Practica 2
Clasificación de frijoles mediante tecnicas de machine learning

1st Erick Franco Gaona
Departamento de Estudios Multidisciplinarios
Universidad de Guanajuato
Yuriria, México
e.francogaona@ugto.mx

***Resumen*—La clasificación es muy utilizada hoy en dia en diferentes campos de la vida cotidiana. Es todo un reto recolectar información suficiente para poder clasificar elementos ya que es necesario contar con un gran número de datos. Debido a esto, se utilizan datasets públicos para ahorrar tiempo de adquisición de datos, aunque no existen bases de datos para todo tipo de casos de estudio. En este trabajo se utilizan 6 tecnicas de clasificación mediante machine learning y se comparan los resultados en una dataset público para clasificar frijoles. Las técnicas utilizadas fueron clasificador bayesiano simple, K vecinos más cercanos, árboles de desición, bosques aleatorios, máquina de vectores de soporte y redes neuronales con un 89.68 %, 92.69 %, 90.89 %, 92.03 %, 93.46 % y 93.88 % de exactitud respectivamente.**

I. Introducción

La clasificación consta en ordenar u organizar las cosas en un conjunto de categorías o clases. Puedes categorizar ideas, objetos o cualquier tipo de referencia. El concepto de clasificación tiene diferentes vertientes: aprendizaje supervisado, no supervisado, semi supervisado y por refuerzo. El aprendizaje supervisado cuenta con un conocimiento a priori, es decir para la tarea de clasificar un objeto dentro de una categoría o clase se tienen modelos ya clasificados (objetos agrupados que tienen características comunes). En la primera fase se tiene un conjunto de entrenamiento o de aprendizaje (para el diseño del clasificador) y otro llamado de prueba o de validación (para clasificación), estos sirven para construir un modelo o regla general para la clasificación. En la segunda fase se clasifican los objetos o muestras de las que se desconoce la clase a las que pertenecen. A diferencia del aprendizaje supervisado, el aprendizaje no supervisado o clustering no se cuenta con conocimiento a priori, por lo que se tiene un área de entrenamiento disponible para la tarea de clasificación. En este tipo de clasificación contamos con "objetos" o muestras que tiene un conjunto de características, de las que no sabemos a qué clase o categoría pertenece, entonces la finalidad es el descubrimiento de grupos de "objetos" cuyas características afines nos permitan separar las diferentes clases.

En ocasiones, es muy complicado disponer de un conjunto de datos completamente etiquetado. Este tipo de aprendizaje tiene un poco de los dos anteriores. Usando este enfoque, se comienza etiquetando manualmente algunos de los datos. Una vez se tenga una pequeña porción de datos etiquetados, se entrena uno o varios algoritmos de aprendizaje supervisado sobre esa pequeña parte de datos etiquetados y se usan los modelos resultantes del entrenamiento para etiquetar el resto de los comentarios. Finalmente, se entrena un algoritmo de aprendizaje supervisado utilizando como etiquetas las etiquetadas manualmente y las generadas por los modelos anteriores. Por último, el aprendizaje por refuerzo es un método de aprendizaje automático que se basa en recompensar los comportamientos deseados y penalizar los no deseados. Es un aprendizaje que fija objetivos a largo plazo para obtener una recompensa general máxima y lograr una solución óptima. El juego es uno de los campos más utilizados para poner a prueba el aprendizaje por refuerzo. En estos casos, el agente recibe información sobre las reglas del juego y aprende a jugar por sí mismo. En un principio se comporta de manera aleatoria, pero con el tiempo empieza a aprender movimientos más sofisticados. Este tipo de aprendizaje se aplica también en otras áreas como la robótica, la optimización de recursos o sistemas de control.

En este trabajo se cuenta con un dataset público que posee 16 características de 6 diferentes tipos de frijol, por lo que, se utiliza el aprendizaje supervisado. En las siguientes secciones se explica la teoria y ejemplifica mediante el caso de estudio su aplicación.

