Trabajo 4: Clasificación de dígitos MNIST

Técnicas en aprendizaje estadístico

Elaborado por:

Mateo Zuluaga Giraldo
Santiago Restrepo Álvarez
Daniel Restrepo Mejía
Sebastián Ramírez Tamayo
David Ossa

Universidad Nacional de Colombia – Sede Medellín

2018-2

CLASIFICACIÓN DE DÍGITOS

PLANTEAMIENTO DEL PROBLEMA

Uno de los principales problemas de visión por computador es el problema de la clasificación de imágenes, que se ocupa de determinar la presencia de estructuras visuales en una imagen de entrada. Más precisamente, se nos da un set de imágenes de entrada: $X = \{X\square_1, ..., X_n\}\square$ y una matriz de etiquetas $Y' = [y'_1, ..., y'_n]$, donde cada $y'_i \epsilon B^k$ para todos los $i = 1,2,..,n$, y k denota el número de clases. El objetivo es generar un *set* de predicciones $Y^{\epsilon} R_{nxk}$ basados en el set de X para que se parezcan a Y tanto como sea posible[3]. Para analizar este problema se utilizará MNIST.

MNIST posee una base de datos donde se encuentran matrices que consisten en imágenes de 28x28 pixeles, donde cada *pixel* representa la coloración en una escala de grises y cada matriz por sí misma representa un de los diez dígitos escrito a mano:

Figura 1. Ejemplo de números escrito a mano de MNIST[1]

Cada entrada de la matriz representa cuán oscuro es cada *píxel*, por ejemplo se tiene a continuación el número 1 escrito a mano y su respectiva matriz de *píxeles*:

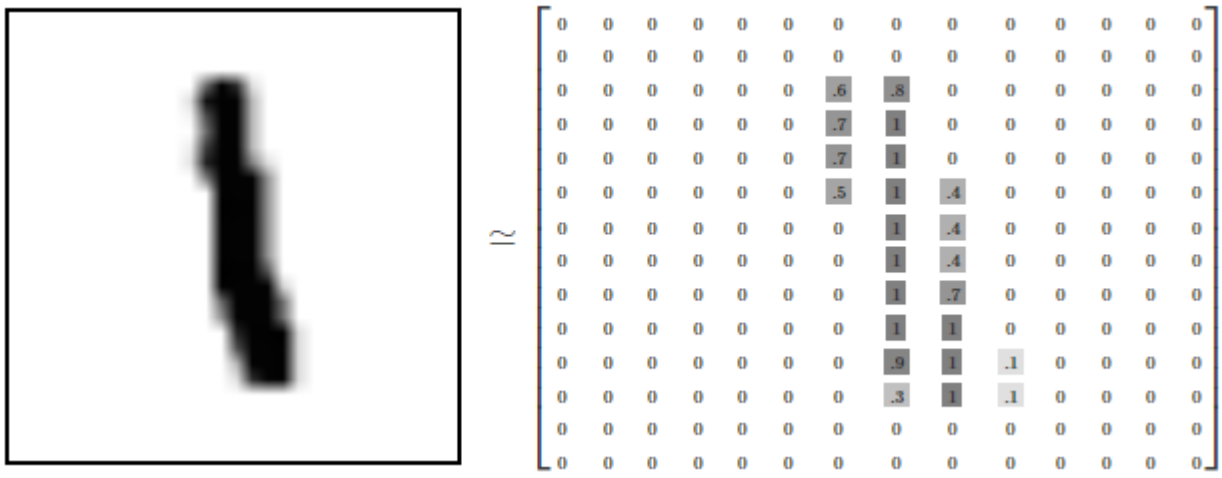


Figura 2. Representación matricial del número 1 escrito a mano. [1]

El primer objetivo de este análisis, es construir cinco clasificadores multiclase diferentes utilizando las siguientes técnicas para resolver el problema de la clasificación de dígitos escritos a mano [2]:

1. Regresión logística multinomial
2. Árboles de clasificación
3. Bosques aleatorios
4. Máquinas de soporte vectorial
5. Redes neuronales

Y luego definir cuál de las anteriores técnicas es la más adecuada y bajo qué configuración, ya que cada técnica tiene definidos ciertos parámetros que permiten encontrar una mejor solución dependiendo a qué características tiene el problema.

