Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ingeniería Eléctrica

Proyecto EL4106

Clasificador de actividades físicas

Integrantes: Vicente Pinochet R.
Profesor: Javier Ruiz del Solar
Auxiliar: Patricio Loncomilla Z.
Ayudantes: Benjamín Llancao Mella
José Luis Antonio Cadiz S.
Sebastián Tinoco
Vicente Bórquez Z.

Fecha de realización: 4 de noviembre de 2023
Fecha de entrega: 19 de julio de 2023
Santiago de Chile

Índice de Contenidos

Índice de Figuras

1. Introducción

En este proyecto, se plantea el objetivo de implementar un clasificador de actividades físicas utilizando una versión modificada de la base de datos Human Activity Recognition Using Smartphones Data Set. Esta base de datos, contiene mediciones de personas realizando diversas actividades físicas, capturadas mediante una unidad de mediciones inerciales (IMU) de un Samsung Galaxy S II. La IMU incluye un acelerómetro 3D y un giroscopio de 3 ejes, lo que permite obtener una variedad de señales.

El conjunto de datos original consta de mediciones para seis actividades físicas distintas: caminar, subir escaleras caminando, bajar escaleras caminando, estar sentado, estar parado y estar acostado. A partir de estas señales, se calculan nueve características finales mediante la aplicación de un filtro a las señales del acelerómetro y el giroscopio.

El desafío consiste en diseñar, implementar, entrenar y validar un sistema de clasificación que pueda recibir las señales de entrada y determinar la acción realizada, es decir, la clase de actividad física correspondiente.

2. Metodologia

2.1. Carga de la base de datos

Se importan las bibliotecas necesarias, como 'pandas' para el manejo de los datos en forma de DataFrames y 'matplotlib.pyplot' para visualización de datos, así como la biblioteca 'files' para cargar el archivo de la base de datos desde el entorno de Google Colab.

Se utiliza el código para cargar la base de datos desde el archivo 'dataset_2023_full_alumnos_v01.zip'. El archivo se carga mediante la función 'files.upload()' y se descomprime utilizando '!unzip dataset_2023_full_alumnos_v01.zip'.

Se leen los archivos CSV que contienen las diferentes señales de las actividades físicas, tanto para el conjunto de entrenamiento, validación y prueba. Cada archivo se lee utilizando la función 'pd.read_csv()' de la biblioteca pandas, y se almacena en un DataFrame correspondiente. Se utilizan variables con rutas de archivo predefinidas para cada una de las señales de entrada.

Se realiza un preprocesamiento de los datos para asegurar que los DataFrames tengan el formato adecuado. Se elimina la primera fila de cada DataFrame utilizando la función 'iloc[1:]' para eliminar las etiquetas de las columnas. Esto se realiza para todos los DataFrames correspondientes a las diferentes señales de entrada.

Se leen los archivos CSV que contienen las etiquetas correspondientes a las actividades físicas para el conjunto de entrenamiento y validación. Estas etiquetas se almacenan en un DataFrame llamado 'df_etiquetas' y 'df_etiquetas_validacion', respectivamente.
Se crea un diccionario llamado 'actividades' que mapea los números de etiquetas a las actividades físicas correspondientes. Esto se utilizará más adelante para traducir los números de etiquetas a nombres de actividades.

2.2. Visualización de señales del conjunto de datos

Se selecciona cada actividad específica para visualizar. Esto se hace mediante la identificación de los índices en el DataFrame de etiquetas que corresponden a la actividad específica de interés.
Luego se crea una figura y un conjunto de ejes utilizando la función 'plt.subplots()'.

Utilizando un bucle, se itera sobre los índices obtenidos en el primer paso. Para cada índice, se accede a la señal correspondiente en el DataFrame de la característica seleccionada (por ejemplo, 'df_body_acc_x') y se grafica utilizando la función 'ax.plot()'. Esto se repite para cada una de las nueve señales disponibles.
Finalmente, se muestra el gráfico utilizando la función 'plt.show()'.

Se repiten los pasos para cada una de las nueve señales disponibles y cada una de las seis actividades físicas. Esto permitirá visualizar las señales de todas las actividades en el conjunto de datos.