II. Teoría

Los conceptos básicos de las teoría utilizada en este trabajo es descrita a continuación. Se tomó información de [1], [2], [3]

II-A. Clasificador bayesiano simple

El teorema de Bayes es utilizado para calcular la probabilidad de un suceso, teniendo información de antemano sobre ese suceso. Es posble calcular la probabilidad de un suceso A, sabiendo además que ese suceso cumple cierta característica que condiciona su probabilidad. El teorema de

la probabilidad total hace inferencia sobre un suceso B, a partir de los resultados de los sucesos A. Por su parte, Bayes calcula la probabilidad de A condicionado a B. Para calcular la probabilidad tal como la definió Bayes en este tipo de sucesos, es necesaria la siguiente fórmula:

$$P[A_n/B] = \frac{P[B/A_n] * P[A_n]}{\sum P[B/A_i] * P[A_i]} \quad (1)$$

donde B es el suceso sobre el que se tiene información previa y A_n son los distintos sucesos condicionados. En la parte del numerador se encuentra la probabilidad condicionada, y el denominador la probabilidad total.

Los clasificadores NaiveBayes (NBC por sus siglas en inglés) son algoritmos de aprendizaje automático simples pero potentes. Se basan en la probabilidad condicional y el teorema de Bayes. SE deben seguir los siguientes pasos para implemntar el algoritmo:

1. Convertir el conjunto de datos en una tabla de frecuencias
2. Crear una tabla de probabilidad calculando las correspondientes a que ocurran los diversos eventos o haciendo una distribución Gaussiana
3. La ecuación NaiveBayesse se usa para calcular la probabilidad posterior de cada clase
4. La clase con la probabilidad posterior más alta es el resultado de la predicción

II-B. Clasificación vecino más cercano (K-NN)

El clasificador de vecinos más cercanos se basa en calcular las distancias entre el datoa clasificar y los ejemplos de entrenamiento para decidir a que clase pertenece dependiendo de la menor distancia. No define de forma explícita una frontera de separación entre clases. La frontera se define implícitamente a partir de la distancia a las muestras del conjunto de entrenamiento. No requiere una representación de cada imagen en forma de vector, únicamente la definición de una función de distancia entre dos imágenes. Se considera un método de lazy learning debido a que la generalización más allá de los datos de entrenamiento es demorada hasta que se hace una pregunta al sistema.

De igual forma, K vecinos más cercanos es uno de los algoritmos de clasificación más básicos y esenciales en Machine Learning. Pertenece al dominio del aprendizaje supervisado y encuentra una aplicación intensa en el reconocimiento de patrones, la minería de datos y la detección de intrusos. Este algoritmo consiste en seleccionar un valor de K vecinos. Al momento del análisis los K datos más cercanos al valor que se desea predecir será la solución. Acá lo importante es seleccionar un valor de K acorde a los datos para tener una mayor precisión en la predicción. Si k es muy pequeño el modelo será muy sensitivo a puntos que son atípicos o con ruido. Si K es muy grande el modelo tiende a asignar siempre la clase más grande.

Algorithm 1 Clasificador vecinos cercanos NN

Require:
D ▷ datos de entrenamiento
k ▷ número de vecinos
t ▷ dato para clasificar
Ensure: c ▷ clase para clasificar t
$N \leftarrow 0$
for $d \in D$ **do**
  if $|N| \leq k$ **then**
    $N = N \cup d_i$
  else
    if $\exists\, u \in N$ tal que $sim(t, u) \geq sim(t, d)$ **then**
      $N = N - u$
      $N = N \cup d_i$
    end if
  end if
end for
$c =$ clases para los mejores $u \in N$ que son clasificados

Para el calculo de la distancia entre los vecinos se suelen utilizar la distancia euclidiana (ecuación 2) y distancia Manhattan (ecuación 3) siendo la primera la más común. La distancia euclidiana es un número positivo que indica la separación que tienen dos puntos en un espacio. La distancia entre dos puntos A y B de un espacio euclidiano es la longitud del vector AB perteneciente a la única recta que pasa por dichos puntos. Por otr lado, la distancia Manhattan dice que la distancia entre dos puntos es la suma de las diferencias absolutas de sus coordenadas. Es decir, es la suma de las longitudes de los dos catetos del triángulo rectangulo.

