**Paltatrón**

Introducción

La red neuronal aquí presentada tiene como objetivo predecir si una palta está madura o no. Se escogió una base de datos sintética[[1]](#footnote-1) que contempla distintas variables y clasifica la madurez de la palta en cinco categorías de madurez. Nuestro trabajo redujo estas cinco categorías a solo dos (madura o no), dado que se trata de una red neuronal de clasificación binaria.

Parte 1. Análisis de la base de datos

**1. Descripción de las columnas de la base de datos**

La base de datos elegida contempla una serie de variables de las paltas que se resumen en este cuadro.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Variable** | **Unidad y rango (si hubiere)** | **Tipo** | **Categorización** | **Aclaraciones metodológicas** |
| Peso | Gramos (g) | Continua | - Dura: 200-300 g  - Preacondicionada: 180-280 g  - En transición: 160-260 g  - Madura-firme: 150-240 g  - Madura: 150-220 g | Estas categorías se usan luego para computar la densidad (peso/tamaño), cuya categorización es:  - Dura: 1,4-1,8  - Preacondicionada: 1,2-1,6  - En transición: 1,0-1,4  - Madura-firme: 0,8-1,2  - Madura: 0,6-1,0 |
| Tamaño | Centímetros cúbicos (cm³) | Continua | - Dura: 100-200 cm³  - Preacondicionada: 120-220 cm³  - En transición: 140-240 cm³  - Madura-firme: 160-260 cm³  - Madura: 180-300 cm³ |
| Firmeza | Newtons (N) | Continua | - Dura: 80-100 N  - Madura: 10-20 N | Es la fuerza que hay que hacer para atravesar la corteza. |
| Sonido | Decibeles (dB) | Continua | - Dura: 70-80 dB  - Madura: 30-40 dB | Es el sonido que hace la palta al ser golpeada con una fuerza determinada. |
| Tono | Grados (0-360°) | Discreta | - Verde oscuro: 60-120°  - Verde: 45-90°  - Violeta: 270-330°  - Negro: 0-30° | Estas tres categorías conforman el color. Fueron transformadas desde el patrón RGB. |
| Saturación | Porcentaje (0-100 %) | Discreta | - Verde oscuro: 70-100 %  - Verde: 60-90 %  - Violeta: 50-80 %  - Negro: 30-60 % |
| Brillo | Porcentaje (0-100 %) | Discreta | - Verde oscuro: 40-70 %  - Verde: 50-80 %  - Violeta: 30-60 %  - Negro: 10-40 % |
| Color |  | Categórica | Verde oscuro, verde, violeta, negro |  |
| Madurez |  | Categórica | Dura, preacondicionada, en transición, madura-firme, madura |  |

**2. Análisis de correlaciones**

Todas las correlaciones entre las variables son negativas, excepto tamaño y madurez que es positiva. En módulo, las mayores correlaciones con la madurez son las de firmeza, sonido y tamaño, dado que tienen más de 0,8.

Tomamos las siguientes variables:

- Firmeza: alta correlación negativa.

- Tono, saturación y brillo: las tres juntas conforman el color. Por sí solas no tienen valor para nuestro análisis (dado que no representan nada en el mundo real), pero juntas sí.

- Sonido en decibeles: es la variable que tiene la correlación más alta.

- Densidad: tomamos esta variable (calculada como peso/tamaño) y no las variables de peso y tamaño (a pesar de que estas últimas por sí mismas tengan más correlación), porque la densidad permite comparar paltas de distinto peso y tamaño.

Nos llamó la atención la alta correlación negativa entre el sonido (medido en decibeles a través de un decibelímetro) y la madurez de la palta; a diferencia de la firmeza, variable con una correlación muy similar, pero que es de "sentido común" que se vincula a la madurez (a mayor firmeza, menor madurez y viceversa).

Ver gráfico 1 (“Correlaciones entre las variables”).

**3. Análisis de factibilidad**

La base de datos es adecuada para entrenar una red neuronal de clasificación porque:

- Hay una variable *target* (a predecir).

- Hay variables *features* (predictores) que tienen correlación con la variable *target*.

- No contiene valores atípicos.

Entrenaremos la red para que pueda predecir si una palta se puede considerar lista para comer o no según su grado de madurez.

**4. Datos atípicos y limpieza de datos**

Se hizo un análisis de datos atípicos con dos métodos: rango intercuartílico y desvío estándar. Según el primer método, no hay valores menores que la diferencia entre el rango y el primer cuartil ni mayores que la suma entre el rango y el tercer cuartil. Por lo tanto, no hay valores atípicos en ninguna de las variables numéricas.