2.3. Diseño del filtros

Mediante pruebas se seleccionaron las siguientes señales: 'df_body_acc_y', 'df_body_gyro_x', 'df_body_gyro_z', 'df_body_total_x', 'df_body_total_y' y 'df_body_total_z' como las señales relevantes para la solucion de este problema.

Se definen los parámetros para un filtro pasa alta y dos pasa baja, como la frecuencia de corte, el orden del filtro y la frecuencia de muestreo. En el código proporcionado, se utiliza la función 'signal.butter()' para diseñar los filtros.

Se utiliza la función 'signal.lfilter()' para aplicar el filtro a cada una de las señales de interés. En el código proporcionado, se muestra cómo se aplica el filtro pasa alta a la señal 'df_body_acc_y', y los filtros pasa baja a las señales 'df_body_gyro_x', 'df_body_gyro_z', 'df_body_total_x', 'df_body_total_y' y 'df_body_total_z'.

Para asegurar una mejor comparabilidad y estandarización de las señales, se aplico una normalización a los datos, se utilizo la clase 'StandardScaler' de la biblioteca 'sklearn.preprocessing' para normalizar las señales.

2.4. Función de cálculo de características

Se define una función llamada 'calcular_caracteristicas' que toma como entrada una señal y devuelve un DataFrame que contiene las características calculadas. Las características utilizadas son: la media, varianza, cruzamiento de cero y asimetría.

La función 'calcular_caracteristicas' retorna el DataFrame que contiene todas las características calculadas. Esto permite utilizar el DataFrame en las señalas a utilizar para calcular sus características para luego utilizarlas en el sistema de clasificación de actividades físicas.

2.5. Selección de caracteristicas

Se combinan las características calculadas para las seis señales. Ademas se agrega un sufijo a las columnas para identificar el eje correspondiente.

Se utiliza la función 'pd.concat()' para combinar todas las características en un solo DataFrame llamado 'caracteristicas_combinadas'. Se especifica el eje de concatenación como 1 para combinar las características en forma de columnas.
Se obtienen las etiquetas correspondientes a las actividades físicas del DataFrame de etiquetas. En el código proporcionado, se utiliza la columna 'Category' como etiquetas.

Se utiliza el método de selección de características 'SelectKBest' con la función de puntuación 'f_classif' para realizar la selección. Se especifica el parámetro 'k' para indicar el número deseado de características seleccionadas. En donde se eligió seleccionar 22 características.

Se utiliza la función 'fit_transform()' del objeto 'seleccionador' para obtener las características se-

leccionadas a partir del DataFrame 'caracteristicas_combinadas'. Esto devuelve un nuevo DataFrame llamado 'caracteristicas_seleccionadas'.

2.6. Clustering

Se utiliza la clase 'PCA' de la biblioteca 'sklearn.decomposition' para realizar la reducción de dimensionalidad. La reducción de dimensionalidad se aplica al DataFrame de características seleccionadas, 'caracteristicas_seleccionadas_df'.

Se utiliza la función 'sns.scatterplot()' de la biblioteca 'seaborn' para visualizar los resultados de la reducción de dimensionalidad. En el código proporcionado, se muestra un gráfico de dispersión en el que se representan los puntos correspondientes a los componentes principales en el plano definido por 'Componente1' y 'Componente2'. Los puntos se colorean según la categoría de actividad física utilizando la columna 'Category' del DataFrame 'componentes_df'.

2.7. Seleccionar y entrenar clasificadores

Se aplica el mismo proceso a las señales de validacion, calculando y obteniendo las mismas caracteristicas que el conjunto de entrenamiento.

Se crean diversos clasificadores, dos Random forest dos redes neuronales y una SVM Polinomial.

2.7.1. Randon Forest

Se crea un clasificador utilizando la clase 'RandomForestClassifier' de la biblioteca 'sklearn.ensemble'.

Se utiliza el método 'fit()' del clasificador para entrenarlo utilizando las características seleccionadas y las etiquetas correspondientes. Se entrena el clasificador utilizando el DataFrame de características seleccionadas 'caracteristicas_seleccionadas_df' y las etiquetas 'etiquetas'.

