DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 2

Clasificación y validación cruzada

10 de marzo de 2024 — Laboratorio de Datos

Monty Python

Integrante	LU	Correo electrónico
Kiara Yodko	785/23	kiarayodko9@gmail.com
Juan Cruz Mendoza	627/23	juancruzmendoza04@gmail.com
Manuel Gutman	923/23	gutmanmanuel@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

1. Introducción

En este trabajo, analizaremos una fuente de datos llamada Sign Language MNIST [1], la cual es un conjunto de imágenes que representan letras en el lenguaje de señas americano.

En primer lugar, llevaremos a cabo un análisis exploratorio en donde intentaremos entender mejor la composición de los datos y encontrar qué atributos de las imágenes pueden llegar a ser más relevantes para predecir a qué letra corresponden. A partir de esto, construiremos distintos modelos predictivos de K-Nearest-Neighbors (KNN) y Árboles de Decisión para poder clasificar las imágenes, utilizando K-fold Cross-Validation para evaluarlos y compararlos.

2. Análisis exploratorio

2.1. Características generales

En primera instancia, sabemos que la fuente de datos contiene 27.455 imágenes de 28x28 píxeles (784 atributos), todas en escala de grises, es decir, valores del 0 a 255 (por lo que no será necesario reescalar los datos). Además, cada fila contiene el atributo "Label" (etiqueta) que indica la letra a la que corresponde la imagen.

Veamos las señas para cada letra, junto con la cantidad de imágenes por letra:

Figura 1

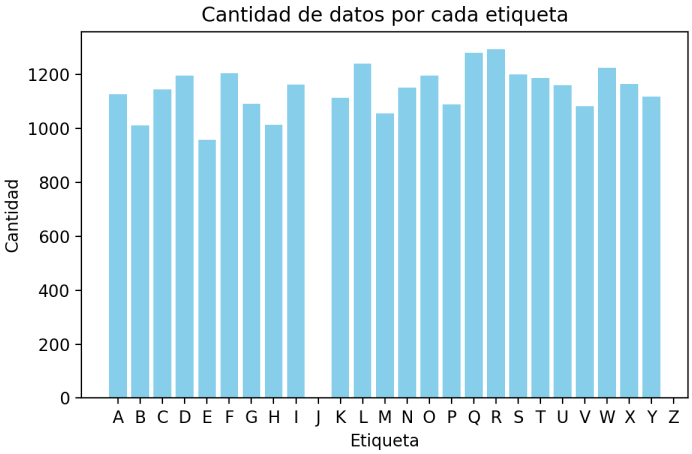

Figura 2

Notar que no aparecen las letras J y Z, dado que las mismas en el lenguaje de señas americano requieren movimiento, pero exceptuando estas, el resto de letras parece tener una cantidad de muestras similar. Esto es importante ya que contar con clases muy desbalanceadas podría introducir sesgos en los modelos, afectando negativamente su capacidad predictiva.

2.2. Diferencias entre imágenes

Por otro lado, nos preguntamos qué tan diferentes son las distintas señas. En la Figura 3, se observa que hay grandes diferencias entre la seña de la letra E y L, sin embargo, la E y la M tienen muchas semejanzas, e incluso un humano podría confundirse una por otra a simple vista. Hay que tenerlo presente a la hora de elegir qué letras queremos intentar clasificar, dado que es esperable que un modelo tenga más problemas con imágenes similares.

