Asignatura	Datos del Equipo	Fecha
Aprendizaje Automático	Nombre: Gabriel	06/03/2023
	Apellidos: Garcia Zambrano	

CLASIFICACIÓN CON MÁQUINA DE VECTORES DE SOPORTE Y REDES DE NEURONAS

RESUMEN: El presente trabajo de investigación se analizan dos técnicas de aprendizaje automático: las Máquinas de Vectores de Soporte (SVM) y las Redes Neuronales. Además, se detallan los tipos de variables y análisis de datos utilizados en el conjunto de datos Çlasificación de precios de teléfonos móviles", y se muestra una matriz de correlación para entender mejor la relación entre las variables. El objetivo de este estudio es comparar la eficacia de ambos modelos en la tarea de clasificación de precios de teléfonos móviles. Se presenta un modelo SVM que logra un alto porcentaje de precisión del 97 %, y se explica cómo se utilizaron técnicas de iteración de hiperparámetros para mejorar aún más el resultado. Por otro lado, se detalla la construcción de varios modelos de redes neuronales, con un proceso de ajuste de parámetros para lograr una precisión del 94.45 %. En conclusión, este estudio muestra que ambas técnicas pueden lograr altas precisiones en la clasificación de precios de teléfonos móviles, y se discute las ventajas y desventajas de cada modelo.

Keywords: *Inteligencia Artificial, Python, Aprendizaje Automático, Máquinas de vectores de soporte, Redes Neuronales, Mobile Price classification, análisis de datos, matriz de correlación*

1. INTRODUCCIÓN: INTELIGENCIA ARTIFICIAL Y APRENDIZAJE AUTOMÁTICO

La *inteligencia artificial* (IA) se trata de dotar a las máquinas con la capacidad de llevar a cabo tareas que, por lo general, solo podrían ser realizadas por seres humanos gracias a habilidades como el aprendizaje, la lógica y la percepción. En el campo de la IA hay múltiples definiciones, pero una de las más utilizadas fue propuesta por el investigador John McCarthy en 1956. Según McCarthy, la IA se enfoca en la creación de máquinas inteligentes a través de la ciencia y la ingeniería (1).

El aprendizaje automático, también conocido como *machine learning*, se refiere al conjunto de técnicas y algoritmos que permiten a las máquinas aprender a partir de datos, sin ser programadas explícitamente para hacerlo (2)(3)(4).

Existen tres tipos principales de aprendizaje automático:

- **Aprendizaje supervisado**: en este tipo de aprendizaje, el algoritmo recibe datos etiquetados, es decir, datos que ya tienen la respuesta correcta, y aprende a asociar características o patrones específicos con esa respuesta. Luego, el algoritmo puede utilizar ese conocimiento para hacer predicciones precisas sobre nuevos datos.

- **Aprendizaje no supervisado**: en este tipo de aprendizaje, el algoritmo recibe datos sin etiquetas y debe encontrar patrones y relaciones por sí mismo. Este tipo de aprendizaje es útil para la identificación de grupos o categorías de datos, así como para

la detección de anomalías.

- **Aprendizaje por refuerzo**: en este tipo de aprendizaje, el algoritmo aprende a través de la interacción con un entorno y la retroalimentación que recibe en función de sus acciones. El objetivo del algoritmo es maximizar una recompensa a largo plazo, por lo que aprende a tomar decisiones que le permitan alcanzar ese objetivo. Este tipo de aprendizaje se utiliza comúnmente en robótica y juegos.

2. MÁQUINA DE VECTORES DE SOPORTE

Las Máquinas de Vectores de Soporte (SVM) representan una metodología de aprendizaje supervisado que se aplica tanto para la clasificación como para la regresión. Para la clasificación, las SVM buscan identificar un hiperplano en un espacio de alta dimensión que permita una separación eficiente de los puntos de datos en distintas categorías. En el caso de la regresión, el objetivo es encontrar la función que se ajuste de manera óptima a los puntos de datos (5) (6).

El proceso de entrenamiento de una SVM implica hallar el hiperplano que maximice la distancia entre los puntos de datos de cada categoría, es decir, el "margen máximo", que se convierte en la solución más adecuada para una separación correcta de los datos en diferentes categorías.dimensión.

El "margen.[es] la distancia que existe entre el hiperplano y los puntos de datos más cercanos de cada categoría. Los "vectores de soporte"son los puntos de datos que se encuentran en el margen y son cruciales porque definen la ubicación del hiperplano y su habilidad para clasificar nuevos datos.

En aquellos casos en los que los datos no puedan ser separados mediante un hiperplano lineal, las SVM recurren a la técnica denominada "kernel trick"que posibilita proyectar los datos a un espacio de mayor dimensión donde la separación sí puede ser realizada. El kernel es una función que mide la similitud entre dos puntos de datos en el espacio original y se utiliza para construir el hiperplano en el espacio de mayor dimensión.