Por otro lado, para el segundo método, se calculó el desvío estándar para cada columna de las variables numéricas y se analizó qué valores estaban a más de tres desvíos estándares según la media; y se concluyó que ningún dato cumplía con esa condición, por lo que, según este otro método, tampoco tenemos valores atípicos en la base de datos.

En cuanto a la limpieza, implicó recategorizar la variable de salida (madurez): de cinco categorías pasamos a dos, es decir, dicotomizamos la variable y la guardamos como 0 o 1 para trabajar con la red.

**5. Transformaciones preliminares**

- Se computó la densidad (peso/tamaño).

- Se binarizó la variable de madurez (de cinco categorías pasamos a dos).

- Se analizó si había valores atípicos con rango intercuartílico y con desvío estándar, y no hay.

- Normalización: usamos dos métodos (dividir por el máximo o el del desvío estándar). Generamos dos datasets distintos con cada uno de los métodos y los preservamos durante el desarrollo de la red para comparar ambos resultados, como se desarrolla en el notebook.

- División prueba/entrenamiento: se separó 1/3 de los datos para la prueba y 2/3 para el entrenamiento. Se hizo con la misma semilla.

Todas estas transformaciones son necesarias para ajustar valores “crudos” a las necesidades de nuestro modelado de la red. No podríamos hacer un modelo que predijera bien las relaciones entre magnitudes y escalas tan diferentes si no las hiciéramos.

Parte 2. Desarrollo de la red neuronal

**1. Arquitectura de la red**

Diagrama

El contenido generado por IA puede ser incorrecto.

Hay dos capas (una oculta y otra de salida), con seis neuronas en la capa oculta. La capa oculta se activa con una función ReLU (que transforma los valores negativos en 0 y deja iguales los positivos) y la capa de salida se activa con una función logística (que “comprime” todos los valores en un rango entre 0 y 1).

De cada variable de entrada sale una flecha que simboliza un peso *w* (no se dibujó en el esquema por ser muy confuso). Al ser seis variables de entrada por seis neuronas, los pesos en total son 36 (para la entrada de la capa oculta), donde la neurona A recibe los pesos *w1* a *w6* (uno por cada variable de entrada), la neurona B recibe los pesos *w7* a *w12*, y así sucesivamente.

Asimismo, cada neurona tiene un sesgo *b*.

De esta forma, estas son las fórmulas para cada neurona de la capa oculta:

A = x1​⋅w1​+x2​⋅w2​+x3​⋅w3​+x4​⋅w4​+x5​⋅w5​+x6​⋅w6​+b1

B = x1​⋅w7​+x2​⋅w8​+x3​⋅w9​+x4​⋅w10​+x5​⋅w11​+x6​⋅w12​+b2

C = x1​⋅w13​+x2​⋅w14​+x3​⋅w15​+x4​⋅w16​+x5​⋅w17​+x6​⋅w18​+b3

D = x1​⋅w19​+x2​⋅w20​+x3​⋅w21​+x4​⋅w22​+x5​⋅w23​+x6​⋅w24​+b4

E = x1​⋅w25​+x2​⋅w26​+x3​⋅w27​+x4​⋅w28​+x5​⋅w29​+x6​⋅w30​+b5

F = x1​⋅w31​+x2​⋅w32​+x3​⋅w33​+x4​⋅w34​+x5​⋅w35​+x6​⋅w36​+b6

En cuanto a la capa de salida, su fórmula sería:

G = w37​⋅A + w38⋅B + w39⋅C + w40⋅D + w41⋅E + w42⋅F + b7

Una vez que activamos las neuronas[[2]](#footnote-2), sus fórmulas serían:

A\_activada = ReLU(A)

B\_activada = ReLU(B)

C\_activada = ReLU(C)

D\_activada = ReLU(D)

E\_activada = ReLU(E)

F\_activada = ReLU(F)

G\_activada = logistic(G)

Las fórmulas de las neuronas sin activar conforman una matriz Z1 y Z2, mientras que las neuronas activadas integran la matriz A1 y A2, respectivamente. La forma matricial es útil para simplificar el cómputo de los pesos y sesgos.

**2. Implementación en numpy**

Esta parte está desarrollada en el archivo .ipynb, pero haremos una serie de aclaraciones generales:

- Los pesos y sesgos fueron inicializados de manera aleatoria.

- Los cálculos fueron hechos de forma matricial.

- Declaramos una clase AvocadoPerceptron que se inicializa con:

\* Cantidad de variables de entrada (6)

\* Cantidad de neuronas de la capa oculta (6)

\* Cantidad de neuronas de la capa de salida (1)

\* Semilla (parámetro opcional, esta vez inicializado en 42)

A su vez, la clase tiene los siguientes métodos (cada uno documentado en el notebook).

