Grupo 3: Oscar Andrés Patiño Patarroyo - Código 202223234, Andrés Sierra Urrego - Código 202120960, John Edinson Rodríguez Fajardo - Código 202226240

Mini proyecto: Predicción año lanzamiento de canciones

GitHub del miniproyecto: https://github.com/oapatinop/DL_miniproyecto

1 Exploración de los datos

Exploración de los datos para su entendimiento dentro del contexto organizacional: Se utilizan histogramas, correlogramas y estadísticas descriptivas para la exploración preliminar de los datos del problema, para dejar claro el rol que cumple cada una de las variables a utilizar. Además, se argumenta la razón por la que el uso de estas variables puede contribuir a la solución del problema por medio de modelos predictivos. [10 puntos]

Se inicia el análisis exploratorio de datos presentando una tabla con algunas métricas descriptivas de las variables y las dimensiones del dataset.

	count	mean	std	min	25%	50%	75%	max
Y	77779.0	2002.308129	10.811038	1926.00000	1998.000000	2006.00000	2010.000000	2014.00000
V1	77779.0	43.425185	6.128869	4.83688	40.060315	44.32385	47.900080	60.03401
V2	77779.0	-0.136720	4.370466	-69.68087	-2.612435	-0.06300	2.465950	23.81526
V4	77779.0	3.755313	17.609183	-165.22161	-7.016240	2.02210	12.776450	274.65858
V5	77779.0	-2.339768	14.483975	-121.47534	-10.685075	-2.05456	6.423900	160.81522
...	...	...	...	...	...	...	...	...
V86	77779.0	17.575097	115.243644	-3168.92457	-31.569390	15.23736	67.361475	2144.10391
V87	77779.0	-25.628857	173.310304	-4319.99232	-100.668480	-21.58164	51.333150	2833.60895
V88	77779.0	4.463484	13.526414	-236.03926	-2.569360	3.13690	10.002145	275.35366
V89	77779.0	18.664885	186.690183	-7458.37815	-60.121980	5.94149	84.389125	5289.11138
V90	77779.0	1.240194	22.379654	-281.15060	-8.900120	-0.09534	9.520700	600.76624

90 rows × 8 columns

```
 Dimensiones del conjunto de datos
(77779, 90)

 Rango de la media de los variables
-191.30189603273377 2419.778632412219

 Variable con la media más pequeña
{'V39': -191.30189603273377}

 Variable con la media más grande
{'V14': 2419.778632412219}
```

Figura 1. Métricas descriptivas.

El resultado anterior permite ver que las dimensiones del conjunto de datos son de 77779 filas y 90 columnas, donde la variable de respuesta la representa la variable Y, y las demás variables, que

empiezan por la letra V, son las variables predictoras. Por otro lado, a partir del resultado de las métricas descriptivas, se observa que el rango de la media de las variables del conjunto de datos está entre -191 y 2419, donde el primer valor corresponde a la variable V39 y el segundo a V14.

Luego de mencionar algunos resultados que se obtuvieron de las métricas descriptivas, se presentarán los histogramas y diagramas de cajas de algunas variables.