Una vez que se entrena una SVM, esta puede utilizarse para clasificar nuevos datos según su ubicación en relación al hiperplano. Los datos que se encuentran en un lado del hiperplano se clasifican en una categoría y los que se encuentran en el otro lado, en otra categoría. Las Máquinas de Vectores de Soporte son un método potente de aprendizaje supervisado aplicable a la clasificación y regresión. A través de la optimización del margen máximo, las SVM encuentran el hiperplano más adecuado para separar de manera eficiente los datos en distintas categorías(7) (8).

2.1. Redes de Neuronas

Una neurona es la unidad básica del cerebro y sistema nervioso que transmite información eléctrica y química. En inteligencia artificial se utilizan las neuronas artificiales como una representación simplificada de la neurona biológica. Una neurona artificial consiste en una entrada, pesos sinápticos, una función de activación y una salida, que juntos forman las redes neuronales artificiales. En resumen, las neuronas artificiales son una versión simplificada de las neuronas biológicas que se utilizan en redes neuronales artificiales(9).

Las redes neuronales son un conjunto de algoritmos de aprendizaje automático inspirados en la estructura y funcionamiento del cerebro humano. Cada red neuronal está compuesta por neuronas artificiales que se organizan en capas y están interconectadas mediante sinapsis artificiales. El proceso de funcionamiento de una red neuronal implica tres etapas: entrada de datos, procesamiento y salida de datos(10).

Las redes neuronales convolucionales son un tipo de red neuronal que se utilizan principalmente para tareas de visión por computadora, como la detección de objetos o la clasificación de imágenes. Las redes neuronales recurrentes se utilizan para procesamiento de lenguaje natural y otras tareas de secuencia de datos, como el reconocimiento de voz y la predicción de series de tiem-

po. Las redes neuronales de propagación hacia atrás se utilizan para la clasificación y regresión de datos(11).

Además, las redes neuronales tienen la capacidad de aprender de los datos y mejorar su precisión a través del entrenamiento. Durante el entrenamiento, se ajustan los pesos de las sinapsis artificiales para que la red neuronal pueda hacer inferencias más precisas sobre los datos de entrada(12).

3. DESARROLLO

En esta sección se describe el objetivo general, objetivos particulares, librerías utilizadas, así como la descripción paso a paso para generar los modelos de SVM y Redes neuronales para la predicción de precios en el dataset "Mobile Price Classification".

3.1. Objetivo General

El objetivo de esta actividad es la comprensión y generación de modelos de aprendizaje automático utilizando máquinas de vector de soporte y redes de neuronas en el Dataset *"Mobile Price Classification"*. Se busca comprender de forma práctica las diferencias en el proceso de entrenamiento de ambos tipos de modelos.

3.1.1. Objetivos Particulares

1. Descargar los archivos *test.csv* , *train.csv* y *predicted_prices.csv*

2. Analizar los datos importandos mediante Python

3. Utilizar proceso de entrenamiento SVM y analizar los resultados

4. Utilizar proceso de entrenamiento de Red Neuronal y analizar los resultados

5. Comparar ambos modelos

3.2. Software Utilizado

Para el diseño y generación de modelos se utilizo Lenguaje *Python* y a continuación se mencionan las librerías empleadas: pandas, sklearn, numpy, seaborn, matplotlib y keras

3.2.1. Procedimiento: Análisis de datos

El conjunto de datos "Mobile Price Classificationçontiene varias variables numéricas que pueden ser relevantes para la tarea de clasificación de precios de teléfonos móviles. A continuación, se describen las variables numéricas y sus estadísticas descriptivas más importantes en el archivo "train.csv":

1. **battery_power**: Capacidad de la batería en mAh (miliamperios-hora). Rango: 501 a 1998 mAh. Media: 1238.51 mAh. Desviación estándar: 439.42 mAh.

2. **clock_speed**: Velocidad del procesador en GHz (Gigahertz). Rango: 0.5 a 3.0 GHz. Media: 1.52 GHz. Desviación estándar: 0.82 GHz.

3. **fc**: Megapíxeles de la cámara frontal. Rango: 0 a 19 megapíxeles. Media: 4.31 megapíxeles. Desviación estándar: 4.35 megapíxeles.

4. **int_memory**: Memoria interna en GB (Gigabytes). Rango: 2 a 64 GB. Media: 32.05 GB. Desviación estándar: 18.15 GB.

5. **m_dep**: Grosor del teléfono en cm (centímetros). Rango: 0.1 a 1.0 cm. Media: 0.5 cm. Desviación estándar: 0.29 cm.

6. **mobile_wt**: Peso del teléfono en gramos (g). Rango: 80 a 200 g. Media: 140.3 g. Desviación estándar: 35.21 g.

7. **n_cores**: Número de núcleos del procesador. Rango: 1 a 8 núcleos. Media: 4.48 núcleos. Desviación estándar: 2.29 núcleos.

8. **pc**: Megapíxeles de la cámara principal. Rango: 0 a 20 megapíxeles. Media: 10.51 megapíxeles. Desviación estándar: 5.93 megapíxeles.

9. **px_height**: Resolución vertical de la pantalla en píxeles. Rango: 0 a 1960 píxeles. Media: 645.11 píxeles. Desviación estándar: 443.78 píxeles.

10. **px_width**: Resolución horizontal de la pantalla en píxeles. Rango: 500 a 1998 píxeles. Media: 1251.94 píxeles. Desviación estándar: 432.46 píxeles.

