Machine Learning 2024-1 Proyecto 2

Luis Enrique Cortijo Gonzales 100 %, Eddison Pinedo Espinoza 100 %,
Davi Magalhaes Eler 100 %, Sanders Chancan Chanca 100 %
Universidad de Ingenieria y Tecnologia UTEC
Repositorio del proyecto

***Resumen*—En el presente informe, se presenta el proceso, la experimentación y los resultados de la creación de dos modelos de machine learning para clasificación. Estos modelos fueron entrenados y probados con el conjunto de datos "Human Activity Recognition". Este conjunto de datos consiste en un experimento realizado por un grupo de 30 voluntarios que llevaron a cabo seis actividades de movimiento: caminar, subir escaleras, bajar escaleras, sentarse, estar de pie y estar tendidos. Estas actividades fueron monitoreadas usando un teléfono inteligente que registraba datos de los ejes "x", "y" y "z", velocidad de movimiento, aceleración, velocidad angular, entre otros.**

El objetivo del proyecto es plantear los modelos adecuados en base al conjunto de datos, asegurar el correcto uso de los datos para su procesamiento y formular las conclusiones en base a los resultados de la experimentación.

***Index Terms*—Machine learning, clasificación, Human Activity Recognition, ejes de movimiento, velocidad, aceleración, velocidad angular, preprocesamiento de datos.**

I. Introducción

El concepto de clasificación consiste en la acción lógica de agrupar, separar o crear conjuntos y/o series donde los elementos comparten características afines o similares. Este concepto lógico es potenciado y mejorado con la computación, resultando en los modelos de clasificación de Machine Learning. En este sentido, los modelos de machine learning de clasificación consisten en la agrupación, separación o creación de conjuntos utilizando la computación con el fin de agilizar la velocidad del proceso y permitir la clasificación de grandes cantidades de datos, que un ser humano no podría manejar eficientemente.

La acción de clasificar en el contexto computacional, parte de una base matematica en la cual es necesario llevar el dato de entrada a un plano matematico. Este proceso de conversion de un tipo de dato a una representacion matematica (ya sea matriz, vector caracteristico u otra) se llama preprocesamiento. Acción en la cual aplicando distintas tecnicas para realizar el proceso descrito.

Una vez tenemos los datos lo que hacen los modelos es clasificarlos y en base al proceso son capaces de recibir nuevos datos y clasificarlos correspondientemente. Este proceso es la predicción.

En sistesis realizar un modelo de machine learning de clasificación, consta de 3 importantes hitos a lograr correctamente: el preprocesamineto, el entrenamineto y la predicción.

I-A. Objetivos

Los objetivos del proyecto son los siguientes:

- Comprender y analizar los modelos para su correcta elección y desarrollo.
- Analizar los datos y decidir qué herramientas y técnicas usar para su reducción de dimensionalidad.
- Desarrollar 2 modelos de machine learning de clasificación.
- Lograr que el proceso de preprocesamiento y entrenamiento sea eficiente computacionalmente, garantizando la escalabilidad del modelo a mayores cargas de datos.
- Obtener una predicción adecuada a partir de nuestros modelos.

I-B. Importancia

Las aplicaciones de un modelo de machine learning en la clasificación son fundamentales cuando se trabaja con volúmenes considerables de datos o cuando la clasificación humana es complicada o incluso imposible. Estas aplicaciones son cruciales en áreas como la identificación de personas y objetos, la provisión de respuestas asistidas con inteligencia artificial, la detección de tumores o elementos anómalos en el campo de la salud, y en cualquier otro ámbito que requiera la clasificación de elementos.

II. Dataset

El dataset es un conjunto de datos tomados de un grupo de 30 voluntarios realizando 6 actividades: caminar, subir escaleras, bajar escaleras, sentarse, estar de pie y estar tendidos. Estas actividades se registraron mediante el acelerómetro y giroscopio de un smartphone. Un dato importante a tener en cuenta es que los datos fueron particionados aleatoriamente en un 70 % para el entrenamiento y un 30 % para las pruebas, ademas de presentar una frecuencia similar entre cada variable, por lo que se puede afirmar que no esta desbalanceado.

