objetivo es segmentación completa del objeto)
- Marco teórico
 - Relacionado con las tarea asignadas en las prácticas anteriores: A la información contenida en el reporte anterior agregar ahora el tema de interés, los clasificadores lineales, el perceptrón.
- Desarrollo-propuesta
 - Este segmento se refiere a la descripción de la imagen seleccionada, la estrategia de la extracción de rasgos, es decir, los pasos de los procesos que el equipo consideró es la adecuada para el objeto a extraer o segmentar de forma completa en la imagen en conjunto con algunas pruebas realizadas para llegar a esta solución (describir si se requirió de conversión de niveles de gris, umbralización, etc). En esta sección deberá incluir los segmentos de código relacionados al algoritmo que reporta, ya sea el del perceptrón y/o el de la memoria asociativa.
- Resultados
 - Agregar la sección que describa la estrategia de solución para la extracción del objeto de interés con el clasificador que en el reporta (el perceptrón y/o memoria asociativa), anexando el logro de la clasificación del objeto con el algoritmo.
- Conclusiones
 - Esta sección deberá contener las conclusiones generales acerca del objetivo planteado en el problema, es decir, si se logro clasificar el objeto de interés elegido con la técnica que aquí reporta o bien con ambas técnicas.

Introducción

En este capítulo se estudiará una de las clases de redes de neuronas, conocida como Perceptrón multicapa o red multicapa con conexiones hacia adelante. El Perceptrón multicapa es una generalización del Perceptron simple y surgió como consecuencia de las limitaciones de dicha arquitectura en lo referente al problema de la separabilidad no lineal. Minsky y Papert [4] mostraron en 1969 que la combinación de varios Perceptrones simples -inclusión de neuronas ocultas- podría resultar una solución adecuada para tratar ciertos problemas no lineales. Sin embargo, los autores no presentaron una solución al problema de cómo adaptar los pesos de la capa de entrada a la capa oculta, pues la regla de aprendizaje del Perceptrón simple no puede aplicarse en este escenario. No obstante, la idea de combinar varios Perceptrones sirvió de base para estudios posteriores realizados por Rummelhart, Hinton y Wilians en 1986 [5]. Estos autores presentaron una manera de retropropagación de los errores medidos en la salida de la red hacia las neuronas ocultas, dando lugar a la llamada regla delta generalizada. Diferentes autores han demostrado independientemente que el Perceptrón multicapa es un aproximador universal, en el sentido de que cualquier función continua en un espacio R_n puede aproximarse con un Perceptrón multicapa, con al menos una capa oculta de neuronas. Tal como se explicó en el capítulo anterior, este resultado sitúa al Perceptrón multicapa como un modelo matemático útil a la hora de aproximar o interpolar relaciones no lineales entre datos de entrada y salida. Dentro del marco de las redes de neuronas, el Perceptrón multicapa es en la actualidad una de las arquitecturas más utilizadas en la resolución de problemas. Esto es debido, fundamentalmente, a su capacidad como aproximador universal, así como a su fácil uso y aplicabilidad.

Por otra parte, esto no implica que sea una de las redes más potentes y con mejores resultados en sus diferentes áreas de aplicación. De hecho, el Perceptrón multicapa posee una serie de limitaciones, como el largo proceso de aprendizaje para problemas complejos dependientes de un gran número de variables; la dificultad para realizar un análisis teórico de la red debido a la presencia de componentes no lineales y a la alta conectividad.

Por otra parte, es necesario señalar que el proceso de aprendizaje de la red busca en un espacio amplio de funciones, una posible función que relacione las variables de entrada y salida al problema, lo cual puede complicar su aprendizaje y reducir su efectividad en determinadas aplicaciones [6].

Objetivo general

Mostrar la habilidad para identificar los parámetros y el manejo e implementación de los algoritmos de reconocimiento de patrones, basados en el enfoque neuronal con la finalidad de que analices la diferencia respecto al modelo que fue asignado al equipo (Clasificadores Basados en Métricas y los Clasificadores Bayesianos), con la finalidad de identificar el tipo de aprendizaje supervisado y el proceso de los datos en el modelo basado en redes neuronales, mediante el trabajo colaborativo usando herramientas de gestión, programación y comunicación en un ambiente virtual (b-learning y m-Learning).

Objetivos particulares

1. Tomar como referencia el problema de la práctica anterior e implementar el modelo del PMC que se adjunta y evaluar si permite discriminar cada región de la imagen. Recordemos que en el caso planteado son tres clases donde los rasgos deben poder describirlos para poder diseñar el clasificador que permita discriminar entre estas. El proceso que genera el Conjunto de Muestras de Entrenamiento o Aprendizaje (CME o CMA), por lo que ahora debe adaptarse este módulo para la tarea 2.

- Bosque
- Tierra
- Cielo

2. Analizar el problema de extracción de rasgos.

Consideremos el hecho de que la entrada de datos al Sistema de Identificación requiere leer una imagen de entrada, extraer el conjunto de muestras de aprendizaje para describir cada región de la imagen:

- Bosque
- Tierra
- Cielo

Recordemos que en la práctica anterior se requirió un proceso que permite a un usuario seleccionar una imagen, mostrarla en pantalla (aquí dependerá de la creatividad si la coloca en el panel o navegador, ventana, pestaña, etc.), donde el Sistema de identificación al colocar el mouse en alguna posición de la misma, indica a qué región de las tres clases pertenece ese pixel. En concreto, ajusten los objetivos de la tarea resuelta y considerando esta tarea, reusen la práctica anterior (proceso de abrir una image, visualizarla y manipular el mouse para que la posición que el apunte, extraiga ese dato y sea enviado al clasificador) ajustando el CMA como entrada al modelo del Perceptrón.