11. **ram**: Memoria RAM en MB (Megabytes). Rango: 256 a 3998 MB. Media: 2129.14 MB. Desviación estándar: 1084.63 MB.

Para el caso de las variables categóricas que pueden ser relevantes para la tarea de clasificación de precios de teléfonos móviles. A continuación, se enumeran las diferentes categorías para cada variable categórica en el archivo "train.csv", junto con la frecuencia de cada una de ellas:

1. **battery_power_range**: Rango de capacidad de la batería en mAh. Categorías y frecuencias: [500, 1000): 408 [1000, 1500): 297 [1500, 2000): 298

2. **blue**: Indica si el teléfono tiene Bluetooth o no. Categorías y frecuencias: 0: 1010 1: 990

3. **clock_speed_range**: Rango de velocidad del procesador en GHz. Categorías y frecuencias: [0, 1): 199 [1, 2): 693

4. **dual_sim**: Indica si el teléfono tiene capacidad para dos tarjetas SIM o no. Categorías y frecuencias: 0: 952 1: 1048

5. **fc_range**: Rango de megapíxeles de la cámara frontal. Categorías y frecuencias: [0, 5): 1035 [5, 10): 409 [10, 15): 78

6. **four_g**: Indica si el teléfono tiene capacidad para conectividad 4G o no. Categorías y frecuencias: 0: 877 1: 1123

7. **int_memory_range**: Rango de memoria interna en GB. Categorías y frecuencias: [0, 8): 463 [8, 16): 271 [16, 24): 144 [24, 32): 74

8. **m_dep_range**: Rango de grosor del teléfono en cm. Categorías y frecuencias: [0, 0.2): 398 [0.2, 0.4): 391 [0.4, 0.6): 198 [0.6, 0.8): 13

9. **mobile_wt_range**: Rango de peso del teléfono en gramos. Categorías y frecuencias: [80, 120): 187 [120, 160): 425 [160, 200): 343

10. **n_cores_range**: Rango de número de núcleos del procesador. Categorías y frecuencias: [1, 3): 238 [3, 5): 440 [5, 7): 175

11. **pc_range**: Rango de megapíxeles de la cámara principal. Categorías y frecuencias: [0, 5): 482 [5, 10): 442

Cabe mencionar que las columnas de dataset son : (['battery_power', 'blue', 'clock_speed', 'dual_sim', 'fc', 'four_g', 'int_memory', 'm_dep', 'mobile_wt', 'n_cores', 'pc', 'px_height', 'px_width', 'ram', 'sc_h', 'sc_w', 'talk_time', 'three_g', 'touch_screen', 'wifi', 'price_range'], dtype='object')
Además se puede utilizar la función describe() para obtener estadística, a continuación se muestra el resultado:

```
data = pd.read_csv('train.csv')
# Seleccionar solo las variables numéricas
numeric_vars = data.select_dtypes(include=['int64', 'float64'])
# Describir las variables numéricas
print(numeric_vars.describe())
```

```
       battery_power       blue  clock_speed     dual_sim           fc  \
count    2000.000000  2000.0000  2000.000000  2000.000000  2000.000000
mean     1238.518500     0.4950     1.522250     0.509500     4.309500
std       439.418206     0.5001     0.816004     0.500035     4.341444
min       501.000000     0.0000     0.500000     0.000000     0.000000
25%       851.750000     0.0000     0.700000     0.000000     1.000000
50%      1226.000000     0.0000     1.500000     1.000000     3.000000
75%      1615.250000     1.0000     2.200000     1.000000     7.000000
max      1998.000000     1.0000     3.000000     1.000000    19.000000

            four_g   int_memory        m_dep    mobile_wt      n_cores  ...  \
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000  ...
mean      0.521500    32.046500     0.501750   140.249000     4.520500  ...
std       0.499662    18.145715     0.288416    35.399655     2.287837  ...
min       0.000000     2.000000     0.100000    80.000000     1.000000  ...
25%       0.000000    16.000000     0.200000   109.000000     3.000000  ...
50%       1.000000    32.000000     0.500000   141.000000     4.000000  ...
75%       1.000000    48.000000     0.800000   170.000000     7.000000  ...
max       1.000000    64.000000     1.000000   200.000000     8.000000  ...

         px_height     px_width          ram         sc_h         sc_w  \
count  2000.000000  2000.000000  2000.000000  2000.000000  2000.000000
mean    645.108000  1251.515500  2124.213000    12.306500     5.767000
std     443.780811   432.199447  1084.732044     4.213245     4.356398
min       0.000000   500.000000   256.000000     5.000000     0.000000
25%     282.750000   874.750000  1207.500000     9.000000     2.000000
50%     564.000000  1247.000000  2146.500000    12.000000     5.000000
75%     947.250000  1633.000000  3064.500000    16.000000     9.000000
max    1960.000000  1998.000000  3998.000000    19.000000    18.000000

         talk_time      three_g  touch_screen         wifi  price_range
count  2000.000000  2000.000000   2000.000000  2000.000000  2000.000000
mean     11.011000     0.761500      0.503000     0.507000     1.500000
std       5.463955     0.426273      0.500116     0.500076     1.118314
min       2.000000     0.000000      0.000000     0.000000     0.000000
25%       6.000000     1.000000      0.000000     0.000000     0.750000
50%      11.000000     1.000000      1.000000     1.000000     1.500000
75%      16.000000     1.000000      1.000000     1.000000     2.250000
max      20.000000     1.000000      1.000000     1.000000     3.000000

[8 rows x 21 columns]
```

Figura 1: Análisis de datos.