En el dataset encontramos las siguientes variables (cada vez que se hace referencia a una variable en un eje, implica la existencia de la misma en los otros ejes):

- `total_acc_x`: Esta variable guarda la aceleración del eje X, en unidades estándar de gravedad "g", cada dato es un vector de 128 elementos.
- `body_acc_x`: La aceleración obtenida al restar la gravedad.
- `body_gyro_x`: La velocidad angular en el eje especificado en radianes/segundo.

En el dataset para el proyecto contamos con los siguientes archivos:

- `activity_labels.txt`: Contiene las etiquetas en inglés de cada una de las actividades que tenían que realizar los voluntarios.

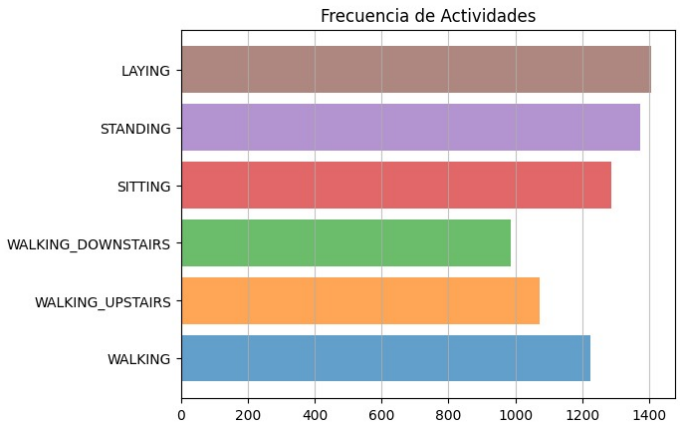

Figura 1. Grafico de distribución de las variables

- `train.h5`: Archivo que contiene las etiquetas y series de tiempo destinadas al entrenamiento del modelo.
- `test.h5`: Archivo que contiene las series de tiempo para probar el modelo.
- `sample_submission.csv`: Un ejemplo de cómo mostrar los resultados obtenidos de nuestro modelo en formato CSV.

III. Metodología

III-A. Extracción de Features

III-A1. PyTS: Para extraer características de series de tiempo, utilizamos el transformer ROCKET de PyTS. Este método utiliza kernels convolucionales, que son filtros que recorren los datos de entrada y realizan operaciones matemáticas para extraer información útil. Cada kernel aplica un producto punto con los datos de la serie de tiempo y suma los resultados para obtener un solo valor.

ROCKET es rápido y permite ajustar el número de features extraídos con el hiperparámetro *n_kernels*. Lo que nos permitió experimentar en la fase de training.

III-A2. TSFresh: Esta librería automatiza la extracción de características relevantes de series temporales, lo que facilita la construcción de modelos predictivos. Para este proyecto principalmente se utilizó la función extract_features para extraer las características más relevantes de las series de tiempos dadas en el dataset. Según la documentación oficial se tienen unos 82 diferentes features que se pueden extraer de una serie de tiempo. En la siguiente tabla se muestra un ejemplo de algunos de estos features junto con su interpretación:

Feature	**Interpretación**
abs_energy(x)	Calcula la energía absoluta de la serie temporal, la cual es la suma de los valores al cuadrado. Es útil para medir la magnitud total de la señal a lo largo del tiempo.
approximate_entropy(x, m, r)	Mide la regularidad y la previsibilidad de las fluctuaciones en una serie temporal. Un valor bajo indica una serie más predecible. Para ello implementa un algoritmo que calcula una aproximación de la entropía vectorizada.
number_peaks(x, n)	Cuenta el número de picos en la serie temporal que tienen al menos *n* puntos de soporte. Este feature es útil para identificar la frecuencia de fluctuaciones significativas.
sample_entropy(x)	Similar a la entropía aproximada pero más consistente para series temporales cortas, evaluando la complejidad de la serie temporal.
cwt_coefficients(x, param)	Calcula los coeficientes de la transformada continua de wavelet, lo cual es útil para detectar patrones a diferentes escalas temporales y frecuencias en la serie.