3. Implementar el clasificador PMC o MLP

Considerar el modelo del perceptrón diseñado para resolver el problema (documento adjunto). Documentar este proceso en el reporte entregado en la práctica anterior, mostrando la respuesta del modelo y ajustando las conclusiones y especifiquen la parte del Código correspondiente al PMC.

Marco teórico

Clasificadores lineales: descripción general

Una clase popular de procedimientos para resolver tareas de clasificación se basa en modelos lineales. Lo que esto significa es que su objetivo es dividir el espacio de características en una colección de regiones etiquetadas de acuerdo con los valores que puede tomar el objetivo, donde los límites de decisión entre esas regiones son lineales: son líneas en 2D, planos en 3D e hiperplanos con más características. Este artículo revisa los modelos lineales populares para la clasificación, proporcionando las descripciones de los métodos discutidos, así como las implementaciones de Python. Cubriremos los siguientes enfoques: análisis discriminante lineal, análisis discriminante cuadrático, análisis discriminante regularizado, regresión logística.

Análisis discriminante lineal

El primer método que se discutirá es el Análisis Discriminante Lineal (LDA). Asume que la densidad conjunta de todas las características, condicionada a la clase del objetivo, es un gaussiano multivariado. Esto significa que la densidad P de las características X , dado que el objetivo y está en la clase k , se supone que está dada por:

$$P(X|y = k) = \frac{1}{(2\pi)^{d/2} |\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(X - \mu_k)^t \Sigma_k^{-1} (X - \mu_k)\right)$$

donde d es el número de características, μ es un vector medio y Σ_k la matriz de covarianza de la densidad gaussiana para la clase k .

El límite de decisión entre dos clases, digamos k y l , es el hiperplano en el que la probabilidad de pertenecer a cualquier clase es la misma. Esto implica que, en este hiperplano, la diferencia entre las dos densidades (y, por lo tanto, también la razón logarítmica de probabilidades entre ellas) debe ser cero.

Una suposición importante en LDA es que los gaussianos para diferentes clases comparten la misma matriz de covarianza: el subíndice k de Σ_k en la fórmula anterior se puede eliminar. Esta suposición es útil para el cálculo de la razón logarítmica de probabilidades: hace que los factores de normalización y algunas partes cuadráticas del exponente se cancelen. Esto produce un límite de decisión entre k y l que es lineal en X :

$$\log\left(\frac{P(y = k|X)}{P(y = l|X)}\right) = \log\left(\frac{P(X|y = k)P(y = k)}{P(X|y = l)P(y = l)}\right) = 0 \Leftrightarrow$$

$$(\mu_k - \mu_l)^t \Sigma^{-1} X = \frac{1}{2}(\mu_k^t \Sigma^{-1} \mu_k - \mu_l^t \Sigma^{-1} \mu_l) - \log \frac{P(y = k)}{P(y = l)}$$

Para calcular la densidad de las características, $P(X | y = k)$, basta con estimar los parámetros gaussianos: las medias μ_k como medias muestrales y la matriz de covarianza Σ como matriz de covarianza muestral empírica. Habiendo calculado esto, la probabilidad de que el objetivo pertenezca a la clase k se puede obtener de la regla de Bayes:

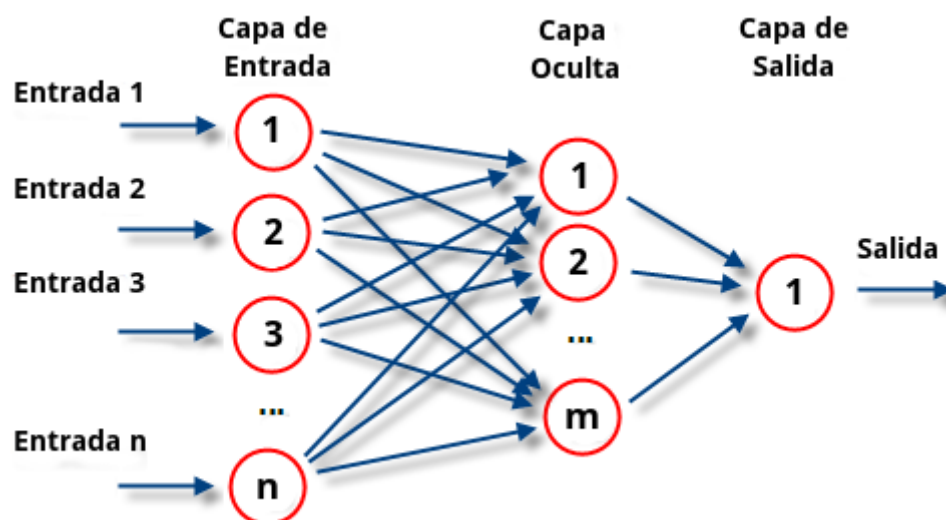
$$P(y = k|X) = \frac{P(X|y = k)P(y = k)}{P(X)}$$

donde $P(y = k)$ es la probabilidad previa de pertenecer a la clase k y puede estimarse mediante la proporción de observaciones de la clase k en la muestra.

Tenga en cuenta que LDA no tiene hiper parámetros para ajustar. Solo se necesitan unas pocas líneas de código para aplicarlo a los datos de spam [7].

Perceptrón multicapa

El perceptrón multicapa es una red neuronal artificial (RNA) formada por múltiples capas, de tal manera que tiene capacidad para resolver problemas que no son linealmente separables, lo cual es la principal limitación del perceptrón (también llamado perceptrón simple). El perceptrón multicapa puede estar totalmente o localmente conectado. En el primer caso cada salida de una neurona de la capa " i " es entrada de todas las neuronas de la capa " $i+1$ ", mientras que en el segundo cada neurona de la capa " i " es entrada de una serie de neuronas (región) de la capa " $i+1$ ".



Tipos

Las capas pueden clasificarse en tres tipos:

Capa de entrada: Constituida por aquellas neuronas que introducen los patrones de entrada en la red. En estas neuronas no se produce procesamiento.