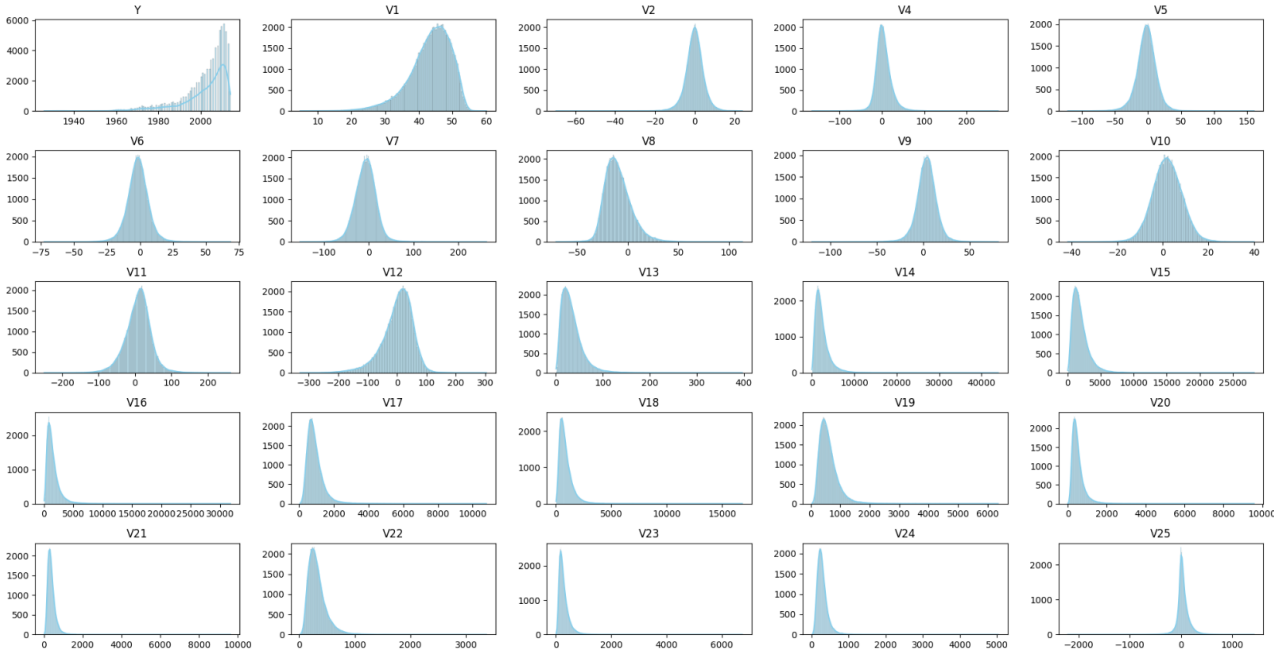

Figura 2. Histogramas de algunas variables.

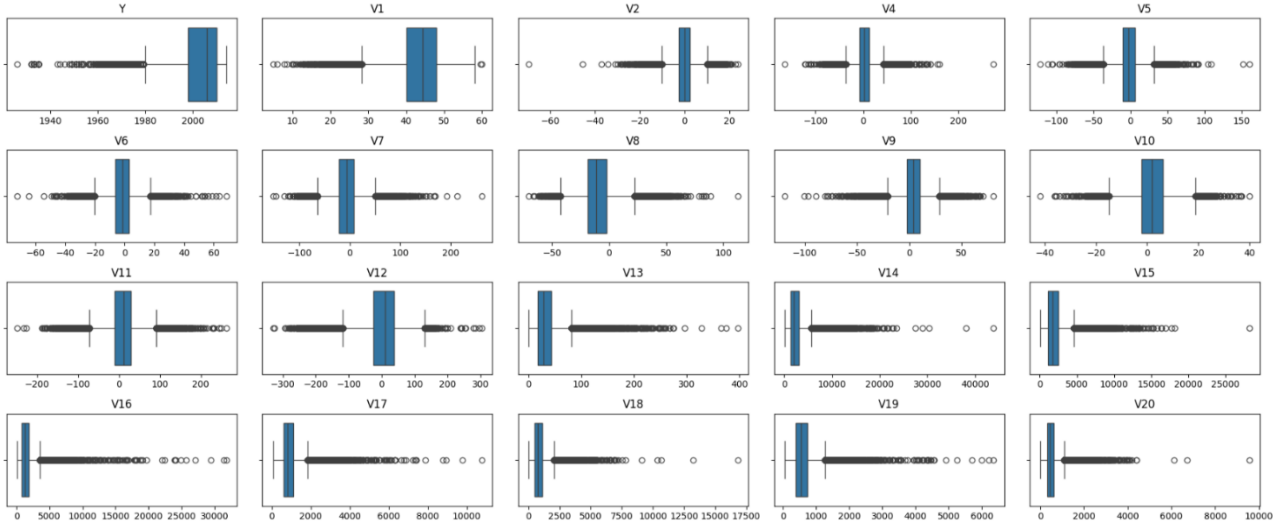

Figura 3. Diagramas de cajas de algunas métricas descriptivas.

Los histogramas anteriores reflejan que hay variables que muestran asimetrías positivas y negativas en los datos y unas cuantas presentan una distribución similar a una normal. Por otro lado, los diagramas de cajas permiten concluir que todas las variables muestran outliers o valores atípicos en sus datos.