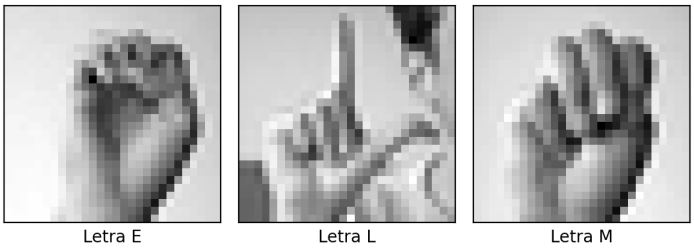

Figura 3: Diferencias entre letras

[1] Link del dataset: https://www.kaggle.com/datasets/datamunge/sign-language-mnist

También es interesante notar cómo cambian las imágenes de una misma clase, como muestra la Figura 4. A pesar de que las manos estén en ángulos y posiciones levemente distintas, no parece haber grandes diferencias. Esto se puede observar mejor en la Figura 5, en donde apilamos todas las imágenes de la letra C y promediamos los valores de los píxeles, y a pesar de ser el resultado de más de mil imágenes, la silueta representa de manera clara la seña correspondiente.

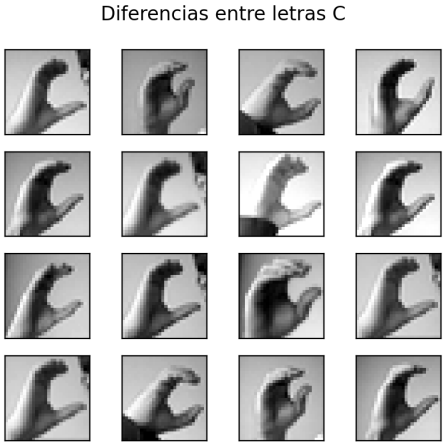

Figura 4

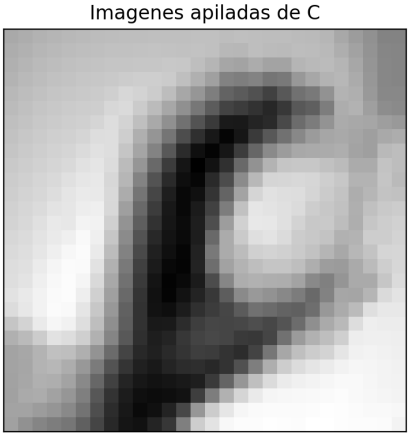

Figura 5

2.3. Atributos más relevantes

Además de mostrar que hay pocas diferencias entre las imágenes de la letra C, la Figura 5 también nos indica que en promedio hay píxeles más relevantes que otros para identificar a la letra. Esto resulta de gran importancia a la hora de modelar, puesto que en lugar de utilizar los 784 píxeles, podríamos reducir la cantidad de atributos para entrenar el modelo.

Para esto, observamos que los píxeles adyacentes tienden a ser similares, por lo que realizamos Max Pooling para mostrar que se puede desechar una gran cantidad de información y todavía la imagen es reconocible.

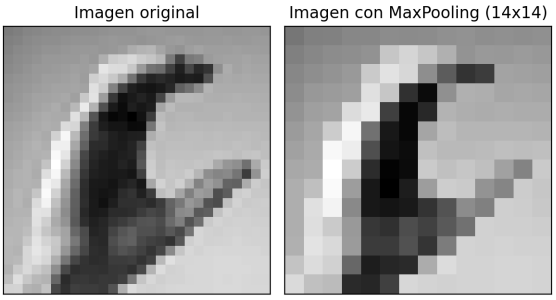

Figura 6

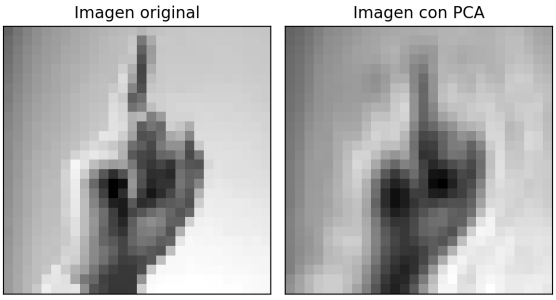

Figura 7

Por otra parte, decidimos hacer reducción de la dimensionalidad con Principal Analysis Components (PCA). De esta manera, encontramos que sólo se necesitan 58 componentes para explicar el 90 % de la varianza de los datos, volviendo a demostrar así que hay mucha información redundante. Además, transformamos las imágenes y las reconstruimos a partir de los componentes principales (se ilustra un ejemplo en la Figura 7), siendo estas fácilmente identificables.