Capas ocultas: Formada por aquellas neuronas cuyas entradas provienen de capas anteriores y cuyas salidas pasan a neuronas de capas posteriores.

Capa de salida: Neuronas cuyos valores de salida se corresponden con las salidas de toda la red.

La propagación hacia atrás (también conocido como retropropagación del error o regla delta generalizada), es un algoritmo utilizado en el entrenamiento de estas redes, por ello, el perceptrón multicapa también es conocido como red de retropropagación (no confundir con la red de contrapropagación) [8].

La clasificación de etiquetas múltiples implica predecir cero o más etiquetas de clase.

A diferencia de las tareas de clasificación normales en las que las etiquetas de clase son mutuamente excluyentes, la clasificación de múltiples etiquetas requiere algoritmos de aprendizaje automático especializados que admitan la predicción de múltiples clases o "etiquetas" mutuamente no excluyentes.

Las redes neuronales de aprendizaje profundo son un ejemplo de un algoritmo que admite de forma nativa problemas de clasificación de múltiples etiquetas. Los modelos de redes neuronales para tareas de clasificación de etiquetas múltiples se pueden definir y evaluar fácilmente utilizando la biblioteca de aprendizaje profundo de Keras.

En este marco teórico, descubrirá cómo desarrollar modelos de aprendizaje profundo para la clasificación de múltiples etiquetas.

Después de leer esta información, sabrá:

- La clasificación de etiquetas múltiples es una tarea de modelado predictivo que implica predecir cero o más etiquetas de clase mutuamente no excluyentes.
- Los modelos de redes neuronales se pueden configurar para tareas de clasificación de etiquetas múltiples.
- Cómo evaluar una red neuronal para la clasificación de múltiples etiquetas y hacer una predicción de nuevos datos.

Esta información se divide en tres partes; ellos son:

- Clasificación de etiquetas múltiples
- Redes neuronales para múltiples etiquetas
- Red neuronal para clasificación de etiquetas múltiples

Clasificación de etiquetas múltiples

La clasificación es un problema de modelado predictivo que implica generar una etiqueta de clase dada una entrada

Es diferente de las tareas de regresión que implica predecir un valor numérico.

Normalmente, una tarea de clasificación implica predecir una sola etiqueta. Alternativamente, podría implicar predecir la probabilidad en dos o más etiquetas de clase. En estos casos, las clases son mutuamente excluyentes, lo que significa que la tarea de clasificación asume que la entrada pertenece a una sola clase.

Algunas tareas de clasificación requieren predecir más de una etiqueta de clase. Esto significa que las etiquetas de clase o la pertenencia a clases no se excluyen mutuamente. Estas tareas se conocen como clasificación de etiquetas múltiples o clasificación de etiquetas múltiples para abreviar.

En la clasificación de etiquetas múltiples, se requieren cero o más etiquetas como salida para cada muestra de entrada, y las salidas se requieren simultáneamente. El supuesto es que las etiquetas de salida son una función de las entradas.

Redes neuronales para múltiples etiquetas

Algunos algoritmos de aprendizaje automático admiten la clasificación de etiquetas múltiples de forma nativa.

Los modelos de redes neuronales se pueden configurar para admitir la clasificación de múltiples etiquetas y pueden funcionar bien, según las especificaciones de la tarea de clasificación.

La clasificación de etiquetas múltiples puede ser soportada directamente por redes neuronales simplemente especificando el número de etiquetas de destino que hay en el problema como el número de nodos en la capa de salida. Por ejemplo, una tarea que tiene tres etiquetas de salida (clases) requerirá una capa de salida de red neuronal con tres nodos en la capa de salida.

Cada nodo de la capa de salida debe utilizar la activación sigmoidea. Esto predecirá una probabilidad de pertenencia a una clase para la etiqueta, un valor entre 0 y 1. Finalmente, el modelo debe ajustarse a la función binaria de pérdida de entropía cruzada.

En resumen, para configurar un modelo de red neuronal para la clasificación de etiquetas múltiples, los detalles son:

- El número de nodos en la capa de salida coincide con el número de etiquetas.
- Activación sigmoidea para cada nodo en la capa de salida.
- Función de pérdida de entropía cruzada binaria [9].

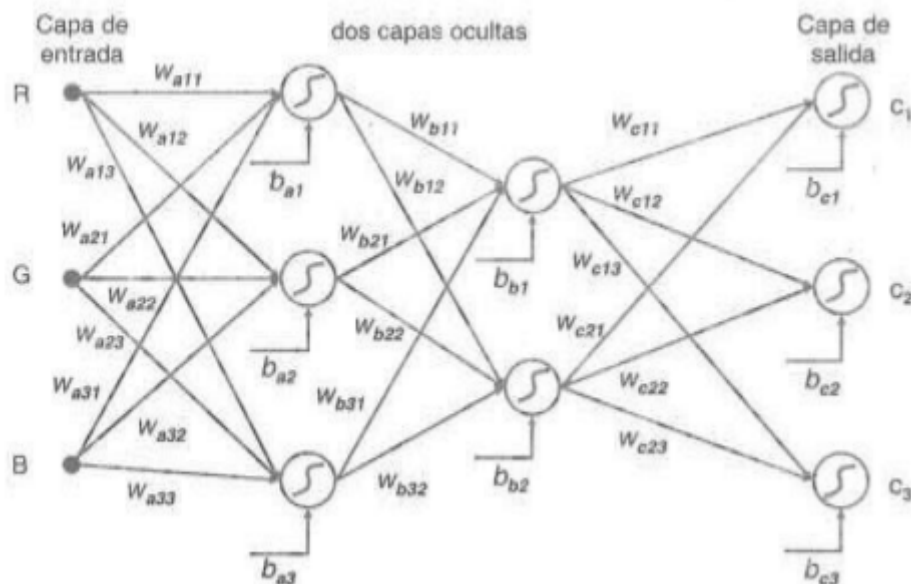
Desarrollo - propuesta

Un sistema basado en visión por computadora trata de clasificar tres tipos de texturas naturales basadas en las tres componentes de color (R,G,B), que se corresponden con regiones de cielo, bosque y suelo. Se han extraído los datos de la imagen dados a continuación. Diseñar un modelo de red neuronal multicapa con entrenamiento por retropropagación para el aprendizaje. A continuación determinar a qué clase pertenecen los pixeles con los valores dados $A=(209, 169, 131)$, $B=(89,133,60)$ y $C=(152,140,111)$

		c_1			c_2			c_3		
R	208	210	212	90	92	87	160	154	148	
G	165	170	172	130	138	128	143	146	137	
B	128	130	133	60	59	62	113	115	110	