También es posible obtener con facilidad un boxplot y un histplot los cuáles muestrab la distribución de los datos de tipo numérico.

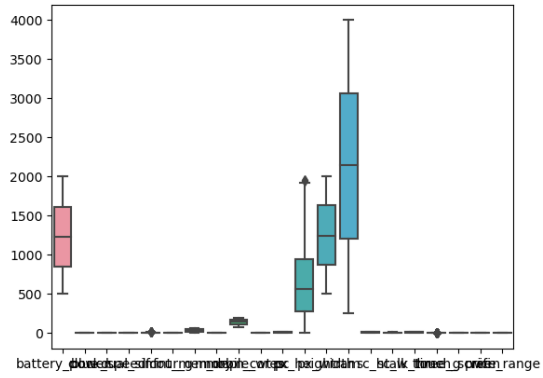

Figura 2: Boxplot de datos numéricos.

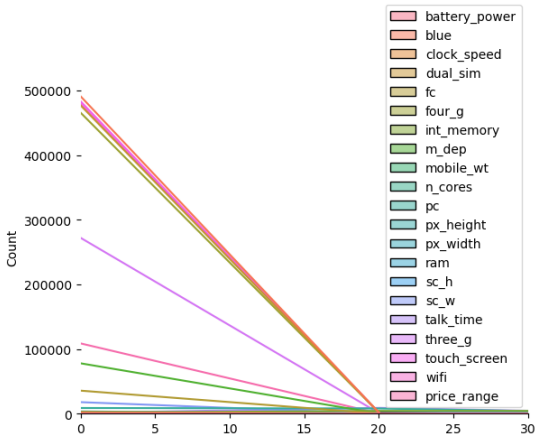

Figura 3: Histplot de datos numéricos.

Una vez analizados los datos se propone construir una matriz de correlaciones (Ver Figura 4), para la realización de dicha matriz como primer paso se consideró no utilizar ningún filtro "mask", pero la relación entre valores era demasiado baja (menos de 0.05), por lo que bajo mi criterio consideré utilizar un filtro para los valores mayores a 0.05 y menores a 1.

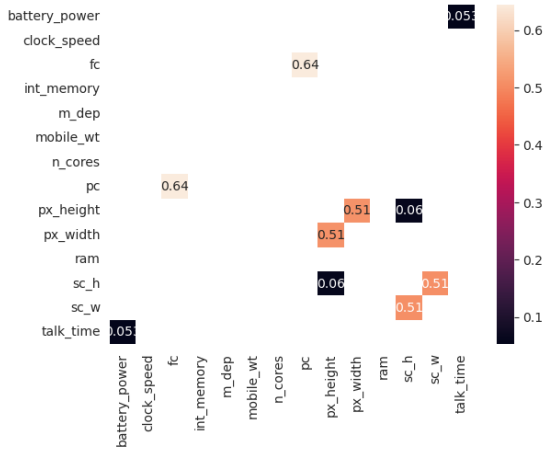

Figura 4: Matriz de Correlaciones utilizando

3.3. Construcción de modelo de Máquinas de vectores de soporte

Una vez analizados los datos de forma estadística se busca utilizar Máquinas de vectores de soporte como un algoritmo de aprendizaje supervisado para construir un modelo de clasificación que prediga el rango de precio de un teléfono móvil en función de sus características.

Asignatura	Datos del Equipo	Fecha
Aprendizaje Automático	Nombre: Gabriel	06/03/2023
	Apellidos: Garcia Zambrano	

Como primer paso se debe importar los archivos *test.csv* y *train.csv* mediante la librería pandas, a continuación se deben dividir los conjuntos de datos en caracteristicas y etiquetas así como dividir los datos de entrenamiento en connjuntos de entrenamiento y validación, posteriormente decido crear un clasificador SVM mediante la función **clf = SVC(kernel="linear")** y con ello calculo el porcentaje de presición de la predicción, a continuación se observa el procedimiento antes mencionado:

```python
# Leer los conjuntos de entrenamiento y prueba desde archivos CSV
train_data = pd.read_csv("train.csv")
test_data = pd.read_csv("test.csv")
# Dividir los conjuntos de datos en características y etiquetas
X_train = train_data.drop("price_range", axis=1)
y_train = train_data["price_range"]
X_test = test_data.drop("id", axis=1)
# Dividir los datos de entrenamiento en conjuntos de entrenamiento y validación
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2
                                                  , random_state=42)
# Crear un clasificador SVM y entrenarlo con el conjunto de entrenamiento
clf = SVC(kernel="linear")
clf.fit(X_train, y_train)
y_pred = clf.predict(X_val)
# Calcular la precisión de la predicción
accuracy = accuracy_score(y_val, y_pred)
print("Precisión de la predicción: {:.2f}%".format(accuracy * 100))
# Predecir las etiquetas para el conjunto de prueba
y_test_pred = clf.predict(X_test)
# Guardar las predicciones en un archivo CSV
test_data["price_range"] = y_test_pred
test_data[["id", "price_range"]].to_csv("predicted_prices.csv", index=False)