- \_propagar\_adelante(self, vector\_entrada)

- predecir(self, vector\_entrada, umbral)

- getPesos(self)

- reajustar\_pesos(self, pesos)

- entrenar(self, vector\_entrada, Y)

Este último método es el algoritmo de la retropropagación, que básicamente consiste en tomar una predicción (producto del método predecir()), tomar el valor real de la base de datos (parámetro Y) y comparar el costo mediante la función de costo[[3]](#footnote-3). En nuestro caso, la predicción es la salida activada A2. Debemos ir derivando la función de costo de manera parcial en función de los pesos y sesgos *w\_i* y *b\_i*. Por lo tanto, tendremos que partir de A2 e ir derivando hacia los pesos y sesgos de Z2, luego de A1 y, por último, de Z1. El detalle de los cálculos está implementado y documentado en el notebook.

Hay muchos algoritmos para minimizar la función de costo. En nuestro caso, utilizamos el descenso de gradiente estocástico, que parte de tomar una tasa de aprendizaje y disminuir el valor de los pesos y sesgos multiplicando esa tasa por el valor de la última derivada.

**3. Entrenamiento y evaluación**

La red fue entrenada con el método entrenar(), que aplica la retropropagación.

Las curvas de precisión están graficadas en el notebook. Hicimos una prueba para ver en qué momento el aprendizaje se estancaba (para evitar el sobre-entrenamiento u *overfitting*) y comparamos la precisión de la fase de entrenamiento con la de la fase de prueba. Logramos obtener una precisión de 1 (es decir, el máximo posible) para la parte de prueba utilizando el dataset normalizado por desvío estándar. Con el otro método de normalización (división por el máximo valor) logramos buenos resultados, pero no óptimos, dado la precisión fue un valor cercano a 1, pero menor.

También quisimos ver si un umbral de decisión más permisivo o más estricto de la activación de la neurona de salida mejoraba los resultados. Es decir, probamos con nueve umbrales distintos (detallado en el notebook) y comparamos la precisión con cada uno de ellos.

Después de obtener esos resultados quisimos ver si podíamos llegar a niveles similares reduciendo la cantidad de variables de entrada. Para ello, creamos redes alternativas basadas en diferentes variables (detallado en el notebook).

En cuanto a las iteraciones, al principio probamos con 100 000 y, luego de entrenar la red, vimos que era un número excesivo. Sin embargo, dada la naturaleza del dataset, la precisión de salida no empeoró, pero sí se estancó la mejora de la precisión de entrenamiento.

**4. Análisis de overfitting**

Debido a que la precisión de prueba no disminuye a mayor entrenamiento, podemos concluir que el modelo no se está sobreajustando a los datos de entrada. Los motivos de esto último los ensayamos en la conclusión.

Parte 3. Comparación con scikit-learn

Instanciamos un MLPClassifier poniéndole un máximo de iteraciones igual al que usamos nosotros (40 000), utilizando el mismo método de descenso de gradiente estocástico, con las mismas variables de entrada. Para la parte de prueba, también llegó a una precisión de 1.

Creemos que un resultado de test tan alto de nuestra red neuronal y la necesidad de tocar el umbral de decisión de la función logística son producto de haber tomado un dataset sintético, de pocos registros y modelado con alta correlación.

Parte 4. Conclusión

Logramos construir un modelo que, a partir de un conjunto reducido de variables (el sonido y la firmeza), puede estimar con una alta precisión si una palta está madura o no. Construir esto implicó un proceso iterativo de comparación, análisis de métricas y *feature engineering* que, basado en evidencia, pudo ser optimizado hasta lograr los resultados deseados.

En el proceso de entrenar la red “manualmente” nos fuimos dando cuenta de que, contrariamente a lo que habíamos supuesto en un principio, nuestras variables eran muchas (no pocas). Más aún, hay variables de entrada cuya alta correlación con la variable de salida es bastante intuitiva (la firmeza en relación con la madurez); pero hay otras que no lo eran para nosotros (como el sonido, que dio una precisión incluso mayor que la firmeza).

1. <https://www.kaggle.com/datasets/amldvvs/avocado-ripeness-classification-dataset> [↑](#footnote-ref-1)
2. Recordemos cómo son las fórmulas de esas funciones:

   ReLU: f(x) = max(0, x)

   logistic: f(x) = [↑](#footnote-ref-2)
3. Recordemos que la función de costo es:

   C = (y\_predicha – y\_real)^2 [↑](#footnote-ref-3)