2.4. Comentarios

Por último, es interesante destacar las diferencias que hubo al trabajar con imágenes en lugar de variables categóricas o continuas convencionales. En ciertos aspectos esto introdujo dificultades, dado que hay una gran cantidad de atributos y sus valores no se pueden interpretar a simple vista, por lo que es más complicado intentar relacionar cada atributo particular a la clase correspondiente.

Sin embargo, al ser imágenes, esto ayudó a tener una noción intuitiva de qué atributos podríamos descartar y poder visualizar mejor las transformaciones que hacíamos de los datos, sin perder la capacidad de reconocer la imagen.

3. Clasificación binaria

3.1. Exploración de modelos KNN

En esta sección, construiremos modelos de clasificación con KNN, solamente con un subconjunto de datos constituído por las imágenes de las letras A y L (el cual comprobamos que esté balanceado). El objetivo será maximizar la performance del modelo, mientras minimizamos la cantidad de atributos necesarios para clasificar los datos.

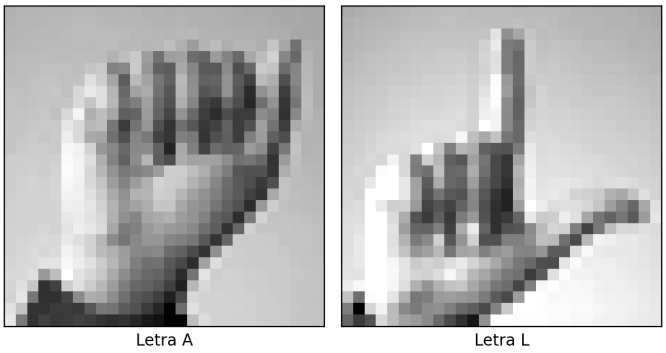

Figura 8: Muestras de A y L

Como se puede observar en la Figura 8, estas dos letras presentan grandes diferencias, por lo que el entrenamiento de los modelos será más fácil.

En primera instancia, separamos entre datos de desarrollo (80 % para el entrenamiento y test/validación) y de evaluación (20 % con los cuales el modelo nunca será entrenado).

Para el primer modelo, comprobamos que si tomamos todos los píxeles y entrenamos un modelo KNN, sin importar la cantidad de vecinos, obtenemos una exactitud del 100 % en los datos de evaluación.

A continuación en la Figura 9, utilizando K-fold Cross-Validation, comparamos la performance del modelo (con 3 vecinos), en función de la cantidad de atributos elegidos al azar, repitiendo el proceso varias veces y promediando los resultados para que estos sean más representativos. De esta manera, encontramos que aproximadamente a partir de los 9 atributos, la exactitud (Accuracy) no mejora significativamente con más atributos. Es destacable que, a pesar de ser aleatorios, sólo con 3 píxeles el modelo tuvo en promedio una exactitud de más del 85 % en los datos de test.

Además, en la Figura 10, evaluamos la exactitud de un modelo KNN con 9 atributos elegidos al azar, pero cambiando la cantidad de vecinos (también con K-fold Cross-Validation y repitiendo el proceso). Sorprendentemente, la performance es peor a medida que aumenta el número de vecinos.