```

```
Precisión de la predicción: 97.00%
```

Figura 5: Primer modelo de SVM propuesto

Durante la experimentación de resultados consideré modificar los Hiperparámetros, pero al ser demasiados decidí emplear un algoritmo de iteración de los parámetros C [0.1,1,10,100], Gamma [1,0.1,0.01,0.001] y kernel ['rbf', 'linear', 'poly', 'sigmoid'], para así encontrar cuáles hiperparámetros se ajustaban mejor y encontrar con ello mejorar los resultados, a continuacón se ilustra una porción de lo obtenido así como la matriz de confusión:

```python
# Creación del modelo
model = SVC()

# Definición de los parámetros a ajustar
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel':

# Ajuste de los parámetros utilizando la técnica de validación cruzada
grid = GridSearchCV(model, param_grid, refit=True, verbose=3)
grid.fit(X_train, y_train)

# Predicción utilizando el modelo ajustado
y_pred = grid.predict(X_val)

# Calcular la precisión de la predicción
accuracy = accuracy_score(y_val, y_pred)
print("Precisión de la predicción: {:.2f}%".format(accuracy * 100))

# Predicción para el conjunto de prueba
y_test = grid.predict(X_test)

# Calcular la matriz de confusión
conf_mat = confusion_matrix(y_val, y_pred)
print("Matriz de confusión:")
print(conf_mat)
```

```
[CV 3/5] END .C=100, gamma=0.001, kernel=linear;, score=0.984 total time=  12.0s
[CV 4/5] END .C=100, gamma=0.001, kernel=linear;, score=0.959 total time=   1.9s
[CV 5/5] END .C=100, gamma=0.001, kernel=linear;, score=0.984 total time=  22.2s
[CV 1/5] END ...C=100, gamma=0.001, kernel=poly;, score=0.959 total time=   0.1s
[CV 2/5] END ...C=100, gamma=0.001, kernel=poly;, score=0.956 total time=   0.1s
[CV 3/5] END ...C=100, gamma=0.001, kernel=poly;, score=0.944 total time=   0.1s
[CV 4/5] END ...C=100, gamma=0.001, kernel=poly;, score=0.953 total time=   0.0s
[CV 5/5] END ...C=100, gamma=0.001, kernel=poly;, score=0.969 total time=   0.1s
[CV 1/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.256 total time=   0.1s
[CV 2/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.256 total time=   0.1s
[CV 3/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.253 total time=   0.1s
[CV 4/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.253 total time=   0.1s
[CV 5/5] END C=100, gamma=0.001, kernel=sigmoid;, score=0.253 total time=   0.1s
Precisión de la predicción: 97.25%
Matriz de confusión:
[[ 99   6   0   0]
 [  0  91   0   0]
 [  0   2  89   1]
 [  0   0   2 110]]
```

Figura 6: Modificación de hiperparámetros y matriz de confusión

Por último decidí observar el comportamiento de los parámetros C, gamma y kernel para observar cómo cambia la precisión y de este modo aprender prácticamente cómo es que los parámetros afectan directamente los resultados, así como comprender que a niveles muy altos el modelo se sobre entrena y genera peores resultados en tiempos más extensos.

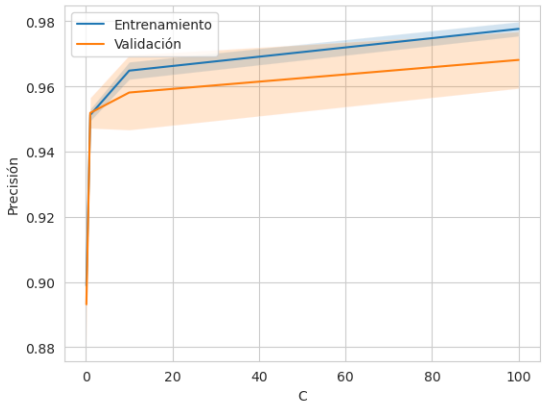

Figura 7: Curva de precisión en función del parámetro C para Entrenamiento y Validación

