Sistema de reconocimiento de dígitos: Lector de NIA

Digit recognition system: NIA Reader

Diego Álvarez Ponce, José Luis Ayuso Baena
Universidad Carlos III de Madrid
100346828@alumnos.uc3m.es
100349472@alumnos.uc3m.es

Resumen: En este artículo se desarrolla el estudio de un sistema de reconocimiento de números basado en la utilización del algoritmo de aprendizaje supervisado SVC. En concreto, empleamos dicho reconocimiento de números en la lectura de los NIA universitarios de los alumnos.
Palabras clave: Algoritmos, SVC, Inteligencia Artificial, Reconocimiento Dígitos, NIA.

Abstract: On this paper, a study on number recognition based on the usage of the supervised learning algorithm SVC is developed. In particular, that number recognition is used in order to read students' NIAs.
Keywords: Algorithms, SVC, Artificial Intelligence, Digit Recognition, NIA.

1 Introducción

El desarrollo de la inteligencia artificial (IA) ha supuesto un gran avance para la tecnología basada en el reconocimiento de patrones con ejemplos como la lectura de matrículas a la entrada de los parkings o la digitalización de textos manuscritos.

Para llevar estos ejemplos a la práctica, se emplean algoritmos de aprendizaje supervisado, capaces de construir modelos que, tras haber sido entrenados por conjuntos de datos similares, pueden realizar clasificaciones acertadas sobre las muestras que se evalúan. Esta forma de programación supone una gran ventaja sobre los antiguos métodos de implementación.

2 Machine learning

2.1 Aprendizaje supervisado

El aprendizaje supervisado es un tipo de algoritmo de Machine learning que permite crear modelos predictivos a partir de un conjunto de datos de entrenamiento. Dentro de este campo, existen dos tipos de aprendizaje distintos:

2.1.1 Regresión

Los valores numéricos obtenidos siguen una distribución continua como se muestra en la Figura 1.

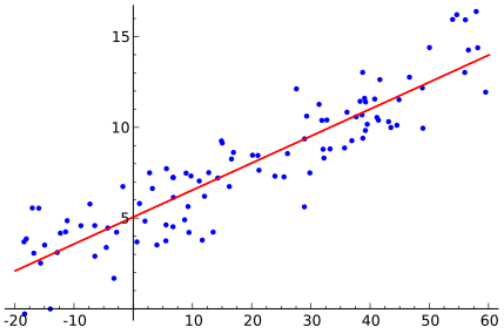

Figura 1: Ejemplo regresión.

2.1.2 Clasificación

Este tipo de aprendizaje se utiliza cuando los datos de entrada pueden separarse en distintas clases. El algoritmo encuentra patrones que le permiten clasificar los elementos en los diferentes grupos. En la Figura 2 se muestra un ejemplo de este tipo de clasificación.

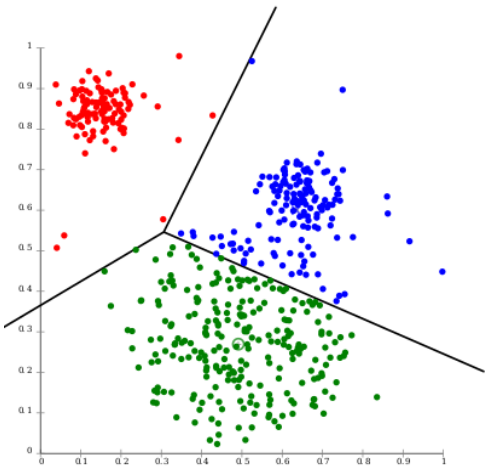

Figura 2: Ejemplo clasificación.

Entre los algoritmos de clasificación más comunes, hemos empleado las máquinas de vectores soporte (SVM).

2.2 Support vector machines

Este algoritmo creado por Vladimir Vapnik y su equipo de AT&T, perteneciente a los conocidos como métodos kernel, se basa en la programación matemática para la clasificación binaria o multi-categoría de los datos.