Para finalizar con el análisis exploratorio de los datos, se presentarán solo los pares de variables que tuvieron correlación fuerte (se decidió omitir la matriz de correlación debido a su complejidad visual).

```
Pares de variables con fuerte correlación:
Pareja: V16 - V18, Correlación: 0.81
Pareja: V16 - V20, Correlación: 0.73
Pareja: V16 - V23, Correlación: 0.84
Pareja: V18 - V22, Correlación: 0.72
Pareja: V18 - V23, Correlación: 0.86
Pareja: V20 - V22, Correlación: 0.87
Pareja: V20 - V23, Correlación: 0.71
Pareja: V22 - V23, Correlación: 0.72

Total de pares de variables: 8
```

Figura 4. Pares de variables con fuerte correlación.

El resultado arrojado permite apreciar 8 pares de variables con fuerte correlación, donde todas muestran una alta correlación positiva, siendo la pareja V20 y V22 las que muestran la mayor correlación con un valor de 0,87.

2 Preparación de datos

Para preparar los datos, se empieza definiendo la variable de respuesta y variables las predictoras, donde la primera la representa la variable Y, mientras que las segundas por todas las que empiezan con V. Tras definir la variable dependiente e independientes, se normalizan estas últimas para que estén en una misma escala.

```
XTotal = dTr.loc[:,dTr.columns!='Y']
yTotal = dTr.loc[:,dTr.columns=='Y']

scaler = StandardScaler(with_mean=True, with_std=True)
scaler.fit(XTotal)
XTotalS=scaler.transform(XTotal)
dTestS=scaler.transform(dataTesting)
```

Figura 5. Definición de variables de respuesta y predictoras, y normalización.

Luego de definir las variables de respuesta y predictoras, y normalizar estas últimas, se divide el conjunto de datos en un set de entrenamiento y de prueba, donde se estableció un tamaño de muestra en el 30% y el 70% en un set de entrenamiento.

```
XTrain, XTest, yTrain, yTest = train_test_split(XTotalS, yTotal, test_size=0.30, random_state=3)
```

Figura 6. División del conjunto de datos en un set de train y test.

Vale la pena resaltar que antes de realizar los pasos anteriores, se verificó la existencia de valores nulos en el conjunto de datos y no evidenció la presencia de ellos.

Por otro lado, pese a que la variable de respuesta es numérica, la convertimos en categórica para abordar el problema del proyecto con un modelo de clasificación. Por lo tanto, a continuación, se presenta la forma cómo dicho preprocesamiento de datos:

```
a = XTotal.min()
b = XTotal.max()
X = (XTotal - a) / (b - a)
Y = yTotal - yTotal.min()

XTrain, XTest, yTrain, yTest = train_test_split(XTotalS, Y, test_size=0.30, random_state=3)

from keras.utils import to_categorical
Y_train = to_categorical(yTrain, 89)
Y_test = to_categorical(yTest, 89)
```

Figura 7. Preprocesamiento de datos para problema de clasificación.

En este nuevo preprocesamiento de datos, realizamos una normalización **min-max** en los datos de las variables predictoras, mientras que para la variable de respuesta realizamos la diferencia entre cada uno de los valores por el valor mínimo de dicha variable. Luego, partimos el conjunto de datos previamente preprocesado en un set de entrenamiento y de prueba, definiendo un tamaño de prueba del 30%. Finalmente, con ayuda de la librería de **keras.utils** convertimos la variable de respuesta en categórica.

3 Análisis preliminar de modelos

En el análisis preliminar de selección de modelos relevantes para predecir el año de las canciones con las 90 variables independientes, se identifican varias opciones que se plantearon para el miniproyecto. Para abordar este problema con redes neuronales, se decidió emplear Keras para construir y entrenar los modelos de redes neuronales de manera eficiente. Utilizando Keras, se pueden tener más flexibilidad en la estructura de las redes neuronales personalizadas con capas densamente conectadas, funciones de activación y pérdida adecuadas para este problema de regresión. Además, se puede utilizar el algoritmo de optimización Adam para ajustar los parámetros del modelo de manera iterativa.

Por otro lado, se decidió tener como opción Scikit-learn con su paquete de redes neuronales y específicamente el algoritmo de MLPRegressor, esto teniendo en cuenta la facilidad de implementación y calibración que se tiene con Scikit-learn, aunque se sacrifica en flexibilidad de la estructura y opciones de calibración que si se tienen con keras, es util para encontrar opciones para la predicción e incluso para incorporarlas en las redes estructuradas con keras.