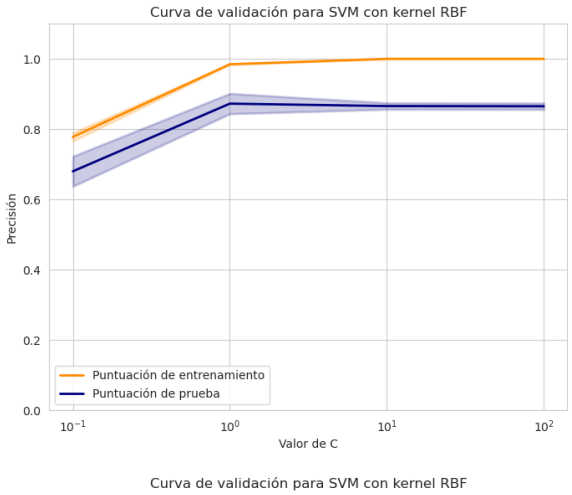

Figura 8: Curva de validación para SVM con kernel RBF en función de C

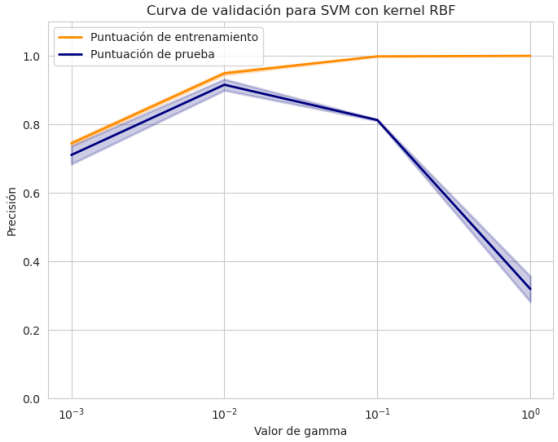

Figura 9: Curva de validación para SVM con kernel RBF en función de gamma

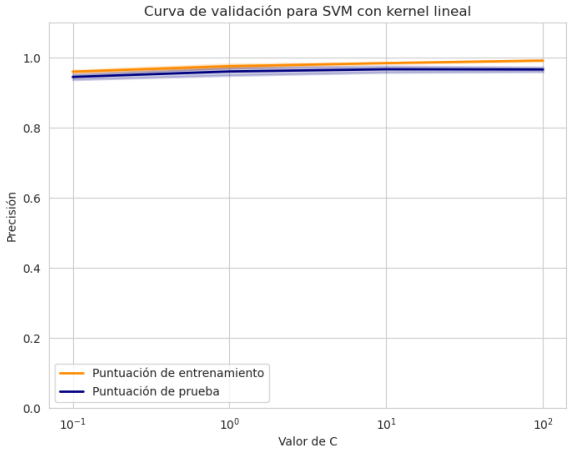

Figura 10: Curva de validación para SVM con kernel lineal

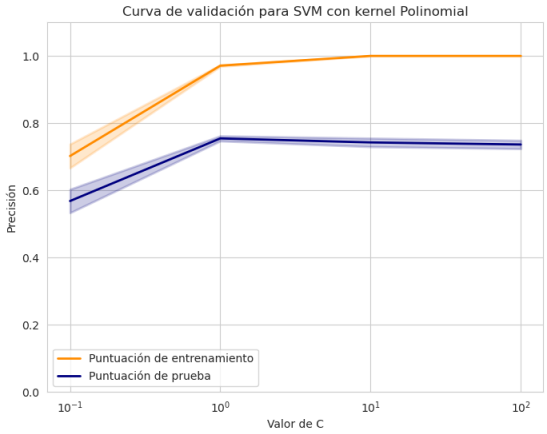

Figura 11: Curva de validación para SVM con kernel polinomial

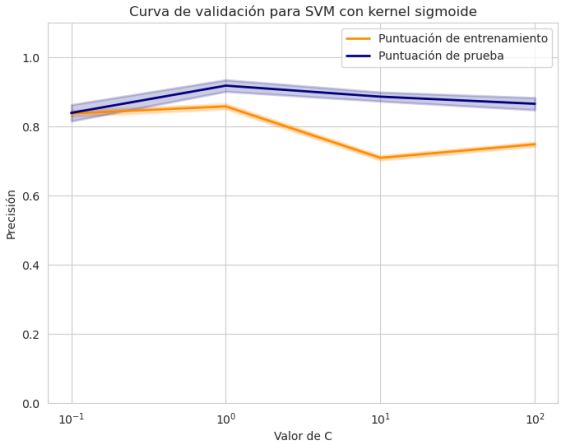

Figura 12: Curva de validación para SVM con kernel sigmoide

3.4. Construcción del modelo de Redes Neuronales

El siguiente tipo de modelo también tiene la caracteristica de poder predecir el rango de precio de un teléfono movil en función de las caracteristicas, a continuación se describen los pasos y funciones que utilice con la finalidad de incrementar el porcentaje de acierto.

Como primer paso importe el archivo de entrenamiento mediante la librería de pandas, posteriormente dividi mis datos en entrenamiento y prueba, una vez realizada la