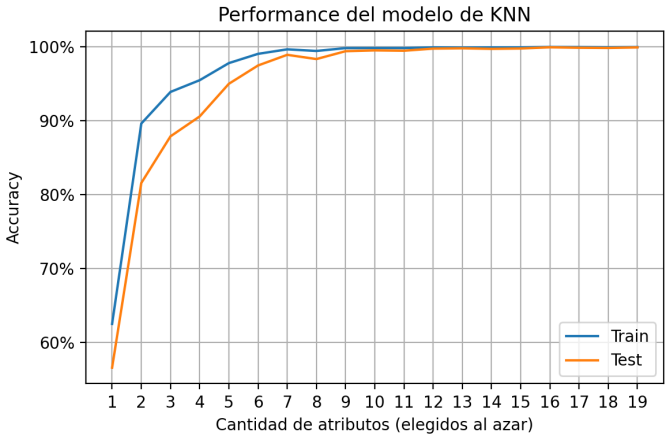

Figura 9

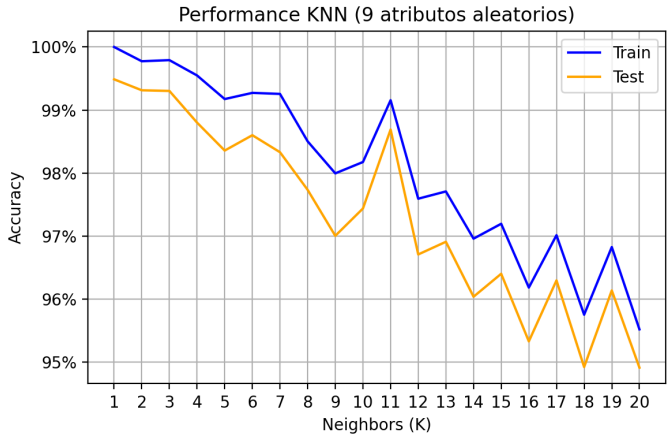

Figura 10

Entonces, a partir de lo visto en el análisis exploratorio y de estos gráficos, deberíamos ser capaces de encontrar un criterio que no sea el azar para seleccionar menos de 9 atributos en el entrenamiento, manteniendo aún así una exactitud cerca del 100 %.

3.2. Primer modelo: KNN con PCA

En primer lugar, probamos utilizando PCA, transformando solamente los datos de desarrollo de las letras A y L. En la Figura 11, notamos cómo la varianza explicada aumenta con la cantidad de componentes principales, alcanzando el 90 % con sólo 25 componentes.

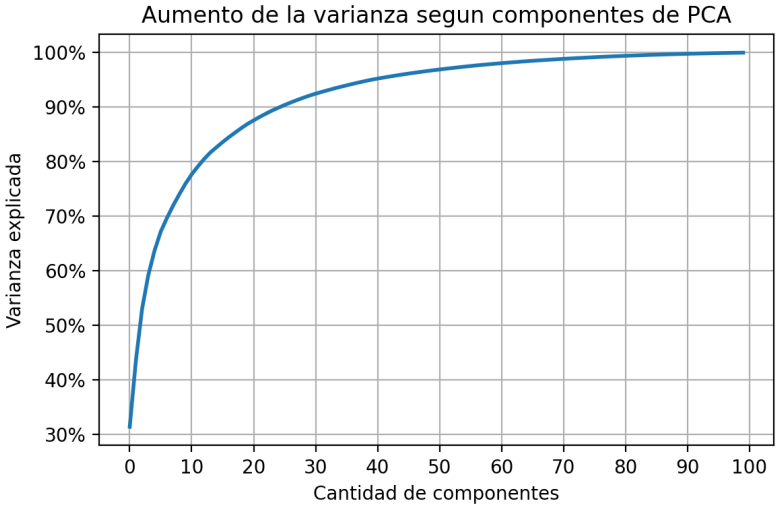

Figura 11

Sin embargo, en este caso no es necesaria una varianza explicada alta para conseguir un porcentaje elevado de exactitud con nuestro modelo. Entrenamos al mismo con solamente 2 componentes y K = 3 (aunque variándolo, pudimos ver que era indistinto), y conseguimos una exactitud del 100 % en los datos de evaluación.

En la siguiente figura, se observan los 2 componentes principales en los ejes y cómo la técnica del PCA separó los datos de manera clara en dos grupos, permitiendo que todos los puntos fueran clasificados correctamente.