Propuesta de solución

La red debe tener 3 neuronas en la capa de entrada puesto que los vectores poseen 3 componentes. Diseñamos una arquitectura de red con 2 capas ocultas y 3 y 2 neuronas respectivamente en cada capa. Finalmente asumimos que la salida posee 3 neuronas para hacerlas corresponderse con las tres clases. Esta arquitectura se muestra en la figura siguiente,



suponiendo una razón de aprendizaje de $\alpha = 0.5$. En todas las neuronas se utilizan las funciones logísticas de activación sigmoideal. Los valores de los vectores de entrada (R,G,B) se normalizan al rango $[0,1]$ dividiendo los valores originales por 255. La capa de salida posee tres neuronas con valores 0 ó 1 asociándose con los siguientes patrones de entrada (1,0,0) con patrones de la clase C_1 , (0, 1, 0) con patrones de la clase C_2 y (0, 0, 1) con patrones de la clase C_3 . Los pesos y bias entre la capa de entrada y la primera capa oculta se identifican como W_a , b_a entre las dos capas ocultas W_b , b_b y finalmente entre la última capa oculta y la de salida W_c , b_c . Los pesos y bias iniciales con los que se comienza el proceso de aprendizaje son los que se muestran a continuación. Según esta nomenclatura, las filas con valores W_{xij} indican el valor del peso de la conexión que une la neurona i con la j .

$$W_a = \begin{bmatrix} w_{a11} & w_{a21} & w_{a31} \\ w_{a12} & w_{a22} & w_{a32} \\ w_{a13} & w_{a23} & w_{a33} \end{bmatrix} \equiv \begin{bmatrix} -2.89 & 22.02 & -24.07 \\ 12.34 & -13.82 & -16.50 \\ 9.31 & -37.61 & -5.19 \end{bmatrix}$$

$$W_b = \begin{bmatrix} w_{b11} & w_{b21} & w_{b31} \\ w_{b12} & w_{b22} & w_{b32} \end{bmatrix} \equiv \begin{bmatrix} 5.26 & 4.66 & -0.63 \\ -1.08 & -4.34 & -5.46 \end{bmatrix}$$

$$W_c = \begin{bmatrix} w_{c11} & w_{c21} \\ w_{c12} & w_{c22} \\ w_{c13} & w_{c23} \end{bmatrix} \equiv \begin{bmatrix} -3.86 & -8.90 \\ 6.45 & -7.24 \\ -9.03 & 3.53 \end{bmatrix}$$

$$b_a(1) = 1.84; \quad b_a(2) = 7.11; \quad b_a(3) = 22.66; \quad b_b(1) = -8.17; \quad b_b(2) = 1.91$$

$$b_c(1) = 11.23; \quad b_c(2) = 0.3954; \quad b_c(3) = -2.0975$$

Con estos valores se inicia el proceso de entrenamiento, fijando como número máximo de iteraciones $1.2e^6$. El límite del error se fija en 0.1. Tras el entrenamiento se obtienen los siguientes valores para los pesos (parámetros aprendidos),

Por motivos de tiempo y complejidad del problema es que tuvimos que modificar un poco el problema inicial, y para eso, tomamos la decisión, en conjunto como equipo y tras la recomendación de la profesora, de reducir la complejidad del problema y eliminar una de las tres clases, en específico la clase SUELO.

2da. Propuesta del problema a desarrollar

Un sistema basado en visión por computador trata de clasificar dos tipos de texturas naturales basadas en las tres componentes de color (R, G, B). Los dos tipos de texturas corresponden a paisajes naturales en los que se pretende distinguir áreas de cielo azul y zonas boscosas con predominio de verdes. Se han extraído los siguientes datos de la imagen:

		c_1			c_2	
<i>R</i>	200	210	215	90	92	87
<i>G</i>	160	170	172	130	138	128
<i>B</i>	120	130	133	60	54	66

Asumiendo una razón de aprendizaje $\alpha = 10^{-4}$ obtener los pesos actualizados asumiendo que la convergencia termina cuando $|w(k+1) - w(k)| < \epsilon$, siendo $\epsilon = 10^{-4}$

Una vez obtenidos los pesos, clasificar las dos nuevas muestras siguientes:

$$\mathbf{A} \equiv (208, 170, 135) \text{ y } \mathbf{B} \equiv (89, 130, 60)$$

2da, Propuesta de solución

Como primera parte de la solución, mostramos la clase maestra de nuestro programa:

```
class MLP():
    # constructor
    def
    __init__(self,xi,d,w_a,w_b,us,uoc,precision,epocas,fac_ap,n_ocultas,n_entradas,n_salida):
        # Variables de inicialización
        self.xi = np.transpose(xi)
        self.d = d
        self.wa = w_a
        self.wb = w_b
        self.us = us
        self.uoc = uoc
        self.precision = precision
        self.epocas = epocas
        self.fac_ap = fac_ap
        self.n_entradas = n_entradas
```

```

self.n_ocultas = n_ocultas
self.n_salida = n_salida

# Variables de aprendizaje
self.di = 0 # Salida deseada en iteracion actual
self.error_red = 1 # Error total de la red en una conjunto de iteraciones
self.Ew = 0 # Error cuadratico medio
self.Error_prev = 0 # Error anterior
self.Errores = []
self.Error_actual = np.zeros((len(d))) # Errores acumulados en un ciclo de muestras
self.Entradas = np.zeros((1,n_entradas))
self.un = np.zeros((n_ocultas,1)) # Potencial de activacion en neuronas ocultas
self.gu = np.zeros((n_ocultas,1)) # Funcion de activacion de neuronas ocultas
self.Y = 0.0 # Potencial de activacion en neurona de salida
self.y = 0.0 # Funcion de activacion en neurona de salida
self.epochs = 0

# Variables de retropropagacion
self.error_real = 0
self.ds = 0.0 # delta de salida
self.docu = np.zeros((n_ocultas,1)) # Deltas en neuronas ocultas

def Operacion(self):
    respuesta = np.zeros((len(self.d),1))
    for p in range(len(self.d)):
        self.Entradas = self.xi[:,p]
        self.Propagar()
        respuesta[p,:] = self.y
    return respuesta.tolist()

def Aprendizaje(self):
    Errores = [] # Almacenar los errores de la red en un ciclo
    while(np.abs(self.error_red) > self.precision):
        self.Error_prev = self.Ew
        for i in range(len(d)):
            self.Entradas = self.xi[:,i] # Senales de entrada por iteracion
            self.di = self.d[i]
            self.Propagar()
            self.Backpropagation()
            self.Propagar()
            self.Error_actual[i] = (0.5)*((self.di - self.y)**2)

```



```

# error global de la red
self.Error()
Errores.append(self.error_red)
self.epochs +=1
# Si se alcanza un mayor numero de epocas
if self.epochs > self.epocas:
    break
# Regresar
return self.epochs,self.wa,self.wb,self.us,self.uoc,Errores

```

def Propagar(self):

```

# Operaciones en la primer capa
for a in range(self.n_ocultas):
    self.un[a,:] = np.dot(self.wa[a,:], self.Entradas) + self.uoc[a,:]

# Calcular la activacion de la neuronas en la capa oculta
for o in range(self.n_ocultas):
    self.gu[o,:] = tanh(self.un[o,:])

# Calcular Y potencial de activacion de la neuronas de salida
self.Y = (np.dot(self.wb,self.gu) + self.us)
# Calcular la salida de la neurona de salida
self.y = tanh(self.Y)

```

def Backpropagation(self):

```

# Calcular el error
self.error_real = (self.di - self.y)
# Calcular ds
self.ds = (dtanh(self.Y) * self.error_real)
# Ajustar wb
self.wb = self.wb + (np.transpose(self.gu) * self.fac_ap * self.ds)
# Ajustar umbral us
self.us = self.us + (self.fac_ap * self.ds)
# Calcular docu
self.docu = dtanh(self.un) * np.transpose(self.wb) * self.ds
# Ajustar los pesos wa
print(f'docu: {self.docu}')
print(f'wa: {self.wa}')
print(f'Entradas: {self.Entradas}')
print(f'fac_ap: {self.fac_ap}')

```

```

print(f'docu Trans: {(np.transpose(self.docu))}')
print(f'Entr * fac_ap: {self.Entradas * self.fac_ap}')
for j in range(self.n_entradas):
    self.wa[:,j] = self.wa[:,j] + (np.transpose(self.docu) * self.Entradas * self.fac_ap)

# Ajustar el umbral en las neuronas ocultas
for g in range(self.n_ocultas):
    self.uoc[g,:] = self.uoc[g,:] + (self.fac_ap * self.docu[g,:])

def Error(self):
    # Error cuadratico medio
    self.Ew = ((1/len(d)) * (sum(self.Error_actual)))
    self.error_red = (self.Ew - self.Error_prev)

```

La clase anterior nos va a permitir declarar el constructor y los métodos de nuestros objetos:

- `def __init__`
 - `self,`
 - `xi,`
 - `d,`
 - `w_a,`
 - `w_b,`
 - `us,`
 - `uoc,`
 - `precision,`
 - `epocas,`
 - `fac_ap,`
 - `n_ocultas,`
 - `n_entradas,`
 - `n_salida`
- `def Operacion`
- `def Aprendizaje`
- `def Propagar`
- `def Backpropagation`
- `def Error`

Continuamos el análisis con dos de las funciones más importantes del programa:

```
def obtener_vector_validacion_bosque_y_rango01():
    with open('boscosoRGB_completo.csv',newline='') as fp:
        data = list(csv.reader(fp))

        r,g,b = 0,0,0
        for row in data[1:]: # Skip the header row and convert first values to integers
            row[0] = int(row[0])
            r += row[0]
            row[0] = float(row[0])
            row[0] = row[0]/255
            row[1] = int(row[1])
            g += row[1]
            row[1] = float(row[1])
            row[1] = row[1]/255
            row[2] = int(row[2])
            b += row[2]
            row[2] = float(row[2])
            row[2] = row[2]/255
            row[3] = int(row[3])

        r = r/len(data[1:])
        g = g/len(data[1:])
        b = b/len(data[1:])

        print(f'Vector de validacion Bosque R: {r} G: {g} B: {b}')
        r = r/255
        g = g/255
        b = b/255
        validation_vector = [r,g,b,2]

        for row in data[1:]:
            row.extend(validation_vector)

    with open("boscoso_rango01.csv", "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerows(data)

def obtener_vector_validacion_cielo_y_rango01():
    with open('cieloRGB_completo.csv',newline='') as fp:
```

```

data = list(csv.reader(fp))

r,g,b = 0,0,0
for row in data[1:]: # Skip the header row and convert first values to integers
    row[0] = int(row[0])
    r += row[0]
    row[0] = float(row[0])
    row[0] = row[0]/255
    row[1] = int(row[1])
    g += row[1]
    row[1] = float(row[1])
    row[1] = row[1]/255
    row[2] = int(row[2])
    b += row[2]
    row[2] = float(row[2])
    row[2] = row[2]/255
    row[3] = int(row[3])

r = r/len(data[1:])
g = g/len(data[1:])
b = b/len(data[1:])

print(f'Vector de validacion Cielo R: {r} G: {g} B: {b}')
r = r/255
g = g/255
b = b/255
validation_vector = [r,g,b,1]

for row in data[1:]:
    row.extend(validation_vector)

with open("cielo_rango01.csv", "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerows(data)