división añadí capas ocultas con activación relu y una capa de salida con activación softmax, por último añadi un modelo de adam y lo entrené para 200 epochs, el resultado fue de 87.75 % de precisión a las 200 Epochs, la siguiente FIgura ilustra el proceso.

```
# Lectura de los archivos CSV
data = pd.read_csv("train.csv")
scaler = StandardScaler()
X = scaler.fit_transform(data.drop("price_range", axis=1))

y = data["price_range"]

# División del dataset en datos de entrenamiento y de prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Creación del modelo de red neuronal
model = Sequential()

# Añadir capas ocultas
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.01)))
model.add(Dropout(0.5))
model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.01)))
model.add(Dropout(0.5))

# Añadir capa de salida
model.add(Dense(4, activation='softmax'))

# Compilación del modelo
optimizer = Adam(learning_rate=0.0001)
model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])

# Entrenamiento del modelo
early_stop = EarlyStopping(monitor='val_loss', patience=20)
history = model.fit(X_train, y_train, epochs=200, batch_size=128, validation_split=0.1, callbacks=[early_stop])

# Evaluación del modelo
loss, accuracy = model.evaluate(X_test, y_test)
print('Precision: %.2f' % (accuracy*100))
```

```
12/12 [==============================] - 0s 2ms/step - loss: 1.0567 - accuracy: 0.6715 - val_loss: 0.8829 - val_accu
racy: 0.8500
Epoch 196/200
12/12 [==============================] - 0s 2ms/step - loss: 1.0785 - accuracy: 0.6583 - val_loss: 0.8766 - val_accu
racy: 0.8562
Epoch 197/200
12/12 [==============================] - 0s 2ms/step - loss: 1.0540 - accuracy: 0.6868 - val_loss: 0.8710 - val_accu
racy: 0.8562
Epoch 198/200
12/12 [==============================] - 0s 2ms/step - loss: 1.0498 - accuracy: 0.6819 - val_loss: 0.8669 - val_accu
racy: 0.8500
Epoch 199/200
12/12 [==============================] - 0s 2ms/step - loss: 1.0525 - accuracy: 0.6826 - val_loss: 0.8638 - val_accu
racy: 0.8562
Epoch 200/200
12/12 [==============================] - 0s 2ms/step - loss: 1.0414 - accuracy: 0.6896 - val_loss: 0.8598 - val_accu
racy: 0.8562
13/13 [==============================] - 0s 561us/step - loss: 0.8204 - accuracy: 0.8775
Precision: 87.75
```

Figura 13: Primer modelo de red neuronal

Con la finalidad de incrementar la precisión dado que el modelo anterior arrojó una precisión de casi 97.25 %, decidí modificar algunos parámetros; utilicé sparse_categorical_crossentropy, también decidí utilizar StratifiedKFold, este modelo incrementó 85.5 % la precisión como se puede observar en la siguiente figura:

```
def create_model():
    model = Sequential()
    model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.01)))
    model.add(Dropout(0.5))
    model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.01)))
    model.add(Dropout(0.5))
    model.add(Dense(4, activation='softmax'))
    optimizer = Adam(learning_rate=0.0001)
    model.compile(loss='sparse_categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model

data = pd.read_csv("train.csv")
scaler = StandardScaler()
X = scaler.fit_transform(data.drop("price_range", axis=1))
y = data["price_range"]

# Crear el modelo
model = KerasClassifier(build_fn=create_model, epochs=200, batch_size=128, verbose=0)

# Validación cruzada
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
results = cross_val_score(model, X, y, cv=kfold)

print("Precisión promedio: %.2f%%" % (results.mean()*100))
```

```
/tmp/ipykernel_122845/1950703141.py:20: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras
thub.com/adriangb/scikeras) instead. See https://www.adriangb.com/scikeras/stable/migration.html for help
  model = KerasClassifier(build_fn=create_model, epochs=200, batch_size=128, verbose=0)

Precisión promedio: 85.50%
```

Figura 14: Segundo modelo de red neuronal

Animado pero insatisfecho por el resultado decidí utilizar otro conjunto de pasos a fin de incrementar la precisión, tales como añadir más capas, modificar el learning_rate, incrementar el número de epochs, entre otros y el resultado de ésta forma incrementó al 94.45 % como se observa en la siguiente Figura

```
def create_model():
    model = Sequential()
    model.add(Dense(256, input_dim=X_train.shape[1], activation='relu'))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.01)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.01)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.01)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.01)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(16, activation='relu', kernel_regularizer=l2(0.01)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))
    model.add(Dense(4, activation='softmax'))
    optimizer = Adam(learning_rate=0.0005)
    model.compile(loss='sparse_categorical_crossentropy', optimizer=optimize
    return model

data = pd.read_csv("train.csv")
scaler = StandardScaler()
X = scaler.fit_transform(data.drop("price_range", axis=1))
y = data["price_range"]

# Validación cruzada
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Crear el modelo
model = KerasClassifier(build_fn=create_model, epochs=1000, batch_size=64, v

results = cross_val_score(model, X, y, cv=kfold)

print("Precisión promedio: %.2f%%" % (results.mean()*100))
```

```
/tmp/ipykernel_122845/4181710883.py:35: DeprecationWarning: KerasClassifier is d
thub.com/adriangb/scikeras) instead. See https://www.adriangb.com/scikeras/stabl
  model = KerasClassifier(build_fn=create_model, epochs=1000, batch_size=64, ver

Precisión promedio: 94.45%
```

Figura 15: Tercer modelo de red neuronal

Como último paso decidí jugar con los parámetros a modo que incrementará la eficiencia de forma empírica, el resultado fue tardado y en muchas ocasiones disminuía, así que me enfoque en el calculo de overfitting, realicé una figura que muestra el comportamiento entre Loss vs Epoch. La siguiente Figura ilustra el resultando, se tiene que para aproximadamente 150 a 200 epoch la curva deja de disminuir o bien, el modelo se sobre entrena.