$$d_{ij} = \sqrt{\sum_{k=1}^{n} (x_{ki} - x_{kj})^2} \quad (2)$$

$$d(p.q) = \sum_{i}^{n} |p_i - q_i| \quad (3)$$

II-C. Árboles de decisión

La idea de la clasificación con árboles de decisión es simple: iterativamente se van generando particiones binarias (es decir de a dos agrupaciones) sobre la región de interés, buscando que cada nueva partición genere un subgrupo de datos lo más homogéneo posible. Primero se establece una condición. Dependiendo de si los datos cumplen o no la condición se tendrá una primera partición en dos subregiones. Y luego se repite el procedimiento anterior, una y otra vez, hasta que al final se obtengan agrupaciones lo más homogéneas posible, es decir con puntos que pertenezcan en lo posible a una sola categoría.

Para medir la homogeneidad se usa el índice Gini (ecuación 4), que mide el grado de impureza de un nodo: índices Gini iguales a cero indican nodos puros (es decir con datos que pertenecen a una sola categoría), mientras que índices mayores que cero y con valores hasta de uno indican nodos

con impurezas (es decir con datos de más de una categoría). La entropía por su parte, es una medida que se aplica para cuantificar el desorden de un sistema (ecuación 5). Si un nodo es puro su entropía es 0 y solo tiene observaciones de una clase, pero si la entropía es igual a 1, existe la misma frecuencia para cada una de las clases de observaciones.

$$Gini(t) = 1 - \sum_{i}^{n} p_i^2 \tag{4}$$

$$H = -\sum_{i}^{n} p_i * \log_2 p_i \tag{5}$$

Los pasos para realizar un árbol de desición para clasificar son los siguientes:

1. Para crear la raíz del árbol, es decir la primera partición, se toman todas las características y, para cada una de ellas, se definen todos los posibles umbrales a que haya lugar. Cada umbral será simplemente el punto intermedio entre dos valores consecutivos de cada característica.
2. Para cada uno de estos umbrales se calcula la partición (nodo izquierdo y nodo derecho) y para cada nodo hijo se calcula el índice Gini. Con estos nodos hijos se calcula la función de costo del nodo padre, que es el promedio ponderado de los índices Gini de sus hijos.
3. Se toma el umbral (o nodo padre resultante) que tenga la función de costo con el menor valor posible, indicando que la partición obtenida es la más homogénea de todas las analizadas.
4. Una vez se haya realizado esta partición, se repite el mismo procedimiento de forma iterativa para los nodos resultantes, exceptuando los que sean nodos hoja

II-D. Bosques aleatorios

El bosque aleatorio tiende a combinar cientos de árboles de decisión y luego entrena cada árbol de decisión en una muestra diferente de las observaciones. Las predicciones finales del bosque aleatorio se realizan promediando las predicciones de cada árbol individual. El algoritmo de bosque aleatorio también puede ayudarte a encontrar características que son importantes en tu conjunto de datos. Esto se debe al algoritmo de Boruta, que selecciona características importantes en un conjunto de datos. El algoritmo funciona completando los siguientes pasos:

1. El algoritmo selecciona muestras en forma aleatoria de la base de datos proporcionada.
2. El algoritmo creará un árbol de decisión para cada muestra seleccionada. Luego obtendrá un resultado de predicción de cada árbol creado.
3. A continuación, se realizará la votación para cada resultado previsto. Para un problema de clasificación, usará la moda, y para un problema de regresión, usará la media.
4. El algoritmo seleccionará el resultado de predicción más votado como predicción final.

II-E. Máquina de vectores de soporte (SVM)

Las máquinas de vectores de soporte son una técnica que encuentra la mejor separación posible entre clases. Normalmente, los problemas de aprendizaje automático tienen muchísimas dimensiones. Así que, en vez de encontrar la línea óptima, el SVM encuentra el hiperplano que maximiza el margen de separación entre clases. Los vectores de soporte son los puntos que definen el margen máximo de separación del hiperplano que separa las clases. Se llaman vectores, en lugar de puntos, porque estos puntos tienen tantos elementos como dimensiones tenga nuestro espacio de entrada. Es decir, estos puntos multidimensionales se representan con vector de n dimensiones.