```

Estas dos funciones son de vital importancia ya que se encargan de extraer de forma precisa los datos que en la primera práctica dedujimos, y nos referimos a los valores de RGB de cada uno de los píxeles de la imagen proporcionada por la profesora (Cielo azul, suelo y zona boscosa, pero para esta práctica únicamente se requirió trabajar con la zona del CIELO y BOSQUE), todo esto para poder almacenarlos en un vector que sea más fácil de manipular y recorrer.

IDENTIFICACIÓN DE TEXTURAS EN UNA IMAGEN DIGITAL

Nos apoyamos de este formato de documento **.csv**, muy utilizado y requerido en Contaduría por la sencillez con la que se organizan los valores, ya que en vez de trabajar con tablas, la información se divide por comas y así es menor el espacio que requiere.

Por último, llegamos al main del programa:

```
# Programa principal
if "__main__" == __name__:
    # Carga de los datos
    obtener_vector_validacion_bosque_y_rango01()
    obtener_vector_validacion_cielo_y_rango01()

    #Carga de los 3 archivos en uno
    data = data2 = ""

    # Reading data from boscoso
    with open('boscoso_rango01.csv') as fp:
        data = fp.read()

    # Reading data from cielo
    with open('cielo_rango01.csv') as fp:
        data2 = fp.read()
        data2 = data2[1:] #Eliminando la primer línea

    # Merging 2 files
    # To add the data of file2
    # from next line
    data += data2

    with open('rgb_bc_juntos.csv', 'w') as fp:
        fp.write(data)

    datos = pd.read_csv('rgb_bc_juntos.csv', low_memory=False)
    matrix_data = np.array(datos)

    #Datos de entrada
    x_inicio = 0
    x_n = 2
    #Datos de entrada validación
    xj_inicio = 4
    xj_n = 6
```

```

# Crear vector de entradas xi
xi = (Datos_entrenamiento(matrix_data,x_inicio,x_n))
d = matrix_data[:,x_n+1]
# Vector de validación
xj = (Datos_validacion(matrix_data,xj_inicio,xj_n))

# Parametros de la red
f, c = xi.shape
fac_ap = 0.5 #Factor de aprendizaje
precision = 0.1 #Precision inicial
epocas = 484 #Numero maximo de epocas (1.2e^6) = 484.1145
epochs = 0 #Contador de epocas utilizadas

# # Arquitectura de la red
n_entradas = c # Numero de entradas
cap_ocultas = 1 # Dos capa oculta
n_ocultas = 2 # Neuronas en la capa oculta
n_salida = 1 # Neuronas en la capa de salida

# # Valor de umbral o bia
us = 1.0 # umbral en neurona de salida
uoc = np.ones((n_ocultas,1),float) # umbral en las neuronas ocultas

# # Matriz de pesos sinapticos
random.seed(0)
w_a = random.rand(n_ocultas,n_entradas)
w_b = random.rand(n_salida,n_ocultas)

#Inicializar la red PMC
red = MLP(xi,d,w_a,w_b,us,uoc,precision,epocas,fac_ap,n_ocultas,n_entradas,n_salida)
epochs,wa_a,wb_a,us_a,uoc_a,E = red.Aprendizaje()

# graficar el error
plt.grid()
plt.ylabel("Error de la red",fontsize=12)
plt.xlabel("Épocas",fontsize=12)
plt.title("Perceptrón Multicapa",fontsize=14)
x = np.arange(epochs)
plt.plot(x,E,'b',label="Error global")

```

IDENTIFICACIÓN DE TEXTURAS EN UNA IMAGEN DIGITAL

```
plt.legend(loc='upper right')  
plt.show
```

```
# validation
```

```
red = MLP(xi,d,wa_a,wb_a,us,uoc,precision,epocas,fac_ap,n_ocultas,n_entradas,n_salida)
```

```
salidas = red.Operation()
```

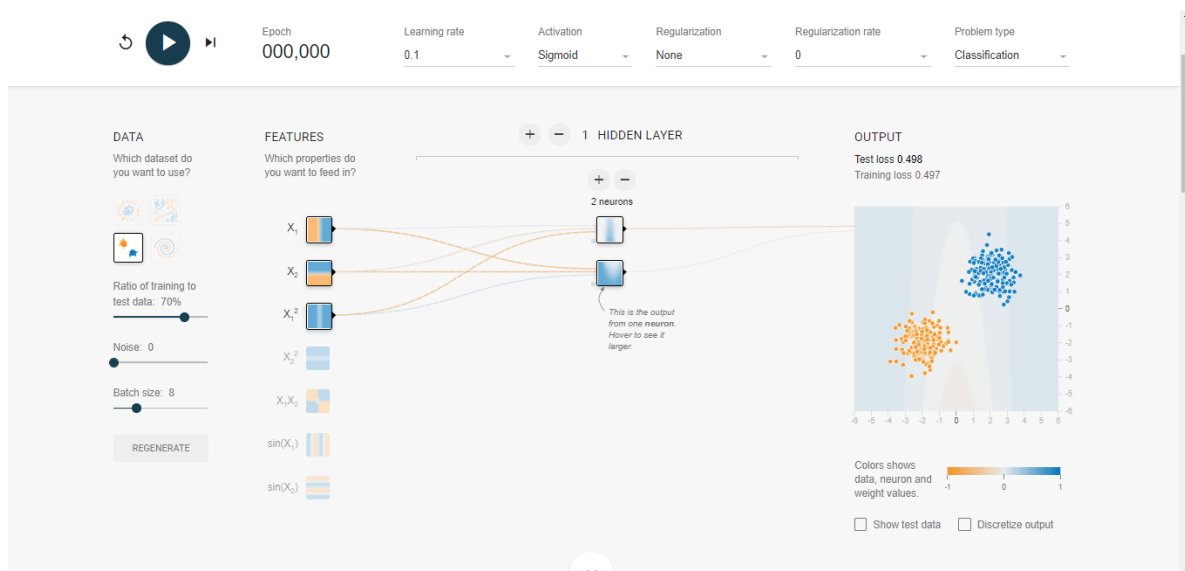
```
print("Salidas: ",salidas)
```

```
print("Epochs: ", epochs)
```