Además de los enfoques basados en redes neuronales de regresión, también se puede considerar el uso de una red neuronal desde el enfoque de clasificación, por lo cual se plantea implementar un modelo de clasificación con redes neuronales con keras, esto para comparar si es mejor plantear el problema para su desarrollo y calibración como un problema de regresión o de clasificación.

Para este análisis preliminar se puede ver en el notebook del github dado al comienzo del documento, sin embargo acá se muestra el código de implementación de los algoritmos planteados.

- Red neuronal de regresión con keras:

```
# Estructura de la red
knn2 = Sequential()
knn2.add(Dense(128, activation='relu', input_shape=(XTrain.shape[1],)))
knn2.add(BatchNormalization())
knn2.add(Dropout(0.2))
knn2.add(Dense(128, activation='relu'))
```

```
knn2.add(BatchNormalization())
knn2.add(Dropout(0.2))
knn2.add(Dense(128, activation='relu'))
knn2.add(Dense(1))
# Compilación del modelo
opt = Adam(learning_rate=0.005)
knn2.compile(optimizer=opt, loss='mean_squared_error')
# Entrenamiento del modelo
knn2.fit(XTrain, yTrain.to_numpy(), epochs=125, batch_size=128)
```

- Red neuronal de regresión con Scikit-learn:

```
sklnn1 = MLPRegressor(hidden_layer_sizes=(128, 128, 128, 128), activation='logistic', solver='lbfgs', learning_rate_init = 0.001, max_iter=250, random_state=3)
sklnn1.fit(XTrain, yTrain)
sklNN1 = sklnn1.predict(XTest)
sklNN1Tr = sklnn1.predict(XTrain)
metrics_sklNN1 = calculo_rmse(sklNN1, yTest, sklNN1Tr, yTrain)
```

- Red neuronal de clasificación con keras: para esta red se realizó una transformación adicional a la variable “y”.

```
# Transformación de las y
from keras.utils import to_categorical
Y_train = to_categorical(yTrain, 89)
Y_test = to_categorical(yTest, 89)
# Second Keras Model
model2 = Sequential()
model2.add(Dense(180, input_shape=(89,)))
model2.add(Dropout(0.2))
model2.add(Dense(360, activation='sigmoid'))
model2.add(Dropout(0.2))
model2.add(Dense(360, activation='sigmoid'))
model2.add(Dropout(0.2))
model2.add(Dense(89, activation='sigmoid'))
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
h = model2.fit( XTrain, Y_train, batch_size=128, epochs=50, validation_data=(XTest, Y_test),)
```

4 Desarrollo y calibración

Para la calibración se decidió usar GridSearch para buscar la mejor combinación de hiper-parámetros para las redes neuronales. Estas pruebas se incluyeron en el notebook que se encuentra en el github dado al comienzo, sin embargo acá se incluyen los códigos usados así como los resultados que se encontraron de los mejores hiperparámetros evaluados.

- Red neuronal de regresión con keras: Para la calibración del modelo de regresión con redes neuronales con keras, implementamos un GridSearch con ayuda de la librería de keras "wrappers", que genera el puente para poder usar GridSearchCV de Scikir Learn para buscar los mejores parámetros de la red neuronal, en el cual se busco combinaciones de capas ocultas, épocas, función de optimización y batch size. Con los hiperparámetros encontrados que arrojan el mejor MSE es con el que realizamos el entrenamiento.

```
# Modelo
def build_keras_base(hidden_layers = [512, 512], optimizer='sgd'):
    model_opt = Sequential()
    model_opt.add(Dense(hidden_layers[0], activation='relu', input_shape=(XTrain.shape[1],)))
    model_opt.add(Dropout(0.2))
    model_opt.add(Dense(hidden_layers[1], activation='relu', input_shape=(XTrain.shape[1],)))
    model_opt.add(Dense(1))
    model_opt.compile(optimizer = optimizer, loss = 'mean_squared_error', metrics = ['mean_squared_error'])
    return model_opt
model_keras = KerasRegressor(build_fn = build_keras_base, verbose=0)
# Opciones de parámetros
hidden_layers_opt=[[128,128], [254, 254]]
batch_size_opt= [128, 256]
epochs_opt= [20, 50]
optimizer= ['rmsprop', 'adam']
# Parámetros modelo
parameters_opt = {'optimizer': optimizer,'batch_size': batch_size_opt, 'epochs': epochs_opt}
# Grid Search
grid_search = GridSearchCV(estimator=model_keras, param_grid =parameters_opt, scoring = 'neg_root_mean_squared_error', cv = 3)
grid_result = grid_search.fit(XTrain, yTrain)
```