Las SVM son capaces de construir un hiperplano óptimo, utilizado en forma de superficie de decisión, que permite que las clases de datos se separen lo máximo posible entre ellas. Por su parte, el denominado kernel, no es más que el espacio de estos atributos, que se transforma de forma no lineal para poder adaptarse a la distribución de las distintas clases.

Dentro de este ámbito, se utilizan además diversas técnicas para su implementación, escogiendo entre todas ellas el modelo SVC.

2.3 Aplicaciones de las SVM

Existe una gran variedad de aplicaciones para este algoritmo. A continuación, se darán algunos ejemplos presentes en la vida cotidiana.

2.3.1 Reconocimiento de voz

En la actualidad, ya existen dispositivos que permiten recoger grabaciones de voz de distintos usuarios, diferenciando el tono entre ellos, y ejecutando las acciones requeridas por los mismos.

Un ejemplo de esto es 'Alexa'. Un servicio lanzado al mercado por Amazon, que realiza tareas como la configuración de alarmas y temporizadores, la gestión de listas de tareas y compras o el aporte de información sobre el tiempo o estado del tráfico de una ciudad.

Figura 3: Dispositivo de Amazon.

2.3.2 Reconocimiento facial

En el último año, los desarrolladores de teléfonos móviles han comenzado a sustituir el desbloqueo mediante huella dactilar por el reconocimiento facial de la persona.

El sistema se entrena mediante una serie de fotografías del rostro del usuario en distintos ángulos, siendo capaz de extrapolarlas para crear una imagen más general que se adapte a pequeños cambios como la utilización de gafas, poca luminosidad en el ambiente o cambios en el peinado y la barba.

2.3.3 Coches autónomos

Empresas como la marca Tesla están revolucionando el sector automovilístico con la incorporación de sistemas de conducción autónoma (AutoPilot), que funcionan mediante la utilización de sensores, cámaras y posicionamiento GPS.

El coche es capaz de adaptarse al entorno, tomando las decisiones oportunas en cada instante durante la conducción. A continuación, se muestra como funciona el sistema de reconocimiento de obstáculos.

Figura 4: Reconocimiento de obstáculos.

2.3.4 Lectura de imágenes

Dentro de este campo, existen diversas aplicaciones que comprenden tareas desde la lectura de matrículas hasta la digitalización de documentos.

La aplicación que hemos desarrollado está fuertemente relacionada con este tipo de utilidad y se detallará en los siguientes apartados.

3 Desarrollo del lector de NIA

La aplicación lectora de NIAs ha sido implementada en Python por su simplicidad a la hora de trabajar con sets de datos. Se han seguido los siguientes pasos.

3.1 Entrenamiento de la SVC

En primer lugar, para el entrenamiento de la máquina, hemos cargado un set de datos disponible en los datasets de sklearn. Esta base de datos está compuesta por una gran cantidad de imágenes de números escritos a mano. Cada una de ellas tiene un tamaño de 8x8 píxeles.

Además, vienen acompañadas por su correspondiente target, necesario a la hora de relacionar la imagen con el dato correcto. En la Figura 5 se muestran algunos ejemplos de datos cargados, acompañados en la esquina inferior izquierda por su valor correspondiente.

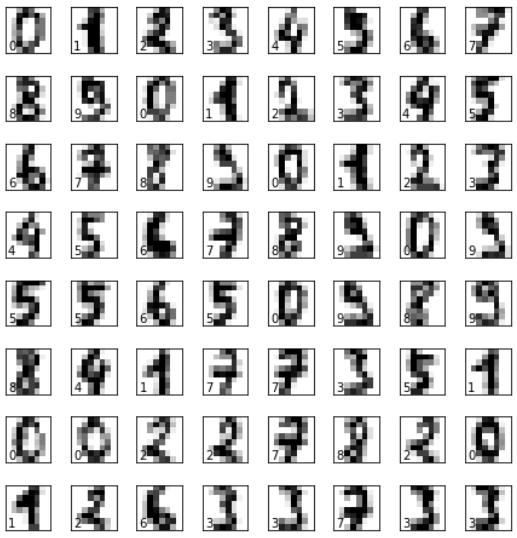

Figura 5: Representación del dataset.