Es bastante frecuente que los datos tengan ruido, que no estén etiquetados perfectamente, o que el problema sea tan difícil que, para unos pocos puntos, sea muy complicado clasificarlos correctamente. Para estos casos, es posible indicarle al SVM, que generalice bien para la mayoría de los casos, aunque algunos pocos casos del conjunto de entrenamiento no estén perfectamente clasificados. Lo que normalmente se busca es la construcción de modelos de aprendizaje automático que generalicen bien.

En algunas ocaciones no hay forma de encontrar un hiperplano que permita separar dos clases. En estos casos las clases no son linealmente separables. Para resolver este problema es posible usar el truco del kernel. El truco del kernel consiste en inventar una dimensión nueva en la que se encuentre un hiperplano para separar las clases. Al añadir una dimensión nueva, es posible separar las clases con una superficie de decisión. También hay métodos para separar los datos $(x_i.y_i)$ directamente aun no siendo separables linealmente, mediante funciones polinómicas y funciones de base radial (RBF).

II-F. Redes neuronales

Una red neuronal es un modelo simplificado que emula el modo en que el cerebro humano procesa la información. Funciona simulando un número elevado de unidades de procesamiento interconectadas como versiones abstractas de neuronas (Figura 1). Las unidades de procesamiento se organizan en capas. Hay tres partes normalmente en una red neuronal : una capa de entrada, con unidades que representan los campos de entrada; una o varias capas ocultas; y una capa de salida, con una unidad o unidades que representa el campo o los campos de destino, vease Figura 2. Los datos de entrada se presentan en la primera capa, y los valores se propagan desde cada neurona hasta cada neurona de la capa siguiente. al final, se envía un resultado desde la capa de salida.

La red aprende examinando los registros individuales, generando una predicción para cada registro y realizando ajustes a las ponderaciones cuando realiza una predicción incorrecta. Este proceso se repite muchas veces y la red sigue mejorando sus predicciones hasta haber alcanzado uno o varios criterios de parada. Al principio, todas las ponderaciones son aleatorias y las respuestas que resultan de la red son, posiblemente,

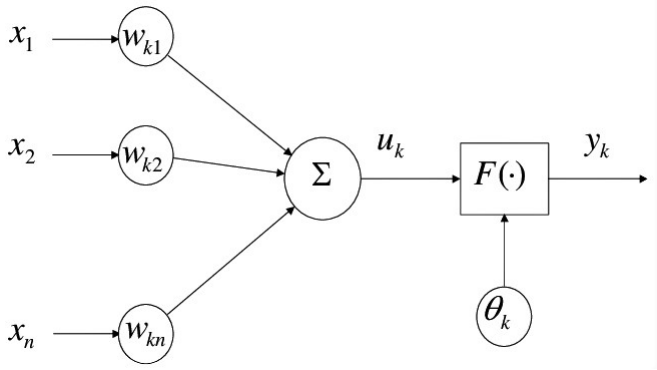

Figura 1. Estructura de un neurona artificial.

disparatadas. La red aprende a través del entrenamiento. Continuamente se presentan a la red ejemplos para los que se conoce el resultado, y las respuestas que proporciona se comparan con los resultados conocidos. La información procedente de esta comparación se pasa hacia atrás a través de la red, cambiando las ponderaciones gradualmente. A medida que progresa el entrenamiento, la red se va haciendo cada vez más precisa en la replicación de resultados conocidos. Una vez entrenada, la red se puede aplicar a casos futuros en los que se desconoce el resultado.