ESTRATEGIA METODOLÓGICA:

Se hará un análisis descriptivo de los datos de entrenamiento para observar cómo es su estructura y organización con respecto a las filas y columnas del dataset (etiquetas y pixeles) y se obtendrá información que nos brinde más entendimiento del dataset para así de esa forma trabajar con él.

Para mejorar el rendimiento de los clasificadores y que los algoritmos sean más rápidos, se normalizan los datos, convirtiendo los valores en 0 para los pixeles blancos y 1 para los negros. La Figura 3 nos muestra cómo se distribuye la cantidad de dígitos que aparecen en la base de datos, se puede observar que los datos están aproximadamente distribuidos de manera uniforme lo cual nos permite asegurar que el *dataset* tiene un buen balance de dígitos para entrenar los modelos.

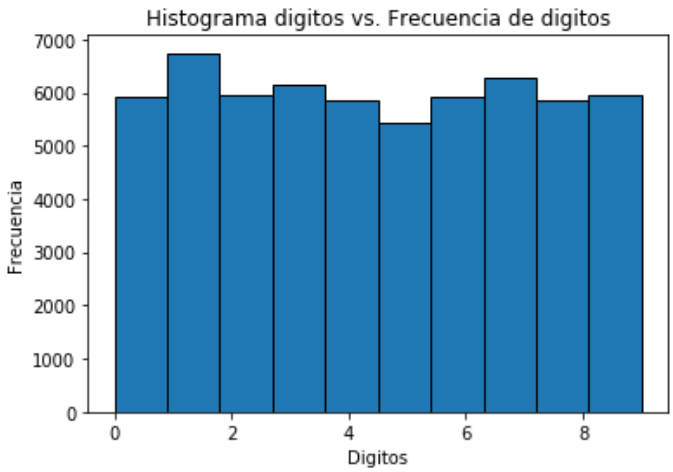


Figura 3. Distribución de los dígitos en la base de datos

Después de este breve análisis descriptivo se procede a implementar cada modelo.

Regresión logística multinomial

Para este modelo se usa la librería *Sklearn* [4] para usar este modelo se pueden usar varios algoritmos; "*newton-cg*", "*sag*", "*saga*" y "*lbfgs*" que son algoritmos que funcionan bien para problemas multinomiales como es nuestro caso. Se descarta "*liblinear*" por ser un algoritmo que funciona bien para *datasets* pequeños, contrario al *dataset* de MNIST. Se entrenan cuatro modelos con los algoritmos mencionados anteriormente para determinar cuál es el mejor de todos y el resultado obtenido es casi igual para todos los modelos, ganando ***lbfgs*** por orden de diferencia de 0.001 con los demás algoritmos.

Validación del modelo con *mnist_test*: Luego de hacer las predicciones pertinentes con los datos de validación, se obtuvo una precisión del 93%.

Árboles de clasificación

Se usa la misma librería anterior y en este caso el parámetro que vamos a variar en el Árbol de Clasificación es "*min_samples_split*", que nos indica la cantidad de muestras mínimas necesarias para dividir un nodo, esta variable se alternará en valores de 10% a 100% con un aumento del 10% entre valores. Se halla que la mayor precisión del modelo se encuentra dividiendo el árbol en muestras del 10%.

Validación del modelo con *mnist_test*: Luego de hacer las predicciones pertinentes con los datos de validación, se obtuvo una precisión del 62.1%.

Bosques aleatorios

Se usa la misma librería. El parámetro que podemos variar es el número de estimadores para el clasificador (*n_estimators*), según la documentación es la cantidad de árboles que se usará para el clasificador.