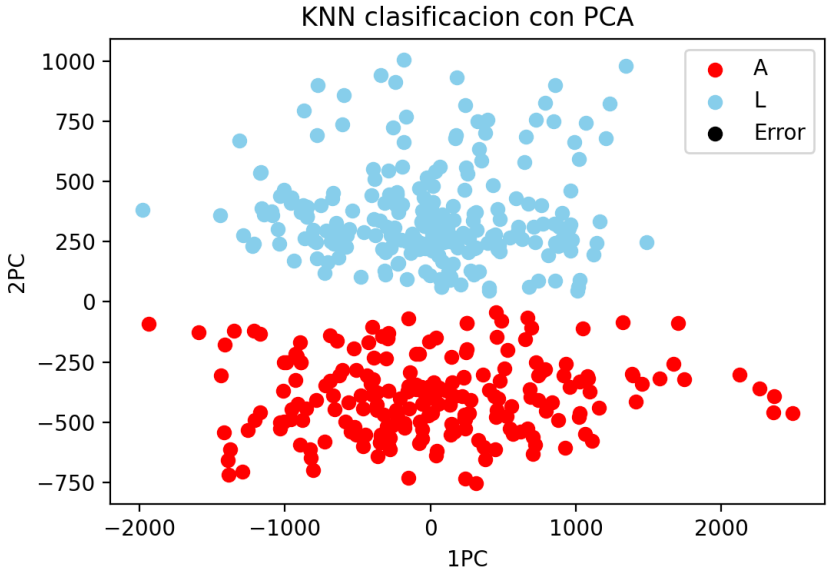

Figura 12

3.3. Segundo modelo: KNN contrastando imágenes

A continuación, apilamos las imágenes de las letras A y L, como hicimos en la Figura 5, pero esta vez restamos los resultados para poder encontrar las zonas de mayor varianza. En la Figura 13, se muestra este resultado, junto con los 2 píxeles con mayor contraste marcados en rojo.

Figura 13

Después, variando la cantidad de atributos de mayor contraste (balanceados entre zonas negras y blancas), aplicamos Grid Search, que utiliza K-fold Cross-Validation para encontrar el mejor valor del hiperparámetro K:

Cantidad de atributos	Mejor K	Accuracy % (cv)
1	3	85,26
2	3	96,14
3	2	99,31
4	1	99,89
5	1	99,95

Figura 14: Grid Search para atributos relevantes

De esta manera, observamos que después de 3 atributos, la performance no aumenta en gran medida. Por lo tanto, elegimos solamente estos 3 atributos para nuestro modelo, y graficamos su exactitud en función de la cantidad de vecinos (Figura 15). Al igual que ocurría con los atributos elegidos al azar, la performance disminuye a medida que aumenta K, y como indicaba la Figura 14, el mejor valor para este hiperparámetro es 2.

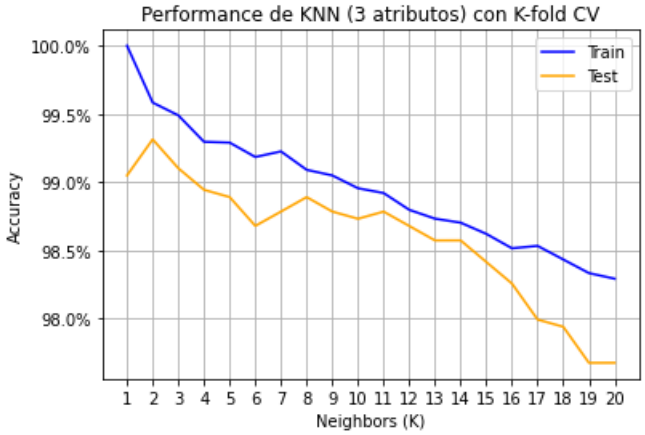

Figura 15

Entonces, con los datos de desarrollo entrenamos un modelo de KNN, con los 3 atributos de mayor contraste y K = 2, y obtuvimos una exactitud del 99,79 % en los datos de evaluación, un resultado considerablemente mejor con respecto a los modelos con atributos aleatorios.