Indagando un poco en internet, encontramos esta herramienta de prueba:

<https://playground.tensorflow.org/>

La cual, nos permitió darnos una idea más gráfica del Perceptrón que programamos:



Resultados

Como muestra el ejemplo que nos proporcionó la profesora, la salida del clasificador debe ser:
O pertenece a la Clase 1 o pertenece a la Clase 2.

$$w^1 A = 10^{-3} \begin{pmatrix} 29 & -37 & 14 \end{pmatrix} \begin{pmatrix} 208 \\ 170 \\ 135 \end{pmatrix} = 1.6936 > 0 \Rightarrow A \text{ pertenece a la clase } c_1$$

$$w^2 B = 10^{-3} \begin{pmatrix} 29 & -37 & 14 \end{pmatrix} \begin{pmatrix} 89 \\ 130 \\ 60 \end{pmatrix} = -1.3732 < 0 \Rightarrow B \text{ pertenece a la clase } c_2$$

En caso de que el vector desconocido o vector de prueba, sea mayor a 0, este va a pertenecer a la clase 1, y en caso contrario, el vector va a pertenecer a la clase 2.

Precisamente, para comprobar la robustez del clasificador, introducimos los mismos vectores de prueba al perceptrón:

$$A = (208, 170, 135) \text{ y } B = (89, 130, 60)$$

A continuación, se presentan capturas de pantalla de las correspondientes pruebas de escritorio:

```

1 import numpy as np
2 import pandas as pd
3 from numpy import random
4 import matplotlib.pyplot as plt
5 import csv
6
7 class MLP():
8     # Constructor
9     def __init__(self, xi, d, w_a, w_b, w_c, uoc, precision, epocas, fac_ap, n_ocultas, n_ocu
10         # Variables de inicialización
11         self.xi = np.transpose(xi)
12         self.d = d
13         self.wa = w_a
14         self.wb = w_b
15         self.wc = w_c
16         self.us = uoc
17         self.uoc = uoc
18         self.precision = precision
19         self.epocas = epocas
20         self.fac_ap = fac_ap
21         self.n_entradas = n_entradas
22         self.n_ocultas = n_ocultas
23         self.n_ocultas2 = n_ocultas2
24         self.n_salida = n_salida
25
26     # Variables de aprendizaje
27     self.di = 0 # Salida deseada en iteracion actual
28     self.error_red = 1 # Error total de la red en una conjunto de iteraciones
29     self.Ew = 0 # Error cuadrático medio
30     self.Error_prev = 0 # Error anterior
31     selfErrores = []
32     self.Error actual = np.zeros((len(d))) # Errores acumulados en un ciclo de mue
33     self.Entradas = np.zeros((1,n_entradas))
34     self.un = np.zeros((n_ocultas,1)) # Potencial de activación en neuronas ocula
35     self.un2 = np.zeros((n_ocultas2,1)) # Potencial de activación en neuronas ocula
36     self.gu = np.zeros((n_ocultas,1)) # Función de activación de neuronas ocula

```

```

In [2]: runfile('C:/Users/georg/Desktop/ESCOM/7mo. Semestre/RP/Perceptron/Codigo/Prueba.py', wdir='C:/Users/georg/Desktop/ESCOM/7mo. Semestre/RP/Perceptron/Codigo')

CARGANDO CONJUNTO DE MUESTRAS DE APRENDIZAJE DEL CIELO...
CARGANDO CONJUNTO DE MUESTRAS DE APRENDIZAJE DEL BOSQUE...
FIN DE LA ETAPA DE APRENDIZAJE...

A = (208, 170, 135)
B = (89, 130, 60)

El vector A pertenece a la Clase 1
El vector B pertenece a la Clase 2

In [3]:

```


Conclusiones

La teoría de Redes Neuronales Artificiales, presenta grandes ventajas con respecto a otros modelos típicos de solución de problemas de Ingeniería, una de ellas es su inspiración en modelos biológicos del funcionamiento del cerebro, lo que facilita su estudio debido a las analogías que pueden introducirse para su análisis.

Los modelos matemáticos en que han sido desarrollados los algoritmos para todos los tipos de redes son modelos sencillos, que aunque exigen cierto grado de conocimientos de cálculo diferencial, pueden ser asimilados y desarrollados en cualquier lenguaje de programación.

Por otro lado, las estructuras de las redes se han definido por medio de notación sencilla y comprensible, cada nuevo desarrollo permite cierta flexibilidad en cuanto a la forma final de la red y esto garantiza su fácil adaptación a aplicación particulares.