Se utiliza el método 'predict()' del clasificador para realizar predicciones en el conjunto de validacion. Se utiliza la función 'classification_report()' de la biblioteca 'sklearn.metrics' para evaluar el rendimiento del clasificador en el conjunto de prueba. Se compara las etiquetas verdaderas 'etiquetas_validacion' con las predicciones realizadas 'y_pred_val' y se obtiene un informe de clasificación que muestra métricas como precisión, recall y F1-score.

Se utiliza la función 'confusion_matrix()' de la biblioteca 'sklearn.metrics' para obtener la matriz de confusión entre las etiquetas verdaderas y las predicciones. Luego, se normaliza la matriz de confusión dividiendo cada valor por la suma de los valores en esa fila. Se muestra la matriz de confusión normalizada utilizando la función 'heatmap()' de la biblioteca 'seaborn'. En el código proporcionado, se muestra cómo visualizar la matriz de confusión normalizada.

2.7.2. Red Neuronal

Se convierten las etiquetas 'etiquetas' y 'etiquetas_validacion' en arreglos de NumPy y se les resta 1 a cada etiqueta para ajustar el rango de las etiquetas a partir de 0. Se almacenan en las variables 'etiquetas_corregidas' y 'etiquetas_corregidas_val', respectivamente.

Se utiliza la API secuencial de Keras para definir la arquitectura de la red neuronal. En el código proporcionado, se crea una red neuronal con dos capas ocultas de 64 neuronas cada una y una capa de salida con 6 neuronas. Se utiliza la función de activación 'relu' en las capas ocultas y la función de activación 'softmax' en la capa de salida.

Se utiliza el método 'compile()' del modelo para compilar la red neuronal. Se especifica el optimizador 'adam', la función de pérdida 'sparse_categorical_crossentropy' (apropiada para problemas de clasificación con etiquetas enteras) y la métrica 'accuracy' para evaluar el rendimiento.

Se utiliza el método 'fit()' del modelo para entrenar la red neuronal. Se proporcionan las características seleccionadas 'caracteristicas_seleccionadas_df' y las etiquetas corregidas 'etiquetas_corregidas' como datos de entrenamiento. Se especifica el número de épocas y el tamaño del lote.

Se utiliza el método 'evaluate()' del modelo para evaluar el rendimiento del modelo en el conjunto de validación. Se proporcionan las características seleccionadas del conjunto de validación 'caracteristicas_seleccionadas_val' y las etiquetas corregidas del conjunto de validación 'etiquetas_corregidas_val' como datos de evaluación. Se almacena la puntuación obtenida en la variable 'puntuacion'.

Se utiliza el método 'predict()' del modelo para obtener las probabilidades de predicción en el conjunto de validación. Luego, se obtienen las etiquetas predichas tomando el índice del valor máximo en cada probabilidad. Se calcula la matriz de confusión entre las etiquetas verdaderas y las etiquetas predichas y se muestra la matriz de confusión normalizada utilizando la función 'heatmap()' de la biblioteca 'seaborn'. Por último, se calcula y muestra el accuracy del modelo en el conjunto de validación utilizando la función 'accuracy_score()' de la biblioteca 'sklearn.metrics'.

2.7.3. Red neuronal en PyTorch

Los conjuntos de entrenamiento y validación se convierten en tensores de PyTorch utilizando la clase torch.tensor(). Se definen los tensores inputs_train, labels_train, inputs_val, y labels_val a partir de las características seleccionadas y las etiquetas corregidas.

Se crea el modelo de red neuronal utilizando la clase nn.Sequential(). En el código proporcionado, se definen dos capas ocultas con funciones de activación 'ReLU' y 'Tanh', respectivamente. La última capa es la capa de salida con 6 neuronas.

Se configura el dispositivo de ejecución para utilizar la GPU mediante device = torch.device('cuda'). Esto acelerará el entrenamiento si se dispone de una GPU compatible.
Se define la función de pérdida utilizando nn.CrossEntropyLoss() y el optimizador utilizando optim.Adam().

Se itera sobre las épocas de entrenamiento y se realizan los pasos de entrenamiento y evaluación en cada época. En cada época, se calcula el loss de entrenamiento y validación y se almacenan en las listas train_losses y val_losses, respectivamente. Además, se guarda un checkpoint del modelo cada cierto número de épocas utilizando torch.save().