Para comprobar la tasa de acierto que tiene nuestra máquina, hemos separado el conjunto en datos de entrenamiento y datos de test. El primer grupo será mayor que el segundo para que pueda aprender la mayor cantidad de ejemplos posibles, dedicando por ello, el 80% al entrenamiento y el 20% restante a las pruebas, como se adjunta a continuación.

```
N_train = np.round(0.8 * N).astype(np.int32)
N_test = np.round(0.2 * N).astype(np.int32)

mask = np.random.permutation(len(digits.images))

np.random.seed(seed = 1234)
x_train = data[list(mask[0:N_train])]
y_train = digits.target[list(mask[0:N_train])]

x_test = data[list(mask[N_train:-1])]
y_test = digits.target[list(mask[N_train:-1])]
```

Figura 6: Segmentación de los datos.

Tras este paso, hemos entrenado la máquina y predicho los resultados con los datos de test.

Figura 7: Predicción del dataset original.

Como se puede comprobar en los dígitos mostrados, no se ha cometido ningún error a la hora de predecir los valores. No obstante, calcularemos la matriz de confusión.

```
Confusion matrix:
[[34  0  0  0  0  0  0  0  0  0]
 [ 0 37  0  0  0  0  0  0  0  0]
 [ 0  0 37  0  0  0  0  0  0  0]
 [ 0  0  0 34  0  0  0  0  0  0]
 [ 0  0  0  0 31  0  0  0  0  0]
 [ 0  0  0  0  0 39  0  0  0  2]
 [ 0  0  0  0  0  0 35  0  0  0]
 [ 0  0  0  0  0  0  0 33  0  0]
 [ 0  2  0  0  0  0  0  0 40  1]
 [ 0  0  0  0  0  0  0  0  0 33]]
```

Figura 8: Matriz de confusión.

Podemos ver como en la diagonal principal se intersectan los valores reales con los predichos por la máquina. En la mayoría de los casos, ambos números coinciden, lo cual es síntoma de un correcto funcionamiento.

Cabe destacar que se cometen pequeños errores entre algunos números, como es el caso del 8, que se confunde con el 1 y el 9 en ciertas ocasiones.

Determinamos que el funcionamiento del algoritmo, con estos datos, es bastante bueno, debido al alto porcentaje de acierto obtenido.

Ahora sustituiremos los datos de test por las imágenes de los NIAs realizadas por nosotros, por lo que utilizaremos la base de datos completa como entrenamiento.

Finalmente, entrenaremos la máquina como se muestra a continuación.

```
clf = svm.SVC(gamma=0.001, C=100)
clf.fit(x_train, y_train)
```

Figura 9: Entrenamiento de la máquina.

3.2 Lectura de NIA a ordenador y segmentación

Inicialmente, para reducir el número de errores que obtendríamos al cargar dígitos escritos a mano, utilizaremos números a ordenador. De esta forma, nos aseguramos de que el fondo es completamente blanco y que las cifras pueden observarse con mayor claridad. En la Figura 10 se presenta el NIA introducido.

Figura 10: NIA introducido a ordenador.

Para efectuar la segmentación de la imagen en los distintos dígitos, vamos a sumar los valores de las columnas. Se sabe que los colores más claros tendrán en sus componentes RGB valores más próximos a 255, mientras que los más oscuros estarán próximos al cero.

Los puntos comprendidos en la franja verde de la Figura 11, situada al 90% del valor máximo, serán considerados como 1 y el resto serán 0.

A continuación, representamos la gráfica con los valores de las columnas sumados, donde los picos más altos corresponderán a los espacios entre dígitos.