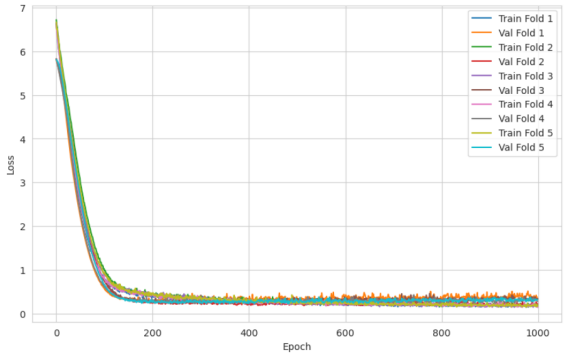

Figura 16: Curva Loss vs Epoch

4. CONCLUSIONES

En el presente trabajo de investigación se detallaron que son las SVM y las Redes neuronales, además se describieron los tipos de variables y análisis de datos del dataset *"Mobile Price Classification"* así como una matriz de correlación. Para el modelo antes mencionado como primer paso se realizó un modelo de SVM el cuál consiguió un porcentaje bastante alto de 97 %, a fin de incrementar la eficiencia se diseño un modelo de iteración de hiperparámetros consiguiendo así un 97.25 %, es decir, casi no aumentó. A fin de entender cómo funcionaban dichos hiperparámetros se realizaron varias graficas que describen hasta cuánto es bueno aumentar o disminuir C, gamma o el tipo de kernel, esto me llevó a descubrir cómo utilizarlos de forma bruta para mejores resultados sin iterar de forma intuitiva.

Como segunda parte se diseñaron varios modelos de Redes neuronales siendo el primero modelo con una precisión de tan solo el 87.75 % lo cuál es muy bajo en comparación con SVM, se modificaron varios parámetros de forma empírica mediante añadir más capas, modificar las Epoch's, entrenar con otro rango de aprendizaje, entre otros. Los resultados fueron tardados pero se llego a obtener un 94.45 % de precesión y a forma de no sobrepasar los tiempos se programó una curva de overfitting, siendo de 150 a 200 Epoch's lo óptimo para reducción de tiempo y evitar sobre entrenamiento.
En conclusión ambos modelos funcionan con una alta precisión, por un lado un modelo SVM es un poco más sencillo y con alta precisión, siendo los hiperparámetros fundamentales para obtener un excelente resultado. Por otro lado los modelos de redes neuronales son más complejos y necesitan ser entrenados bajo diferentes condiciones pero para casos en que se tengan problemas estocásticos, de muchas variables y de gran complejidad son una herramienta muy fuerte y precisa.

REFERENCIAS

[1] S. Russell and P. Norvig, .ªrtificial Intelligence: A Modern Approach,"3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2009.

[2] J. Brownlee, "What Is Machine Learning? A Definition,"Machine Learning Mastery, Jan. 2019. doi: 10.1007/s13398-014-0173-7.

[3] T. M. Mitchell, "Machine Learning,ÏEEE Magazine, vol. 17, no. 4, pp. 22-27, Jul. 1994. doi: 10.1109/6.298385.

[4] C. M. Bishop, "Pattern Recognition and Machine Learning,"Springer, 2006. doi: 10.1007/978-0-387-31073-2.

[5] Cortes, C., Vapnik, V. (1995). Support-vector networks. Machine learning, 20(3), 273-297. doi: 10.1007/BF00994018

[6] Keerthi, S. S., Lin, C. J. (2003). Asymptotic behaviors of support vector machines with Gaussian kernel. Neural computation, 15(7), 1667-1689. doi: 10.1162/089976603321891855

[7] Schölkopf, B., Smola, A. (2002). Learning with kernels: Support vector machines, regularization, optimization, and beyond. MIT press. doi: 10.1007/978-1-4757-3264-1

[8] Chang, C. C., Lin, C. J. (2011). LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST), 2(3), 27. doi: 10.1145/1961189.1961199

[9] McCulloch, W.S., Pitts, W. .ª Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics, vol. 5, pp. 115–133, 1943. https://doi.org/10.1007/BF02478259

[10] Bishop, C. M. (1995). Neural networks for pattern recognition. Oxford university press.

[11] Goodfellow, I., Bengio, Y., Courville, A. (2016). Deep learning. MIT Press.

[12] Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533-536.

[13] Oscar Javier Reyes-Ortiz ,Marcela Mejia,

Asignatura	Datos del Equipo	Fecha
Aprendizaje Automático	Nombre: Gabriel	06/03/2023
	Apellidos: Garcia Zambrano	

Juan Sebastián Useche-Castelblanco, "TÉCNICAS DE INTELIGENCIA ARTIFICIAL UTILIZADAS EN EL PROCESAMIENTO DE IMÁGENES Y SU APLICACIÓN EN EL ANÁLISIS DE PAVIMENTOS", Revista EIA, vol. 16, núm. 31, pp. 189-207, 2019 .Escuela de Ingeniería de Antioquia