Cuadro I
FEATURES DE TSFRESH

Asimismo, la librería también cuenta con una función denominada select_features, que selecciona los features más relevantes basándose en los datos del target. En ese sentido, esta función realiza de cierta forma una reducción de dimensionalidad al vector de características; sin embargo, podría aumentar el riesgo de overfitting, pues se deben cargar los targets para realizar la reducción.

III-B. Modelos de Clasificación

III-B1. Árbol de Decisión: Los árboles de decisión son modelos de aprendizaje supervisado utilizados para la clasificación y regresión de datos. Estos modelos se construyen a partir de una serie de decisiones basadas en los valores de los features de los datos, que subdividen desde conjunto inicial en subconjuntos cada vez más homogéneos, hasta alcanzar las hojas del árbol, que representan las decisiones finales o predicciones

III-B1a. Métricas de Evaluación: Durante la construcción de un árbol de decisión, se emplean principalmente dos métricas para evaluar la calidad de las divisiones:

1. **Entropía**
   La entropía mide el desorden o la incertidumbre en un conjunto de datos y se utiliza para determinar la eficacia de una división en los árboles de decisión. Se define como:
   $$Entropy(Y) = -\sum_{i=1}^{J} p_i \log_2 p_i$$
   donde p_i representa la proporción de la clase i en el conjunto Y, y J es el número total de clases
2. **Impureza de Gini**

La impureza de Gini es una medida que evalúa la probabilidad de una incorrecta clasificación. Se calcula con la fórmula:

$$Gini(Y) = 1 - \sum_{i=1}^{J} p_i^2$$

donde p_i y J son los mismos que en la fórmula de la impureza.

Para este proyecto, se escogió la métrica de Impureza Gini para evaluar la calidad de las divisiones, pues según lo visto en las clases otorga divisiones de mayor calidad.

III-B1b. Proceso de Construcción del Árbol: El proceso de construcción de un árbol de decisión comienza en la raíz y procede de forma recursiva. En cada paso, se selecciona la característica y el valor que mejor subdividen el conjunto de datos, maximizando la pureza de los subconjuntos resultantes según las métricas mencionadas. Este proceso se repite en cada nodo nuevo hasta que se alcanza un criterio de parada, como que todos los datos en un nodo pertenecen a la misma clase. En ese sentido, para la implementación del código de este método de clasificación se tuvo como referencia el siguiente pseudocódigo presentado en clase:

- **Entrada:** Conjunto de ejemplos S.
- **Salida:** Árbol de decisión DT.
- Si todos los ejemplos en S pertenecen a la misma clase c:
  - Devuelve una nueva hoja y etiquétala con c.
- En caso contrario:
  1. Seleccionar un atributo A según alguna función Gini.
  2. Generar un nuevo nodo DT con A como prueba.
  3. Para cada valor v_i de A:
     a) Sea S_i todos los ejemplos en S con $A = v_i$.
     b) Utilizar ID3 para construir un árbol de decisión DT_i para el conjunto de ejemplos S_i.
     c) Generar un enlace que conecte DT y DT_i.

III-B2. Regresión Logística: La regresión logística es un modelo estadístico utilizado en Machine Learning para abordar problemas de clasificación binaria. Este modelo se caracteriza por estimar la probabilidad de que una instancia pertenezca a una clase particular en función de sus características o variables predictoras. El objetivo principal del modelo reside en la identificación de los coeficientes óptimos que maximicen la verosimilitud de los datos de entrenamiento.

Inicialmente, la data presentada contenía varias etiquetas, lo que hacía ineficiente el uso de un modelo de regresión logística binaria. La regresión logística binaria está diseñada para problemas de clasificación con solo dos clases. Al enfrentar un problema con múltiples clases, este modelo no es capaz de manejar adecuadamente la clasificación, ya que solo puede diferenciar entre dos estados (0 o 1). Por tal motivo, se optó por implementar un modelo de regresión logística para múltiples clases, conocido como regresión softmax o regresión logística multinomial.