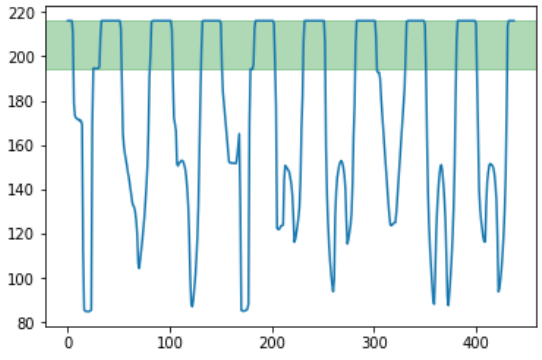

Figura 11: Gráfica de color.

Como se comentó anteriormente, el color verde hace referencia a los valores a partir de los cuales calcularemos los puntos donde recortaremos la imagen. Estos puntos (separadores) serán la distancia media entre los dos extremos de cada pico, para dejar a ambos lados la misma distancia en blanco, quedando centrado el número en cada imagen.

El paso a binario de la gráfica anterior se ha realizado con sentencias condicionales y lógica booleana. Por último, se calcula el punto medio. El código de ambos procesos aparece detallado en la Figura 12.

```
for j in range(image.shape[1]):
    if (binario[0,j] == 1) & (subo == False):
        subo = True
        inicio.append(j)

    if(binario[0,j] == 0) & (subo == True):
        subo = False
        fin.append(j)

    if (j == image.shape[1]-1):
        fin.append(j)

for k in range(len(inicio)):
    punto_medio = int((inicio[k]+fin[k])/2)
    separador.append(punto_medio)
```

Figura 12: Paso a binario y punto medio.

Una vez obtenidos los separadores, cogeremos los píxeles comprendidos entre dos separadores contiguos, obteniendo como resultado los distintos dígitos representados individualmente. La imagen que quedaría tras la segmentación sería la siguiente:

Figura 13: Dígitos a ordenador segmentados.

3.3 Comprobación del NIA

Originalmente, la máquina fue entrenada con imágenes de 8x8 píxeles en escala de grises. Para realizar la validación de los NIAs, debemos transformar a grises las imágenes recortadas y así obtener la misma estructura.

También se cambiarán los valores de los píxeles, comprendidos hasta ahora entre 0 y 1, por valores entre 0 y 16 como en la base de datos de entrenamiento. Acto seguido, debemos invertir los valores, pasando a tener el color blanco en el valor cero y el negro en el valor dieciséis.

Finalmente, eliminaremos los más bajos que obtengamos, pasando a valer cero. De esta forma, dejaremos solo el número y el resto de la imagen en blanco.

Todos estos pasos se han realizado mediante un único bucle que se detalla en la Figura 14.

El limitador marca a partir de qué valor, todos los números que hay por debajo de él se convierten en cero. Si bajamos el limitador, aparecen más grises alrededor del número y pueden darse predicciones erróneas. En contraposición, si subimos este valor, comenzarán a recortarse los grises, pudiendo perderse información del dígito.

Cada imagen transformada, se guarda en un vector que utilizaremos como datos de prueba para predecir sus valores. Con la sentencia que se presenta en la Figura 15 realizaremos esta acción.

```
NIA = clf.predict(array)
```

Figura 15: Predicción del NIA

Una vez obtenidos los dígitos predichos por la máquina pasamos a representarlos en la Figura 16, junto con las imágenes transformadas.

```
El NIA del alumno es:
1 2 3 4 5 6 7 8 9
```

Figura 16: NIA calculado.

Observamos que se predice a la perfección el NIA introducido a ordenador.

3.4 Lectura de NIA manuscrito

3.4.1 Imagen original

Una vez probado el funcionamiento de la máquina para el caso ideal, pasamos a cargar números hechos a mano. Nuestro ejemplo será el mostrado en la Figura 17.

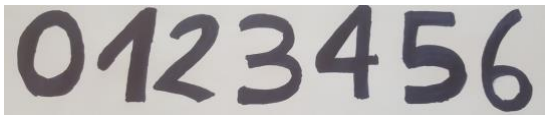

Figura 17: NIA manuscrito introducido.