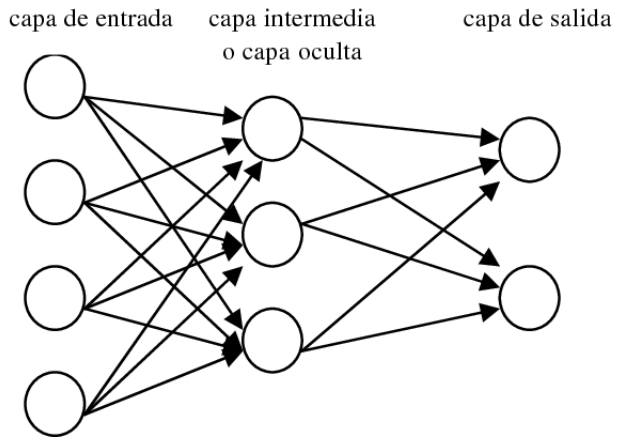

Figura 2. Estructura de red neuronal.

Saber que arquitectura de red neuronal es la ideal para el caso de estudio es el verdadero reto. Actualmente se cuentan con métodos heuristicos de optimización que pueden ser aplicados en cualquiera de los algoritmos descritos anteriormente. En este caso, al ser dificil de probar multiples combinaciones de capas y neuronas, es mejor implementar una heuristica sencilla para encontrar las combinaciones adecuadas. Una de esas técnicas es conocida como busqueda aleatoria. La búsqueda aleatoria implica generar y evaluar entradas aleatorias a una función objetivo. Se puede generar una muestra aleatoria de un dominio usando un generador de números pseudoaleatorios. Cada variable requiere un límite o rango bien definido y se puede muestrear un valor aleatorio uniforme del rango y luego evaluarlo. La función objetivo puede plantearse como una funcion lineal en combinación con las metricas de evaluación de interes como la exactitud o el F1-score o simplemente usar una de las metricas a maximizar.

II-G. Métricas de evaluación

En el aprendizaje automático la matriz de confusión es una herramienta que permite visualizar el desempeño de un algoritmo de aprendizaje supervisado. Cada columna de la matriz representa el número de predicciones de cada clase, mientras que cada fila representa a las instancias en la clase real. Nos permite ver qué tipos de aciertos y errores está teniendo nuestro modelo a la hora de pasar por el proceso de aprendizaje con los datos. EN la matriz de confusión existen 4 resultados posibles:

- Verdadero positivo (VP): El valor real es positivo y la prueba predijo tambien que era positivo. O bien una persona está enferma y la prueba así lo demuestra.
- Verdadero negativo (VN): El valor real es negativo y la prueba predijo tambien que el resultado era negativo. O bien la persona no está enferma y la prueba así lo demuestra.
- Falso negativo (FN): El valor real es positivo, y la prueba predijo que el resultado es negativo. La persona está enferma, pero la prueba dice de manera incorrecta que no lo está. Esto es lo que en estadística se conoce como error tipo II
- Falso positivo (FP): El valor real es negativo, y la prueba predijo que el resultado es positivo. La persona no está enferma, pero la prueba nos dice de manera incorrecta que silo está.

A partir de estas 4 opciones surgen las métricas de la matriz de confusión: Por una parte la exactitud y la precisión y por otra la Sensibilidad y la Especificidad. En este trabajo solo se utiliza la exactitud para comprobar la efectividad del algoritmo. Esa métrica se refiere a lo cerca que está el resultado de una medición del valor verdadero. Se representa como la proporción de resultados verdaderos (tanto verdaderos positivos como verdaderos negativos) dividido entre el número total de casos examinados (verdaderos positivos, falsos positivos, verdaderos negativos, falsos negativos).

$$exactitud = \frac{VP + VN}{VP + FP + FN + VN} \tag{6}$$

III. Resultados

Para realizar la práctica se programó en lenguaje Python los dos sistemas de regresión. Primero se obtuvo el conjunto de datos publico Dry Bean Dataset el cual contiene información de 7 distintos tipos de fijoles como se muestra en la Tabla 1. Se tuvo que hacer un pre-procesamiento de los datos debido las clases son categoricas y los datos se normalizaron (ver anexo A). los datos se dividieron para entrenamiento y pruebas en una proporcion 80/20 respectivamente.