Las redes neuronales son una teoría relativamente nueva que junto a otras técnicas de inteligencia artificial ha generado soluciones muy confiables a problemas de Ingeniería, los cuales a pesar de poder ser solucionados por métodos tradicionales, encuentran en las redes neuronales una alternativa fácil de implementar y altamente segura. En el proceso de entrenamiento de las aplicaciones desarrolladas en este proyecto se observó:

- Al escoger las redes neuronales como método de solución de un problema en particular, es necesario tener un entendimiento profundo y completo de lo que es el problema como tal, pues ello facilita la elección de los patrones de entrenamiento y ofrece una idea general de la arquitectura que debe tener la red y de lo que se espera de ella.
- Después de haber escogido el tipo de red con que se solucionará un problema, no hay ningún criterio establecido para decidir la arquitectura final de la red, la elección del número de capas que la componen y el número de neuronas de cada una de ellas es fijado por la experiencia del diseñador, y muchas veces realizado por un método de ensayo y error.
- Una manera fácil para escoger el tipo de red que debe implementarse, es analizar si el problema al que se está enfrentado cuenta con un conjunto de salida conocido, esta condición restringe las opciones a dos categorías, las redes con aprendizaje supervisado dentro de las cuales se destacan: El Perceptrón, Adaline, Backpropagation, CPN (Counterpropagation), Boltzmann Machine y Cauchy Machine y la siguiente categoría conformada por las redes de aprendizaje no supervisado donde pueden contarse: Las memorias asociativas, red de Hopfield, redes de aprendizaje competitivo como el LVQ (Learning Vector Quantization) y las SOM (Self-Organizing Maps), red de Grossberg, ART (Adaptive Resonance Theory) y La FAM (Fuzzy Associative Memory) que es una combinación entre aprendizaje asociativo y lógica difusa.

- La red tipo Perceptrón es una red que puede implementarse exitosamente para resolver problemas de clasificación de patrones que sean linealmente separables, la red responderá mejor entre más sencillos sean los patrones que debe clasificar. A pesar de que cuenta con serias limitaciones, esta red conserva su importancia ya que sirvió como inspiración para otros tipos de redes, como por ejemplo las redes multicapa.
- Investigaciones prácticas han demostrado que al solucionarse un problema de clasificación de patrones por medio de una red Perceptrón multicapa, no se necesita incorporar más de tres capas (dos capas ocultas y la capa de salida) a la red para que ésta solucione correctamente el problema, pues una red de estas características está en capacidad de generar regiones de decisión arbitrariamente complejas; aunque en ciertos problemas se puede simplificar el aprendizaje aumentando capas ocultas, la tendencia es aumentar la extensión de la función de activación, en lugar de la complejidad de la red.
- Con el nacimiento del algoritmo LMS se logró un gran avance en la minimización del error medio cuadrático de los algoritmos de paso descendente; con base en este método se desarrolló la red Adaline cuya aplicación más común es el filtro adaptivo, caso especial en que la red tiene solo una señal de entrada, muestreada en varios instantes de tiempo de forma que cada muestra representa un grado de libertad que se utiliza para ajustar la señal de entrada a la salida deseada. Una característica especial de este tipo de red es que no necesita un gran número de muestras para obtener buenos resultados, garantizando con esto una rápida convergencia.
- Con base en el algoritmo LMS se desarrolló también el algoritmo Backpropagation, adecuado de tal manera que esta red puede ser aplicada para resolver problemas complejos de aproximación de funciones obteniéndola mayoría de las veces excelentes resultados. La característica más importante de esta red es su capacidad de generalización, es decir su facilidad para entregar una salida satisfactoria a entradas que el sistema nunca vio en su fase de entrenamiento, esta característica le da una gran versatilidad permitiendo que esta red sea implementada en aplicaciones como: codificación de información, traducción de texto en lenguaje hablado, reconocimiento de lenguaje hablado, reconocimiento óptico de caracteres (OCR), además de un sin número de aplicaciones médicas especialmente en cardiología.
- Cuando se está enfrentado a un problema de clasificación de patrones los cuales están agrupados por clases y subclases, el proceso de entrenamiento de la red por medio de aprendizaje competitivo es el más apropiado, este tipo de red es altamente confiable para ubicar patrones en la clase más indicada dependiendo de sus características, además algunas modificaciones como la red LVQ permiten combinar aprendizaje supervisado y no supervisado para garantizar una clasificación exitosa de grupos de patrones con gran número de componentes y características dispares.
- La principal ventaja de las redes neuronales es su capacidad para aprender de la experiencia, de generalizar de casos anteriores a nuevos casos, de abstraer características esenciales a partir de entradas que representan información irrelevante; en términos generales las redes neuronales son una teoría relativamente nueva y como

tal presentan aún algunas limitaciones, pero su facilidad de implementación y la calidad en la información que entregan como respuesta, son la motivación suficiente para que su estudio y desarrollo continúe.

Referencias

- [1] A Brief Overview of Deep Learning:
<http://yyue.blogspot.com.es/2015/01/a-briefoverview-of-deep-learning.html>
- [2] Classifying MNIST digits using Logistic Regression:
<http://deeplearning.net/tutorial/logreg.html>
- [3] Convolutional Networks: <http://www.iro.umontreal.ca/~bengioy/dlbook/convnets.html>
- [4] Convolutional Neural Networks: <http://www.deeplearning.net/tutorial/lenet.html>
- [5] Convolutional Neural Networks for Visual Recognition:
<http://cs231n.github.io/convolutional-networks/>
- [6] El aprendizaje profundo para la identificación de sistemas no lineales:
<http://www.ctrl.cinvestav.mx/~yuw/pdf/MaTesER.pdf>
- [7] Oleszak, M. (2020, 3 julio). Linear Classifiers: An Overview - Towards Data Science. Medium. <https://towardsdatascience.com/linear-classifiers-an-overview-e121135bd3bb>
- [8] Perceptrón multicapa Archivado el 14 de julio de 2014 en la Wayback Machine., Redes de Neuronas Artificiales, UC3M, RAI 2012.
- [9] Haykin, Simon (1998). Neural Networks: A Comprehensive Foundation (2 edición). Prentice Hall. ISBN 0-13-273350-1.