La gráfica de color para este caso presenta variaciones con formas más extrañas debido a

```
limitador = 6.0

for k in range (len(numeros)):
    h,w,c = resized[k].shape
    matrix = np.zeros((h,w))

    for i in range(h):
        for j in range(w):
            numero = resized[k]
            matrix[i,j] = int(16-(((numero[i,j,0]+numero[i,j,1]+numero[i,j,2])/3)*16))
            if(matrix[i,j] < limitador):
                matrix[i,j] = 0

    digits.append(matrix)
```

Figura 14: Bucle para la tranformación de la imagen.

que, al realizar la foto, el fondo no es blanco, sino que tiende a gris por la iluminación.

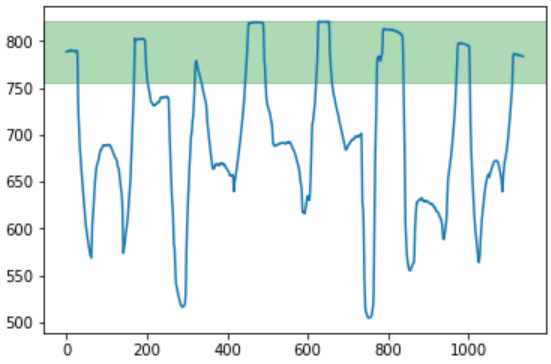

Figura 18: Grafica de color.

Por tanto, para poder segmentar los números correctamente, hemos tenido que bajar el porcentaje por el que multiplicamos el valor máximo. La segmentación quedaría de la siguiente manera:

Figura 19: Dígitos a mano segmentados.

Al estar muy próximos los números 1 y 2, y debido a su vez al color gris del fondo, parte del número 2 se recorta en la misma imagen que el número 1.

Aun así, cuando realizamos la predicción de los valores, la tasa de error obtenida es relativamente baja, fallando en algunos números. Se puede observar en la Figura 20.

```
El NIA del alumno es:
0 1 8 9 9 5 6
```

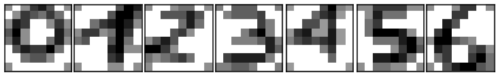

Figura 20: NIA calculado.

Como era de esperar, se obtiene un error mayor que en el caso ideal.

3.4.2 Imagen en blanco y negro

Con el fin de eliminar el fondo gris, hemos editado la imagen con el NIA para pasarla a blanco y negro, quedando de la siguiente forma:

Figura 21: NIA a mano en blanco y negro.

La gráfica de color de la Figura 22, aunque muy similar a la expuesta anteriormente, mejora, haciendo que la suma de las componentes RGB sea mayor, es decir, próxima al blanco.

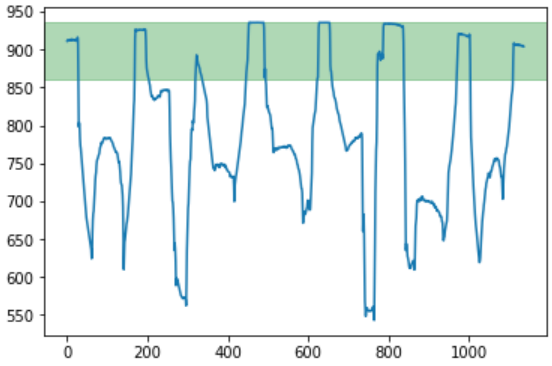

Figura 22: Grafica blanco y negro.

La segmentación para este aparado quedará de la misma forma que para la Figura 19, pero con el fondo en blanco.

Figura 23: Segmentación en blanco y negro.

El NIA que estimará la máquina en este caso será el que aparece a continuación:

```
El NIA del alumno es:
0 1 8 3 9 5 6
```

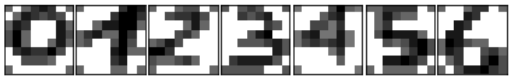

Figura 24: NIA en blanco y negro calculado.