Con el código anterior encontramos que lo mejor es usar 128 neuronas, 20 épocas y el optimizador Adam.

- Red neuronal de regresión con Scikit-learn: Para la calibración del modelo de regresión con redes neuronales con SK-Learn, implementamos un GridSearchCV para buscar los mejores parámetros de la red nueronal, en el cual se buscó combinaciones de capas ocultas, tasa de aprendizaje, función de optimización y de activación. Con los hiperparámetros encontrados que arrojan el mejor MSE es con el que realizamos el entrenamiento.

```
grid_param_MLP = {'hidden_layer_sizes': [(64), (64,64), (64,64,64)],'activation': ['relu', 'logistics', 'identity'],'solver': ['lbfgs', 'adam'],'learning_rate_init': [0.001, 0.01, 0.1]}

sklnNN = MLPRegressor(random_state=3)

grid_GBR = GridSearchCV(estimator=sklnNN, param_grid = grid_param_MLP, scoring = 'neg_root_mean_squared_error', cv = 3, n_jobs=-1)

grid_GBR.fit(XTrain, yTrain)

print("Mejor: %f usando %s" % (grid_GBR.best_score_, grid_GBR.best_params_))
```

Del resultado anterior se encontró que el mejor MSE se obtiene con una red con 3 capas ocultas de 64 nodos, tasa de aprendizaje 0.001, con optimización lbfg y activación relu.

- Red neuronal de clasificación con keras: Por último para el modelo de clasificación la calibración del modelo con keras, implementamos al igual que en el de regresión un GridSearch con ayuda de la librería de keras “wrappers”, para buscar los mejores parámetros de la red neuronal, en el cual se buscó combinaciones de capas ocultas, épocas, función de optimización y batch size. Con los hiperparámetros encontrados que arrojan el mejor MSE es con el que realizamos el entrenamiento.

```
def build_keras_base(hidden_layers = [128, 128, 128], optimizer='sgd'):
    model_opt = Sequential()
    model_opt.add(Dense(hidden_layers[0], activation='relu', input_shape=(XTrain.shape[1],)))
    model_opt.add(Dropout(0.2))
    model_opt.add(Dense(hidden_layers[1], activation='relu'))
    model_opt.add(Dropout(0.2))
    model_opt.add(Dense(hidden_layers[2], activation='sigmoid'))
    model_opt.add(Dropout(0.2))
    model_opt.add(Dense(89, activation='sigmoid'))
    model_opt.compile(optimizer = optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
    return model_opt
model_keras = KerasClassifier(build_fn = build_keras_base, verbose=0)
# Opciones de parámetros
hidden_layers_opt=[[128,128, 128], [64, 64, 64]]
batch_size_opt= [128, 256]
epochs_opt= [20, 50]
optimizer= ['rmsprop', 'adam']
# Parámetros modelo
parameters_opt = {'optimizer': optimizer,'batch_size': batch_size_opt, 'epochs': epochs_opt}
# Grid Search
grid_search = GridSearchCV(estimator=model_keras, param_grid =parameters_opt, scoring = 'accuracy', cv = 3)
grid_result = grid_search.fit(XTrain, Y_train)
```

Con este se obtuvo lo siguiente; 'batch_size': 128, 'epochs': 50, 'optimizer': 'adam'

5 Visualización de resultados

De la evaluación de los hiperparámetros en la cual se buscó un buen desempeño de la red neuronal, evaluando combinaciones de capas ocultas, tasa de aprendizaje, función de optimización y de activación se seleccionaron algunos modelos de los cuales se presenta la información general en el anexo en la **tabla 6.1**. En la tabla también se incluye un modelo de clasificación utilizando la librería de Keras. De los datos de prueba se obtuvieron los resultados de la métrica de desempeño definida como el RMSE para cada uno de los modelos, en la **figura 8** se presenta la comparación del RMSE para cada modelo.