Con los datos pre-procesados se entrenaron los algoritmos mencionados en la sección anterior con los siguientes parametros:

- KNN
  - valor k=5
  - distancia euclidiana
- Árbol de desición
  - Criterio de homogeneidad: gini/entropía
- Bosques aleatorios
  - Número de árboles: 10
- SVM
  - kernel: función de base radial

Para definir el número de capas ocultas de la red neuronal, el número de neuronas por capa, el ratio de aprendizaje y el optimizador se hace uso de la libreria keras-tuner que cuenta con el método de busqueda aleatoria. El algoritmo hizo 20 iteraciones. La función objetivo se elige como la exactitud y se definieron los siguientes valores como área de busqueda:

- Número de capas ocultas: entre 0 y 8
- Número de neuronas por capa: entre 16 y 256
- Valor de ratio de aprendizaje: entre $1e^{-4}$ y $1e^{-2}$
- Optimizadores: Adam y Stochastic Gradient Descent

Los valores que finalmente resultaron optimos de la busqueda fueron:

- Número de capas ocultas: 7
- Número de neuronas por capa: 124
- Valor de ratio de aprendizaje: 0.00055
- Optimizadores: Adam

Dry-Bean dataset	
Dato	Descripción
Área	El área de una zona de frijol
Perímetro	La longitud de su borde
Longitud del eje mayor	Distancia más larga entre los extremos
Longitud del eje menor	Línea más larga con el frijol perpendicular al eje
Relación de aspecto	Relación entre las longitudes
Excentricidad	Excentricidad de la elipse
Área convex	Píxeles en el polígono convexo más pequeño
Diámetro equivalente	Diámetro que tiene la misma área que un frijol
Extensión	Relación de los píxeles delimitadores al área del frijol
Solidez	relación píxeles capa convexa y dentro de los frijoles
Redondez	Calculando $\frac{4piA}{P^2}$
Compacidad	Mide la redondez de un objeto $\frac{Ed}{L}$
Factor de forma 1	SF1
Factor de forma 2	SF2
Factor de forma 3	SF3
Factor de forma 4	SF4
Clase	(Seker, Barbunya, Bombay, Cali, Dermosan, Horozy Sira)

Tabla I
CONJUNTO DE DATOS DRY-BEAN

Los resultados de los experimentos se pueden observar en la Tabla 2. En la Figura 3 se muestra la matriz de confusión de SVM, clasificador con mejor eficacia y eficiencia.

Método de clasificación	Exactitud
Clasificador bayesiano	89.68 %
K-NN	92.69 %
Árboles de desición	gini: 88.6 % entropía: 90.89 %
Bosques aleatorios	92.03 %
SVM	93.46 %
Redes neuronales	93.88 %

Tabla II
RESULTADOS DE LOS EXPERIMENTOS

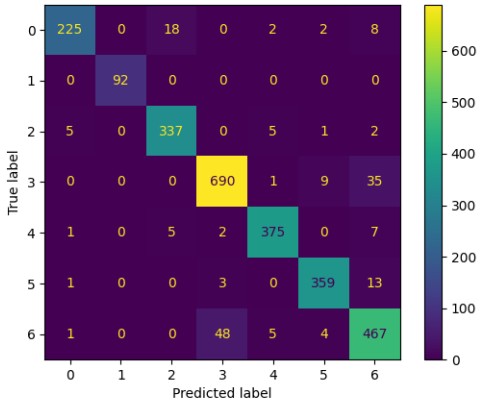

Figura 3. Matriz de confusión de SVM.

CONCLUSIONES

Como se pudo observar en los resultados, la máquina de vectores de soporte y las redes neuronales tuvieron los mejores resultados. Apesar de que las redes neuronales en cuanto a tiempo tarda mucho más tiempo que un SVM el desempeño no es razonablemente mayor para preferlas sobre SVM. Para este caso de estudio probablemente sea mejor utilizar un SVM ya que tiene menor carga computacional por una exactitud practicamente similiar a una red neuronal. El problema con las redes neuronales es que puede tener multiples combinaciones en su arquitectura que si bien se puede aplicar un método de optimización, esto implica un proceso extra sobre los otros métodos.