Como puede observarse, la predicción es más acertada que la obtenida en la Figura 20. Gracias al blanco y negro, hemos conseguido aumentar en uno el número de dígitos acertados.

4 Posibles mejoras

Como ya hemos visto en los apartados anteriores, la diferencia entre el caso ideal y el real es relativamente significativa. Ya se detalló que el cambio a blanco y negro mejoraba las prestaciones del algoritmo puesto que uno de los factores más significativos a tener en cuenta para el reconocimiento de la imagen es la luminosidad. Cuando esta es baja, el fondo es grisáceo y no se pueden distinguir los dígitos con claridad.

Para seguir mejorando la predicción, se podría validar el NIA reconocido con una base de datos y cambiar posibles números erróneos por los que se ajusten a algún NIA existente semejante.

Además, se podrían ajustar automáticamente los valores de la altura y del limitador en función de la imagen cargada. De esta forma, crearíamos un programa flexible que se adapta a cualquier situación, sin tener que realizar cambios de forma manual.

Otra opción habría sido cargar un set de datos propios hechos a mano en lugar de importar datasets de internet. Así, los números podrían reconocerse de forma más exacta al tener todos el mismo tipo de escritura.

Finalmente, los valores de los atributos 'gamma' y 'c' que recibe el algoritmo SVC pueden establecerse con otros valores mediante el uso de validación cruzada. En la Figura 25 se han elegido distintos valores de 'c' y 'gamma' para la predicción del cero. Un menor valor de gamma implica una mayor distancia entre las observaciones que separan subespacios del SVM, luego la estimación es más conservadora. Sin embargo, un parámetro mayor hace que las predicciones estén menos suavizadas.