Según los resultados de desempeño de los modelos, se evidencia que el modelo **keras3** presentó el menor RMSE, lo que indica que las predicciones del modelo se acercan más a los valores reales. La red neuronal se definió con la librería de Keras con 3 capas ocultas,128 neuronas en cada capa oculta, una función de activación relu, el optimizador Nadam con una tasa de aprendizaje de 0.005, normalización por lotes y regulización Dropout con una tasa de abandono del 0.15. El desempeño del modelo posiblemente esté influenciado por el algoritmo de optimización seleccionado para entrenar el modelo de red neuronal, que en este caso fue Nadam el cual es una variante del optimizador Adam (Adaptive Moment Estimation), que combina las ideas de dos algoritmos de optimización populares: RMSprop y Momentum. La principal diferencia entre Adam y Nadam es que Nadam utiliza Nesterov momentum, lo que probablemente ayudó a converger más rápidamente hacia el mínimo global.

En la **figura 8** también se evidenció que el modelo **sklearn1** obtuvo un desempeño muy similar al modelo anterior. La red neuronal se definió con la librería de sklearn con 4 capas ocultas,128 neuronas en cada capa oculta, una función de activación logistic (que es la función sigmoide), el optimizador lbfgs y un número máximo de iteraciones igual que 250. Este modelo al tener más capas ocultas puede aumentar la sensibilidad y el optimizador lbfgs tiene la ventaja de eficiencia en memoria, buena convergencia y puede funcionar bien en problemas no convexos. Por otro lado, el valor del RMSE en los demás modelos estuvo por encima de 9.44.