Se puede observar que después de 32 estimadores el ratio de error y la precisión del modelo es bastante similar a medida que se aumenta la cantidad de estimadores, por esta razón, la mejor cantidad de estimadores de los cuales tuvimos en cuenta para entrenar y predecir el modelo son 32 estimadores.

Validación del modelo con *mnist_test*: Luego de hacer las predicciones pertinentes con los datos de validación, se obtuvo una precisión del 96.51%.

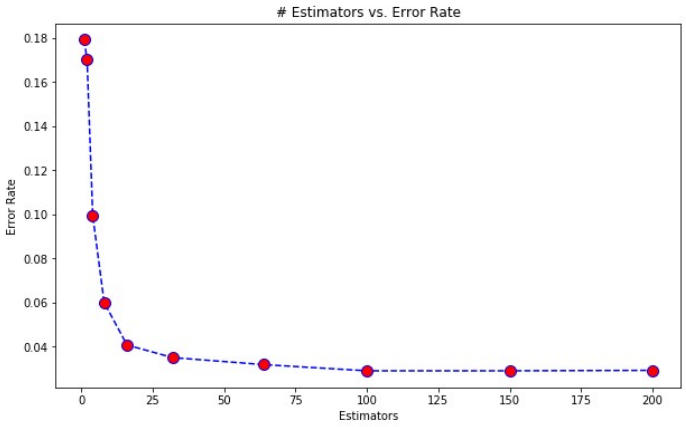


Figura 4. Variación del error según número de estimadores

Máquinas de soporte vectorial

Para máquinas de soporte vectorial usaremos una herramienta de *Sklearn* llamada *GridSearchCV* que nos ayudará a buscar los mejores parámetros a usar en nuestro modelo combinando diferentes valores para parámetros como la penalización por error (c) y el *gamma* del *kernel* (*gamma*) usando validación cruzada.
Este modelo recibe:

- El modelo sobre el que va a iterar, en este caso, sobre *SVC*() que corresponde al modelo de Máquinas de Soporte Vectorial en la librería de *Sklearn*.
- El diccionario con los posibles valores (*param_grid*).
- *Refit* lo usamos para poder obtener el atributo best*params* que nos indicará cuál fue la mejor combinación de parámetros encontrada.
- *n_jobs* que indica la cantidad de procesadores a usar para el entrenamiento.

El mejor modelo se obtuvo con un C = 100 y *gamma* = 0.01 bajo el *kernel rbf* que es el predeterminado en el modelo de SVM.

Validación del modelo con *mnist_test*: Luego de hacer las predicciones pertinentes con los datos de validación, se obtuvo una precisión 98%.

Redes neuronales.

Para construir este modelo, se usa "*Tensorflow*" y "*Keras*" que son librerías muy útiles para trabajar con redes para construir este modelo se usó el artículo de referencia [6].

Se debe hacer un *hot encoding* para que la red neuronal no tome los dígitos de forma numérica sino categórica. Se define una variable "*image_size*" con un valor de 784 que serán la cantidad de neuronas de nuestra capa de entrada.

Nuestra capa de salida tendrá una cantidad de neuronas definida en la variable *num_clases*=10 que corresponde a cada uno de los números del 0 al 9 de la variable categórica del *dataset* de MNIST.

Vamos a construir un perceptrón multicapa con una sola capa oculta variando el parámetro *units* en la capa de entrada, este parámetro indica la cantidad de nodos en una capa de la red neuronal. A continuación, veremos los resultados:

Figura 5. Modelo con 32 nodos, precisión 96.3%

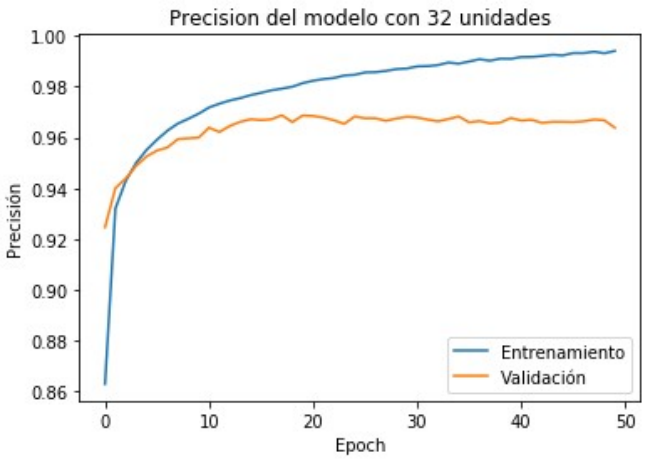


Figura 6. Modelo con 256 nodos, precisión 98.3%

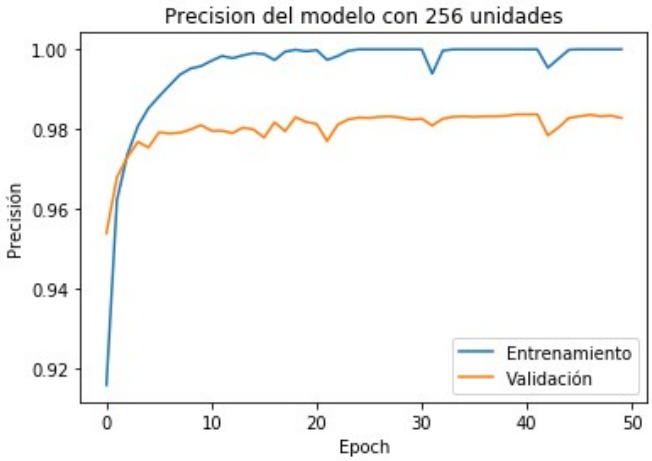


Figura 7. Modelo con 784 nodos, precisión 98.6%

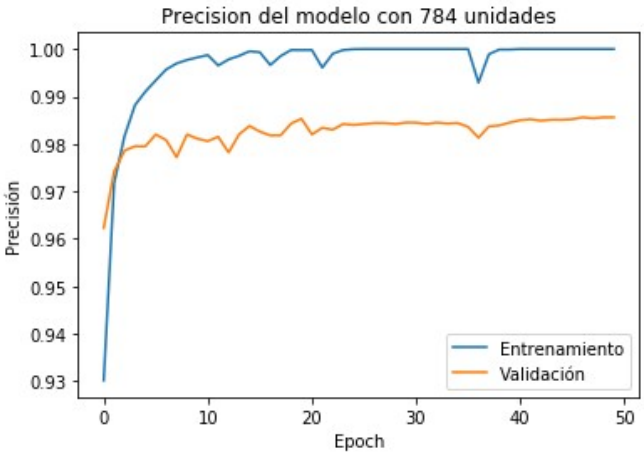
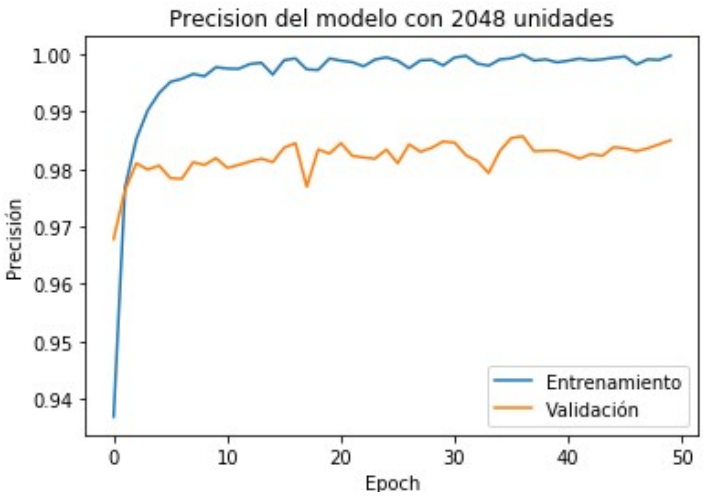


Figura 8. Modelo con 2048 nodos, precisión 98.5%



Validación del modelo con *mnist_test*: Se puede observar que los modelos mantuvieron una precisión entre 96% y 99% en el conjunto de validación sin importar la cantidad de nodos. Algo a notar es que a medida que se fueron aumentando la cantidad de nodos, el tiempo que tardó el *dataset* de entrenamiento en llegar a un punto óptimo de precisión fue menor.