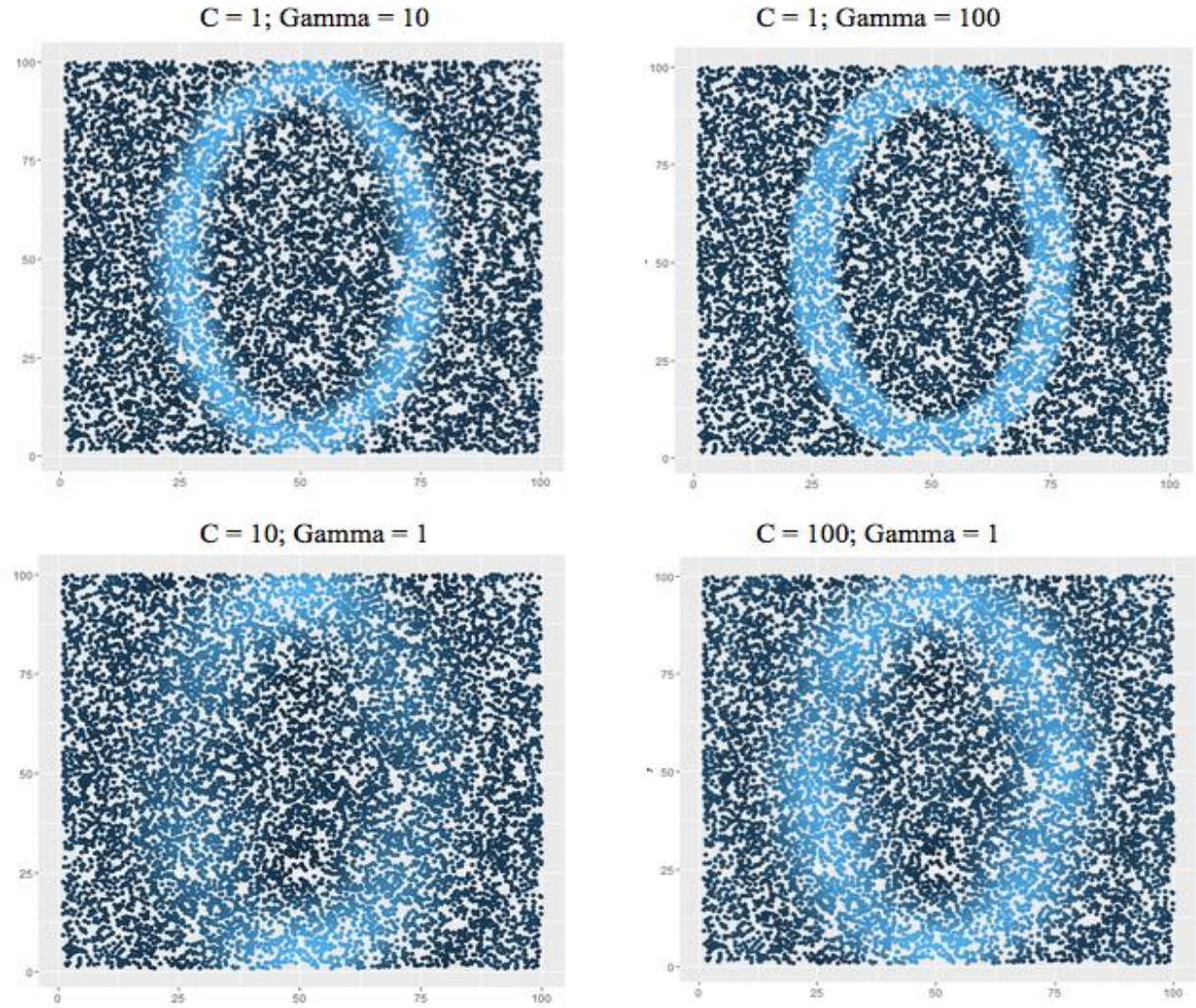

Figura 25: Predicción del cero en función de 'gamma' y 'c'

Estos parámetros varían conforme a la Ecuación 1.

$$y(\mathbf{x}) = \sum_{i=1}^{N} w_i \, \varphi(\|\mathbf{x} - \mathbf{x}_i\|),$$

Ecuación 1: Parámetros de la SVM.

5 Conclusión

Gracias al aprendizaje máquina se ha conseguido implementar aplicaciones basadas en el reconocimiento de patrones de una forma relativamente sencilla en comparación con el uso de la programación clásica. Por tanto, se abre ante nosotros un amplio abanico de posibilidades y tecnologías que marcarán el desarrollo y el progreso en el futuro.

Cada día, estas tecnologías están más presentes y son más familiares dentro del ámbito cotidiano, permitiendo una mayor agilidad en la realización de nuestras tareas.

Con un sistema de reconocimiento de NIAs como el que hemos implementado en este artículo, se pueden aligerar, por ejemplo, los procesos de corrección de exámenes.

La máquina es capaz de pensar y tratar los dígitos de forma similar a como funciona el cerebro humano. Crea una red neuronal que aprende y es capaz de reconocer patrones, para, tras recibir unos datos nuevos que desconoce, poder predecir un resultado.

A pesar de tratarse de una aplicación bastante básica y con infinidad de posibles mejoras, se ha comprobado que el rendimiento ofrecido es bastante bueno. Por ello, comprobamos que esta tecnología es altamente fiable y con infinidad de aplicaciones tanto en el presente, como en el futuro no muy lejano.

6 Bibliografía

Aprendizaje supervisado, Matlab &Simulink - MathWorks.

Máquina de vectores de soporte (SVM), Matlab &Simulink - MathWorks.

Juan Zambrano, 2018. Machine Learning: ¿Aprendizaje supervisado o no supervisado? Diferencias dentro del machine learning.

E.Alpaydin, J. 1998, sklearn datasets load digits - scikit-learn, 0.20.3 documentation.

Rvaquerizo, 2016. Análisis y decisión: El parámetro gamma, el coste, la complejidad de un SVM.

Pablo Martínez Olmos, ATDST, 2018-19. Introduction to Support Vector Machines.