Se carga el checkpoint del modelo con el menor loss de validación obtenido durante el entrenamiento.
Se obtienen las predicciones del modelo en el conjunto de validación y se calcula la matriz de confusión normalizada utilizando la función confusion_matrix() de la biblioteca sklearn.metrics. Luego, se muestra la matriz de confusión utilizando la función heatmap() de la biblioteca seaborn. Se calcula el accuracy del modelo utilizando la función accuracy_score() de la biblioteca sklearn.metrics.

2.7.4. SVM polinomial

Se definen los valores de C y degree que se desean evaluar en la grilla de parámetros. Se concatenan las características seleccionadas y las etiquetas de entrenamiento y validación en los arrays X_concatenado y Y_concatenado, respectivamente.

Se crea el objeto PredefinedSplit utilizando el array train_val_labels, que indica qué datos son de entrenamiento (-1) y cuáles son de validación (1). Se crea el clasificador SVC con kernel polinomial. Se define la grilla de parámetros grilla que contiene los valores a explorar para C y degree.

Se crea el objeto GridSearchCV con el clasificador SVM polinomial, la grilla de parámetros y el objeto PredefinedSplit. Se entrena el objeto GridSearchCV utilizando el método fit(). Esto realizará la búsqueda exhaustiva de los mejores parámetros dentro de la grilla especificada.

Se obtiene el mejor valor de C y degree encontrados utilizando best_params_. También se obtiene el mejor score en validación utilizando best_score_. Se crea un nuevo clasificador SVC polinomial con el mejor valor de C encontrado. Se entrena este clasificador utilizando los datos de entrenamiento.

Se realiza la predicción utilizando el clasificador entrenado sobre el conjunto de validación. Se genera la matriz de confusión normalizada utilizando la función confusion_matrix() de la biblioteca sklearn.metrics y se muestra con la función heatmap() de la biblioteca seaborn. Finalmente, se calcula el accuracy utilizando la función accuracy_score() de la biblioteca sklearn.metrics.

2.7.5. Random Forest 2

Se definen los valores de n_estimators que se desean evaluar en la grilla de parámetros. Se concatenan las características seleccionadas y las etiquetas de entrenamiento y validación en los arrays X_select_rf y Y_select_rf, respectivamente.

Se crea el objeto PredefinedSplit utilizando el array train_val_labels, que indica qué datos son de entrenamiento (-1) y cuáles son de validación (1). Se crea el objeto GridSearchCV con el clasificador RandomForestClassifier, la grilla de parámetros y el objeto PredefinedSplit. Se entrena el objeto GridSearchCV utilizando el método fit(). Esto realizará la búsqueda exhaustiva de los mejores

parámetros dentro de la grilla especificada.

Se obtiene el mejor valor de n_estimators encontrado utilizando best_params_. También se obtiene el mejor score en validación utilizando best_score_. Se crea un nuevo clasificador RandomForestClassifier utilizando los mejores parámetros encontrados. Se entrena el clasificador utilizando las características seleccionadas y las etiquetas de entrenamiento.

Se realiza la predicción utilizando el clasificador entrenado sobre el conjunto de validación. Se genera la matriz de confusión normalizada utilizando la función confusion_matrix() de la biblioteca sklearn.metrics y se muestra con la función heatmap() de la biblioteca seaborn. Finalmente, se calcula la precisión utilizando la función accuracy_score() de la biblioteca sklearn.metrics

2.8. Conjunto de prueba

Finalmente se aplican el mismo proceso a las señales de prueba, calculando y obteniendo las mismas caracteristicas que el conjunto de entrenamiento. Y se aplica el mejor clasificador a este conjunto.

3. Resultados

3.1. Graficar señales

Una parte fundamental del análisis de las señales obtenidas de la base de datos es su representación gráfica. La visualización de las señales nos permite tener una comprensión más clara de su comportamiento y características, lo que puede ser de gran ayuda para el diseño e interpretación del sistema de clasificación.