El dataset probablemente sea necesario modificarlo para que el desempeño aumente en los métodos de clasificación ya que por más que se pueda optimizar un algoritmo si los datos no son optimos no mejora la exactitud. Lo mejor es realizar un análisis de componentes principales para identificar que variables o caracteristicas no aportan realmente información importante. Esto ayuda tambien a reducir dimensiones por lo que el proceso será más rápido. SI bien hay muchos factores que influyen en el desempeño se pueden prevenir problemas realizando un análisis de los datos. Además al no utilizar validación cruzada puede ocurrir sobre entrenamiento, es decir, no generalizar el modelo. Métodos como árboles de desición son sensibles al sobre ajuste por lo que a futuro se deben comprobar los resultados con más pruebas.

REFERENCIAS

[1] A. Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow, 3a ed. Sebastopol: O'Reilly, 2022.

[2] S. J. Russell, Inteligencia artificial: Un enfoque moderno, 2a ed. Madrid: Pearson Prentice Hall, 2004.

[3] C. Huyen, Designing Machine Learning Systems. Sebastopol: O'Reilly, 2022.

IV. ANEXO A

IV-A. Procesamiento de datos

Lectura de conjunto de datos y separandolos en variables dependientes e independientes.

```
dataset=pd.read_csv("Dry_Bean_Dataset.csv")
X=dataset.iloc[:,:-1].values
Y=dataset.iloc[:,-1].values
```

Codificando datos categorticos

```
labelencoder_Y=LabelEncoder()
Y=labelencoder_Y.fit_transform(Y)
```

Normalizado de datos.

```
sc_X=StandardScaler()
X=sc_X.fit_transform(X)
```

Separando el conjunto de datos.

```
X_train, X_test, Y_train, Y_test=
    train_test_split(X,Y,test_size=0.2,
    random_state=0)
```

IV-B. Naive Bayes

```
classifier=GaussianNB()
classifier.fit(X_train, Y_train)
```

IV-C. K-NN

```
classifier=KNeighborsClassifier(n_neighbors=5,
    metric='euclidean')
classifier.fit(X_train, Y_train)
```

IV-D. Árboles de desición

```
classifier=DecisionTreeClassifier(criterion='
    entropy', random_state=0)
classifier.fit(X_train, Y_train)
```

IV-E. Bosque aleatorio

Ejecución del bosque aleatorio estableciendo 10 árboles y calculo de las metricas de evaluación.

```
classifier=RandomForestClassifier(n_estimators
    =10, random_state=0)
classifier.fit(X_train, Y_train)
```

IV-F. SVM

```
classifier=SVC(kernel='rbf', random_state=0)
classifier.fit(X_train, Y_train)
```

IV-G. Red neuronal

```
model=Sequential()
model.add(Flatten())
for _ in range(7):
    model.add(Dense(124, activation='relu'))
model.add(Dense(7, activation='softmax'))
model.compile(loss='categorical_crossentropy',
    optimizer='adam', metrics
    =['accuracy'])
model.fit(X_train, Y_train, epochs=100,
    batch_size=32)
```

V. ANEXO B

V-A. Optimización de red neuronal

```
def construir_modelo(hp):
n_hidden=hp.Int("n_hidden", min_value=0,
    max_value=8, default=2)
n_neurons=hp.Int("n_neurons", min_value=16,
    max_value=256)
learning_rate=hp.Float("learning_rate",
    min_value=1e-4, max_value=1e-2, sampling="
    log")
optimizer=hp.Choice("optimizer", values=["sgd",
    "adam"])
if optimizer=="sgd":
    optimizer=optimizers.SGD(learning_rate=
        learning_rate)
else:
    optimizer=optimizers.Adam(learning_rate=
        learning_rate)

model=Sequential()
model.add(Flatten())
for _ in range(n_hidden):
    model.add(Dense(n_neurons, activation='relu'
        ))
model.add(Dense(7, activation='softmax'))
model.compile(loss='categorical_crossentropy',
    optimizer=optimizer, metrics=['accuracy'])
return model
```