4. Clasificación multiclase

4.1. Exploración de Árboles de Decisión

En este apartado, utilizaremos árboles de decisión para clasificar las vocales, entrenándolos con los datos únicamente que correspondan a estas.

Comenzamos previsualizando las vocales (Figura 16) y analizando si estos datos están balanceados. Como se muestra en la Figura 17, la E cuenta con menos imágenes, por lo que usaremos Stratified K-fold, el cual devuelve el mismo porcentaje de muestras para cada clase.

Figura 16: Previsualizacion de vocales

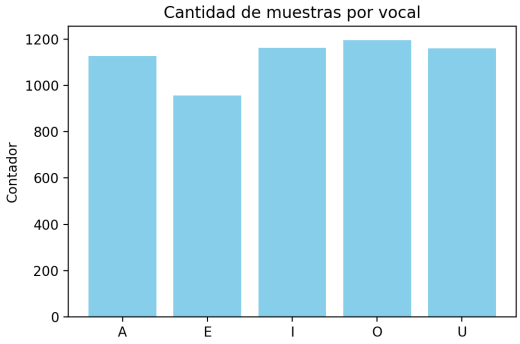

Figura 17

Volvimos a dividir los datos para desarrollo (80 %) y evaluación (20 %).

A continuación, utilizamos Grid Search con Stratified K-fold cross-validation para encontrar el mejor criterio de separación de datos en cada nodo (gini o entropy) y la mejor profundidad del árbol. De esta manera, resultó ser que entropy era el mejor criterio y 12 la mejor profundidad.

Entonces, con entropy como criterio, graficamos la exactitud de los datos en función de la profundidad (Figura 18). A pesar de que Grid Search nos indicaba que 12 era la mejor profundidad, notamos que a partir de 10 ya no hay ninguna mejora significativa de la performance.

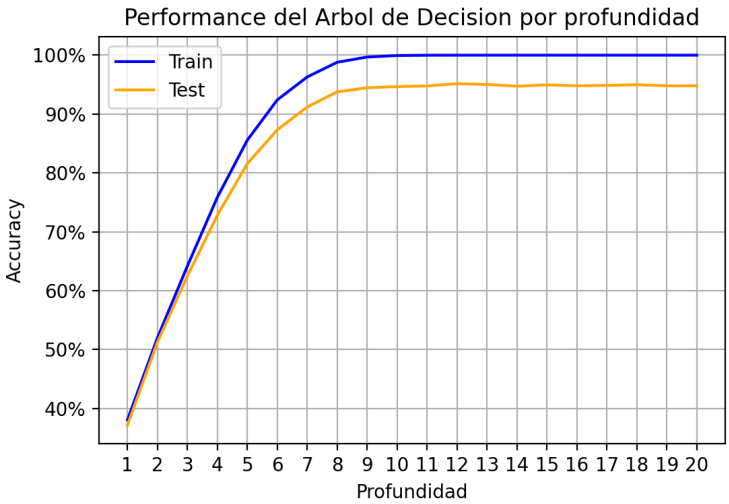

Figura 18

Por lo tanto, decidimos que el mejor modelo podría ser uno con el criterio de entropy y profundidad 10, por lo que lo entrenamos con los datos de desarrollo, y la exactitud en los datos de evaluación fue del 96,34 %.

4.2. Visualizando resultados

Con el modelo mencionado anteriormente, realizamos la matriz de confusión (Figura 19). Podemos ver que las letras con las que más se confundió fueron: la E con la A y la U con la I. Estos resultados son intuitivos ya que a simple vista la A y la E son las más similares, y era esperable que el modelo tenga dificultades con las mismas.