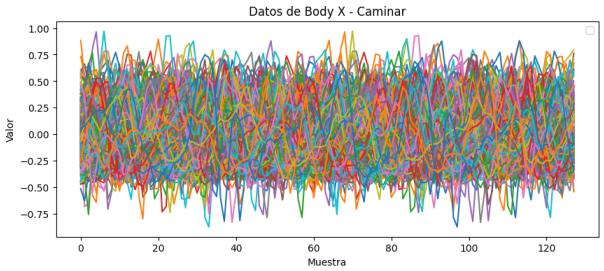

Figura 1: Señales Body X - Caminar

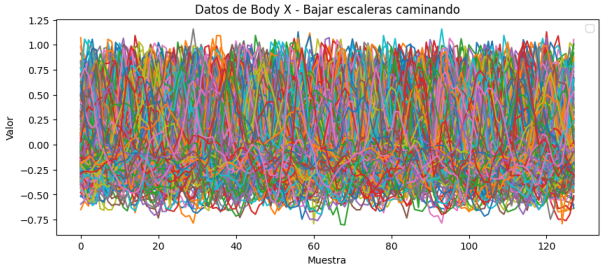

Figura 2: Señales Body X - Bajar escaleras caminando

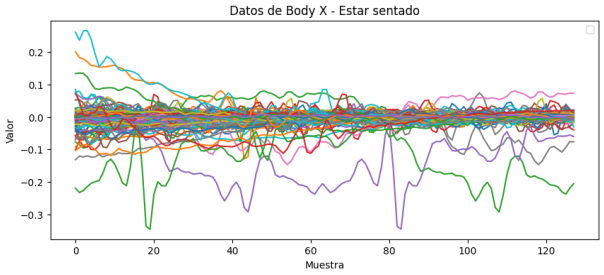

Figura 3: Señales Body X - Estar sentado

3.2. Filtros de señales

En el proceso de preprocesamiento de las señales, se implementaron cuatro filtros para mejorar la calidad de los datos. A continuación, se describe cada uno de ellos junto con sus parámetros y aplicación:

- Filtro Pasa Alta: Frecuencia de corte = 100 Hz, Orden = 5, Frecuencia de muestreo = 1000 Hz. El filtro se aplica a la señal df_body_acc_y
- Filtro Pasa Baja 1: Parámetros: Frecuencia de corte = 200 Hz, Orden = 5, Frecuencia de muestreo = 1000 Hz.El filtro se aplica a la señal df_body_gyro_x, df_body_total_x, df_body_total_y y df_body_total_z
- Filtro Pasa Baja 1: Parámetros: Frecuencia de corte = 202 Hz, Orden = 5, Frecuencia de muestreo = 1000 Hz.El filtro se aplica a la señal df_body_gyro_z

Luego de la aplicacion de los filtros y StandarScaler se visualizad su clusterizacion

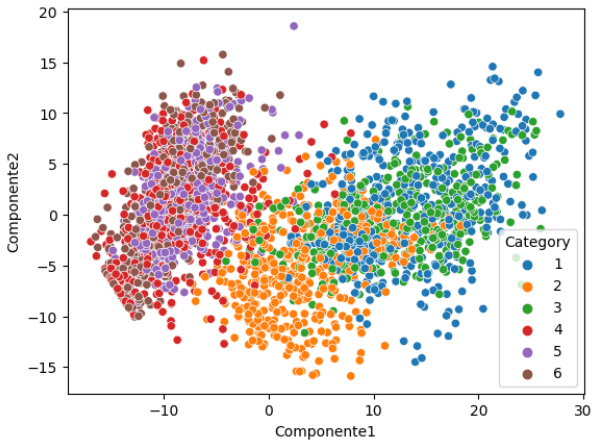

Figura 4: Clustering

3.3. Clasificadores

Los clasificadores utilizados en el proyecto mostraron resultados prometedores en términos de precisión y exactitud en la clasificación de las actividades físicas. A continuación, se presentan los resultados obtenidos para cada clasificador:

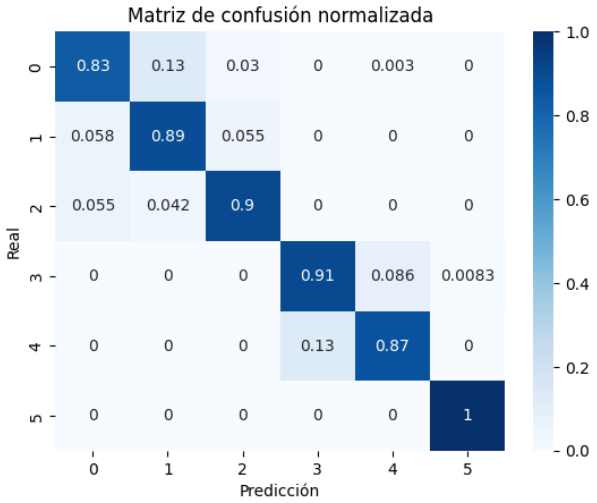

Figura 5: Matriz de confusion RandomForest en el conjunto de validacion

El clasificador Random Forest demostró un alto nivel de precisión en la clasificación de las actividades físicas, con una exactitud del 90.25 %. Este resultado indica que el modelo fue capaz de realizar predicciones precisas en la mayoría de los casos.

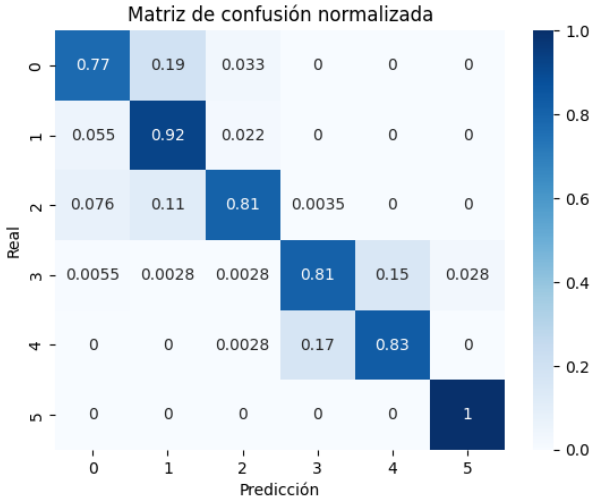

Figura 6: Matriz de confusion red neuronal en el conjunto de validacion

La red neuronal implementada también mostró un buen desempeño, con una exactitud del 85.9 %. Esto indica que el modelo logró clasificar correctamente la mayoría de las actividades físicas en base a las señales de entrada.

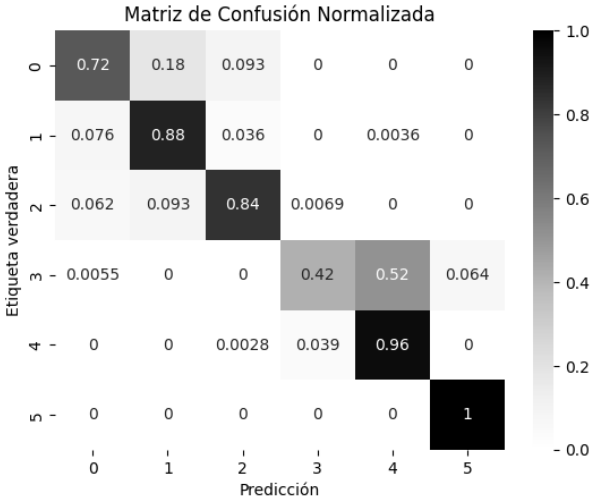

Figura 7: Matriz de confusion red neuronal 2 en el conjunto de validacion

La segunda red neuronal implementada obtuvo una exactitud del 80.15 %. Aunque ligeramente inferior a los clasificadores anteriores, este resultado sigue siendo significativo y muestra la capacidad de la red neuronal para clasificar las actividades físicas con una precisión razonable.