Comparación entre modelos.

Teniendo en cuenta que cada uno de los modelos tiene sus pros y sus contras, se debe analizar cuál de todos los modelos será el mejor teniendo en cuenta su grado de precisión y algunos de estos factores.

Para comenzar, se debe tener en cuenta que en este caso no se presentan variables dependientes de manera directa, ya que lo que se tiene es una matriz de 28x28 para cada uno de los datos del *MNIST*; por lo tanto, los modelos que permiten interpretar de una manera sencilla la dependencia de las variables como la regresión o los árboles de decisión. De igual manera se debe tener en cuenta que las redes neuronales, los bosques aleatorios y las máquinas de soporte vectorial no permiten saber de una manera sencilla esta dependencia, pero al no ser este el caso de estudio, todos los modelos están en igualdad de condiciones.

Otro factor a tener en cuenta, es el requisito computacional a la hora de entrenar los modelos. Las redes neuronales son las que más le exigen a la máquina y las regresiones las que menos le exigen a esta por su sencillez. Pero cómo sólo nos interesa tener éxito en las predicciones, si el modelo ya está definido y correctamente entrenado, no debería ser un factor a tener en cuenta.

Dado todo lo anterior, todo parece indicar que el mejor parámetro a tener en cuenta es la precisión a la hora de predecir. En este caso se procede a usar como mayor punto para la escogencia de un modelo a la precisión de los modelos a la hora de predecir los datos de validación:

Modelo	**Precisión en datos de validación**
Regresión multinomial logística	93%
Árboles de clasificación	62.1%
Bosques aleatorios	96.5%
Máquina de soporte vectorial	98%
Redes neuronales	99%

En este caso el mejor modelo para esto son las redes neuronales ya que obtuvieron una precisión del 99% en los datos de validación

Se recomienda altamente ver en el repositorio de GitHub (https://github.com/Daniel2894/MNIST-TAE-classifiers) el archivo “Trabajo4.ipynb” ya que ahí se explican detalladamente los pasos y supuestos hechos para realizar los cinco modelos.

Aplicación en la vida real

Una muy útil aplicación a los modelos de lectura de dígitos podría ser la clasificación de cartas leyendo los códigos postales, para esta arquitectura muy básica que se explica detalladamente a continuación:

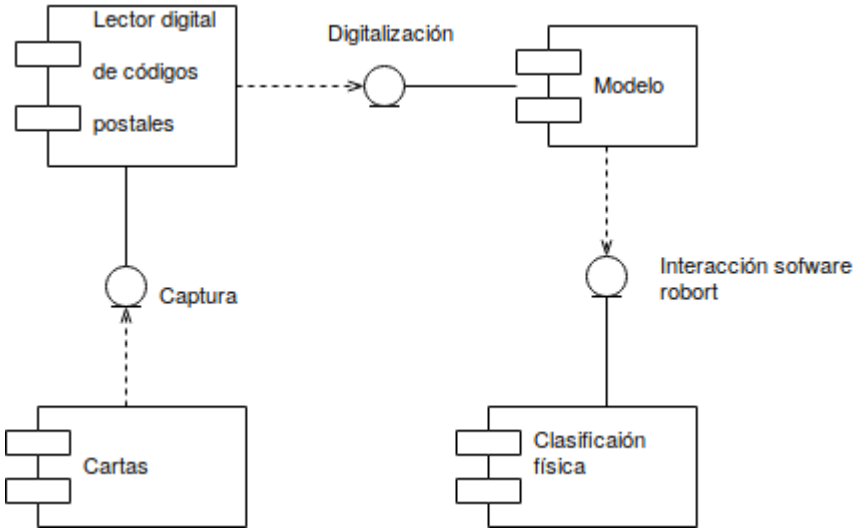


Figura 9. Diagrama de componentes UML

Cartas: se entienden como el conjunto físico de cartas, organizadas de alguna forma específica para que algún robot pueda tomarlas automáticamente para la posterior clasificación.
Captura: la captura se realiza cuándo del proceso anterior se recibe la carta y se fotografía el campo donde está escrito el código postal.
Lector de códigos postales: la función de este lector es fotografiar el campo donde está el código postal.
Digitalización: en este paso se asocia la imagen con la respectiva carta mediante su posición física en el sistema.
Modelo: este es el paso más importante donde se aplica el modelo para obtener los números escritos, se divide la imagen en n sub imágenes que cada una contenga un dígito y se aplica el modelo, este sería el código postal que se asocia nuevamente con la posición física de la carta en el sistema.
Interacción software robot: en este paso se supone que debe haber un robot que reciba el código postal de la carta que está en la fila y realiza la acción correspondiente.
Clasificación física: en este paso el robot clasifica la carta en casilleros o como se requiera.

RESULTADOS Y CONCLUSIONES

Juntando los resultados que se obtuvieron al analizar las diferentes técnicas con diferentes configuraciones de parámetros se tiene lo siguiente:

- La regresión logística multinomial tuvo una precisión del 93%, en los árboles de decisión la mayor precisión fue del 62%, en los bosques aleatorios se tuvo una precisión del 97%, con las máquinas de soporte vectorial se obtuvo 98% y finalmente con las redes neuronales se llegó a la precisión del 97%. Concluyendo que de esta manera con las combinaciones de los parámetros utilizados, la mejor técnica fue la de soporte de máquinas vectoriales, pero con muy poca diferencia con el resultado de las redes neuronales.
- En la técnica de Regresión Logística Multinomial el algoritmo de solución con mejor resultado fue el de 'lbfgs' debido a que es diseñado para problemas de multiclase y maneja la pérdida multinomial.
- En la técnica de Árboles de decisión el parámetro a variar fue el de la cantidad de muestras mínimas necesarias para dividir un nodo y con respecto a los resultados se concluye que entre menor es este parámetro, mayor precisión tendrá el modelo.
- En la técnica de Bosques aleatorios, cuando la cantidad de árboles que usa el clasificador es mayor, la precisión del modelo también lo es, pero se ve reducido el rendimiento del tiempo de entrenamiento del modelo drásticamente.
- En la técnica de redes neuronales se puede concluir que la cantidad de nodos en una capa de la red neuronal no influye demasiado en la precisión de la red neuronal ya que variando este valor siempre se obtuvo una precisión entre el 96 y 97 por ciento.
- El uso de GridSearch puede ser muy útil para encontrar fácilmente los parámetros óptimos de un modelo, sin embargo, implica un alto costo computacional lo que puede volver más lento el proceso de entrenamiento de los modelos, como fue el caso de la técnica de máquinas de soporte vectorial.
- Un alto número de estimadores para bosques aleatorios no asegura un mejor proceso de entrenamiento, teniendo en cuenta que a mayor número de estimadores no hay una mejora significativa en la precisión del modelo y el entrenamiento se vuelve más costoso computacionalmente.
- A medida que se fueron aumentando la cantidad de nodos en las redes neuronales, el tiempo que tardó el dataset de entrenamiento en llegar a un punto óptimo de precisión fue menor, por lo que se concluye que a mayor número de nodos, habrá más velocidad para que el modelo llegue a su punto más alto de precisión.

BIBLIOGRAFIA

Repositorio: https://github.com/Daniel2894/MNIST-TAE-classifiers

[1] http://colah.github.io/posts/2014-10-Visualizing-MNIST/

[2] https://docs.aws.amazon.com/es_es/machine-learning/latest/dg/multiclass-classification.html

[3] http://www1.eafit.edu.co/asr/courses/research-practises-me/2016-1/students/proposal-reports/c_proposal_20161_gallegoposada_montoyazapata.pdf

[4] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[5] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[6] https://www.analyticsvidhya.com/blog/2017/06/getting-started-with-deep-learning-using-keras-in-r/