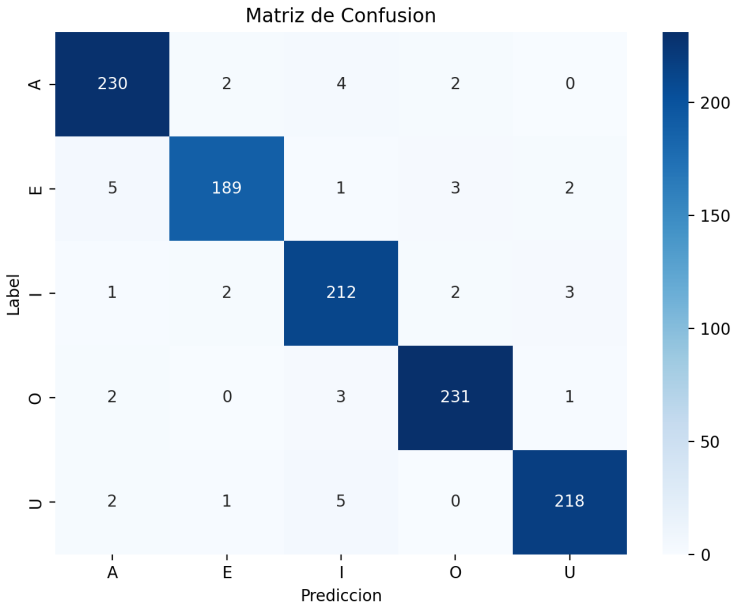

Figura 19

Por último, nos preguntamos qué atributos fueron más importantes para la separación de datos en cada nodo del árbol. Para responder esto, graficamos los 10 píxeles que más contribuyeron a la reducción de la entropía (Figura 20). Como era de imaginar, muchos corresponden a píxeles centrales, y parecen seguir las siluetas de las imágenes.

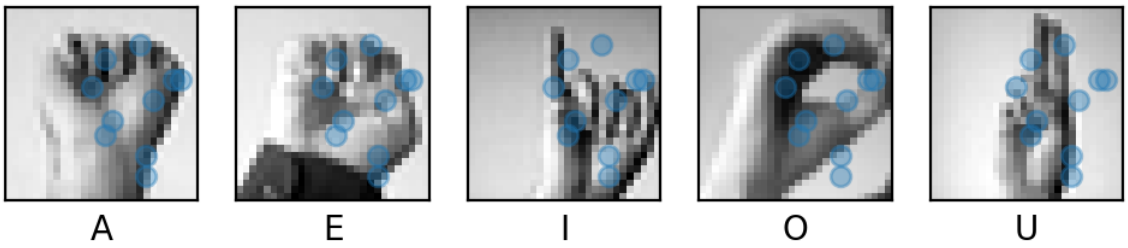

Figura 20: Píxeles más importantes según Árbol de Decisión

5. Conclusiones

Finalmente, podemos concluir que para la clasificación binaria no eran necesarios los 784 atributos para llegar a una exactitud casi perfecta en la evaluación del modelo KNN, dado que con 9 atributos elegidos al azar parecía ser suficiente. Además, el PCA con sólo dos componentes principales no tuvo ningún error en los datos de evaluación, e incluso logramos encontrar los 3 atributos más relevantes en las imágenes (apilando y contrastándolas) que nos permitieron obtener una exactitud de casi el 100 % sin necesidad de transformar los datos previamente.

También cabe destacar que, tanto en los modelos con atributos elegidos al azar como con aquellos entrenados con píxeles relevantes, al variar la cantidad de vecinos no se apreciaba un gran cambio en la performance, e incluso empeoraba levemente a medida que aumentaba este hiperparámetro.

Por otra parte, en la clasificación multiclase, como era esperable, la performance era mayor a medida que se incrementaba la profundidad de los árboles de decisión. También pudimos observar que la exactitud del modelo final de Árbol de Decisión (96,34 %) fue levemente menor que el de clasificación binaria de KNN (100 % con PCA y 99,79 % con pixeles relevantes), princpalmente porque debía diferenciar entre 5 clases en lugar de 2.