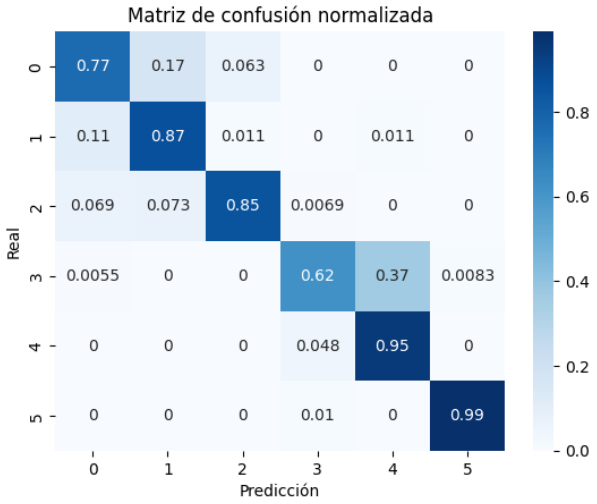

Figura 8: Matriz de confusion SVC polinomial en el conjunto de validacion

El clasificador SVC (Support Vector Machine) con kernel polinomial mostró una exactitud del 84.3 %. Esto indica que el modelo fue capaz de realizar predicciones precisas utilizando una función polinomial para separar las clases de actividades físicas.

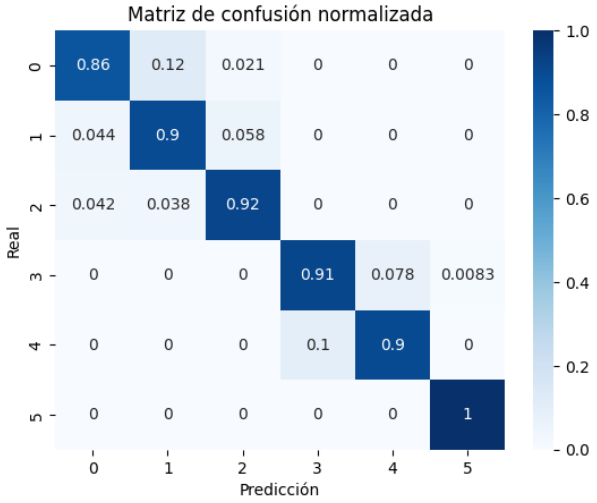

Figura 9: Matriz de confusion RandomForest 2 en el conjunto de validacion

El ultimo clasificador Random Forest demostró un alto nivel de precisión en la clasificación de las actividades físicas, con una exactitud del 91.7 %. Siendo este el mejor clasificador obtenido.

Este ultimo clasificador fue utilizado en la competencia kaggle obteniendo un 93.66 % de etiquetas correctas, lo cual indica un muy buen resultado de este clasificador.

4. Conclusiones

En conclusión, el proyecto de clasificación de actividades físicas utilizando la base de datos Human Activity Recognition Using Smartphones ha arrojado resultados satisfactorios y prometedores. A partir del análisis de las señales de entrada y el diseño e implementación de varios clasificadores, se ha logrado desarrollar un sistema capaz de reconocer y clasificar las actividades físicas con una precisión considerable.

Los resultados obtenidos revelan que el clasificador Random Forest 2 ha demostrado la mayor exactitud, con un valor de 0.917. Estos clasificadores han mostrado su capacidad para realizar predicciones precisas basadas en las señales de entrada.

Además, se han implementado dos redes neuronales, que han alcanzado una exactitud del 0.859 y 0.8015 respectivamente. Estos resultados respaldan la eficacia de las redes neuronales en la clasificación de actividades físicas y resaltan su potencial para abordar problemas de reconocimiento de patrones.

Es importante destacar que el proceso de preprocesamiento de datos, que incluyó el filtrado de señales, ha demostrado ser fundamental para mejorar la calidad de los datos y, por lo tanto, el desempeño de los clasificadores. Los filtros implementados han contribuido a reducir el ruido y mejorar la capacidad de discriminación de las señales.

En general, este proyecto ha demostrado la viabilidad y efectividad de los clasificadores y técnicas utilizados para el reconocimiento de actividades físicas. Los resultados obtenidos indican que el sistema de clasificación desarrollado tiene potencial para su implementación en aplicaciones prácticas, como monitores de actividad física o asistentes de entrenamiento personalizados.

Sin embargo, es importante tener en cuenta que siempre existen áreas de mejora y oportunidades para futuras investigaciones. Entre las posibles mejoras, se pueden considerar la optimización de los parámetros de los clasificadores, la exploración de otros algoritmos de clasificación y la incorporación de características adicionales para mejorar aún más la precisión de la clasificación.