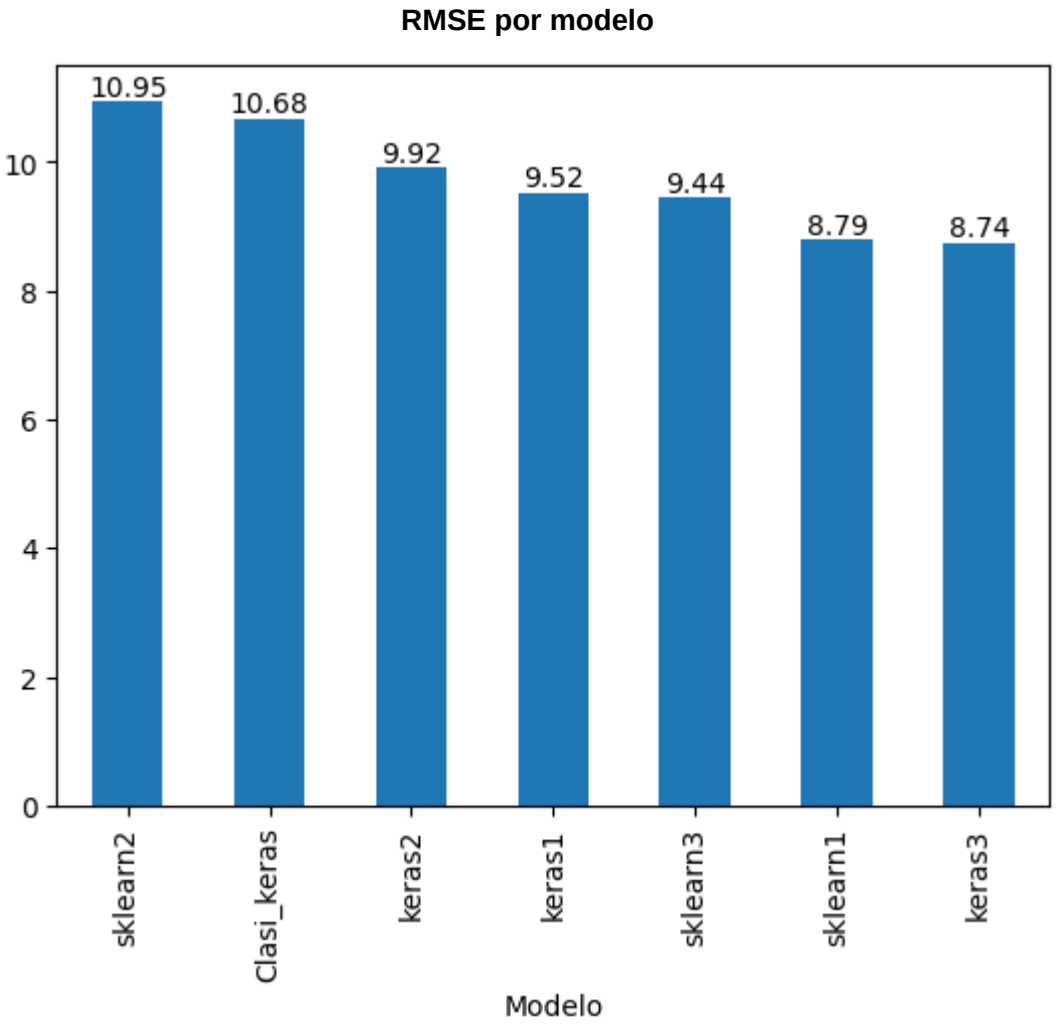

Figura 8. RMSE por modelo.

6 Anexo

6.1 Tabla, Información general modelos con mejor desempeño

Modelo	RMSE	Información general de los parámetros del modelo

Keras1	9,5216	# Definición del modelo
		knn1 = Sequential()
		knn1.add(Dense(64, input_dim=XTrain.shape[1], activation='sigmoid'))
		knn1.add(BatchNormalization(momentum=0.9))
		knn1.add(Dropout(0.2))
		knn1.add(Dense(64, activation='relu'))
		knn1.add(Dense(1))
		# Compilación del modelo
		knn1.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_squared_error'])

Keras2	9,9198	# Definición del modelo
		knn2 = Sequential()
		knn2.add(Dense(128, activation='relu', input_shape=(XTrain.shape[1],)))
		knn2.add(BatchNormalization())
		knn2.add(Dropout(0.2))
		knn2.add(Dense(128, activation='relu'))
		knn2.add(BatchNormalization())
		knn2.add(Dropout(0.2))
		knn2.add(Dense(128, activation='relu'))
		knn2.add(Dense(1))
		# Compilación del modelo
		opt = Adam(learning_rate=0.005)
		knn2.compile(optimizer=opt, loss='mean_squared_error')

Keras3	8,7423	# Definición del modelo
		knn3 = Sequential()
		knn3.add(Dense(128, activation='relu', input_shape=(XTrain.shape[1],)))
		knn3.add(BatchNormalization())
		knn3.add(Dropout(0.15))
		knn3.add(Dense(128, activation='relu'))
		knn3.add(BatchNormalization())
		knn3.add(Dropout(0.15))
		knn3.add(Dense(128, activation='relu'))
		knn3.add(Dense(1))
		# Compilación del modelo
		opt = Nadam(learning_rate=0.005)
		knn3.compile(optimizer=opt, loss='mean_squared_error', metrics=['mean_squared_error'])

Clasificación con keras	10,6754	# Definición del modelo
		model2 = Sequential()
		model2.add(Dense(180, input_shape=(89,)))
		model2.add(Dropout(0.2))
		model2.add(Dense(360, activation='sigmoid'))
		model2.add(Dropout(0.2))
		model2.add(Dense(360, activation='sigmoid'))
		model2.add(Dropout(0.2))
		model2.add(Dense(89, activation='sigmoid'))
		# Compilación del modelo
		model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
		# Entrenamiento del modelo
		h = model2.fit(XTrain, Y_train, batch_size=128, epochs=50, validation_data=(XTest, Y_test),)

sklearn1	8,7895	# Definición parámetros del modelo
		sklnn1 = MLPRegressor(hidden_layer_sizes=(128, 128, 128, 128), activation='logistic', solver='lbfgs', learning_rate_init = 0.001, max_iter=250, random_state=3)

sklearn2	10,9466	# Definición parámetros del modelo
		sklnn1 = MLPRegressor(hidden_layer_sizes=(128, 128, 128, 128), activation='relu', solver='adam', learning_rate_init = 0.001, max_iter=250, random_state=3)

sklearn3	9,436	# Definición parámetros del modelo
		sklnGS = MLPRegressor(hidden_layer_sizes=(128, 64, 32, 16), activation='relu', solver='lbfgs', batch_size=128, learning_rate_init = 0.0001, random_state=3)