Regresión logística multinomial extiende la regresión logística binaria al manejar más de dos clases de salida. Las principales diferencias radican en el número de clases manejadas, la función de activación utilizada (softmax en lugar de sigmoide) y la función de pérdida empleada (entropía cruzada categórica en vez de binaria). No obstante, ambos modelos comparten la base de la regresión logística, modelando probabilidades de pertenencia a clases mediante una función de activación y optimizando una función de pérdida basada en entropía cruzada mediante gradiente descendente. La regresión softmax representa una extensión natural de la regresión logística binaria para abordar problemas de clasificación multiclase, manteniendo los fundamentos de optimización y probabilidad de la regresión logística.

III-B2a. Métricas de Evaluación: Durante la ejecución del modelo de regresión logística multinomial (o softmax), usamos la funcion LOSS como métrica de evaluación para medir su desempeño.

1. **LOSS**
   La métrica principal que el modelo utiliza durante el entrenamiento es la entropía cruzada categórica. Esta métrica mide la diferencia entre las distribuciones de probabilidad predichas y las reales, definida como:

$$LOSS = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log(\hat{y}_{ij})$$

   donde N es el número de nuestras, C es el número de calses, y_{ij} es la entrada i-ésima para la clase j, y $\hat{y}_{ij}$ es la probabilidad predicha para la muestra i-ésima de la clase j.

En la implementacion del modelo tenemos la funcion $Softmax$ con el objetivo de clasificar las entradas en una de varias clases posibles. La función 'softmax' toma como entrada un vector de valores (logits) y los transforma en un vector de probabilidades que suman 1. Esto se hace de manera que cada valor en el vector de salida representa la probabilidad de que la entrada pertenezca a una clase particular. La función 'softmax' se define matemáticamente como:

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

donde z_i es el logit correspondiente a la clase i y C es el número de calses.

IV. Reducción de la dimencionalidad

Dentro del preprocesamiento de los datos, la reducción de la dimensionalidad juega un papel crucial para mejorar el proceso de entrenamiento del modelo. A primera vista, utilizar todos los datos en sus dimensiones originales parece una buena idea, pero esta práctica presenta dos problemas principales: el costo computacional y el sobreajuste (overfitting). Una sola unidad de datos multimedia o similares contiene un gran número de dimensiones al ser convertida a una representación matemática. Asimismo, un conjunto de datos orientado al machine learning suele contener una cantidad considerable

de datos, lo que implica que utilizar toda esta información conlleva altos costos computacionales, que a su vez provocan problemas como tiempos de procesamiento prolongados y requisitos elevados de hardware, con todas las implicaciones que esto conlleva.

Por otro lado, el sobreajuste significa que el modelo se ajusta excesivamente a los datos de entrenamiento, lo que resulta en malas predicciones cuando se enfrenta a nuevos datos. Esto se debe a que el modelo ha aprendido demasiado sobre las particularidades del conjunto de entrenamiento, perdiendo así su capacidad de generalización.

Para abordar estos problemas, reducimos la dimensionalidad de los datos. Esto nos permite obtener un conjunto de datos más fácil de procesar en términos computacionales. Además, al conservar solo las dimensiones relevantes, el modelo no se ajusta a datos irrelevantes, permitiendo así obtener mejores predicciones.

IV-A. PCA

Principal Component Analysis (PCA) es la técnica que elegimos para el preprocesamiento de los datos en este proyecto. Esta función recibe como parámetros el dataset en vectores, ya sean estandarizados previamente o no, y un número entero k.

PCA se basa en la construcción de una matriz de covarianza para identificar la relación entre las características. La fórmula para calcular la matriz de covarianza es:

$$\mathbf{C} = \frac{1}{n-1} \sum_{i=1}^{n} (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^\top$$

donde $\mathbf{x}_i$ representa los datos, $\bar{\mathbf{x}}$ es el vector de medias de los datos, y n es el número de muestras. Posteriormente, se procede al cálculo de los valores propios (eigenvalues) y los vectores propios (eigenvectors) a partir de la matriz de covarianza $\mathbf{C}$. La fórmula para este cálculo es:

$$\mathbf{C}\mathbf{v} = \lambda \mathbf{v}$$

donde λ representa los eigenvalues y $\mathbf{v}$ los eigenvectors. Los eigenvalues y eigenvectors se ordenan de mayor a menor, y se seleccionan los primeros k eigenvalues y sus correspondientes eigenvectors. Los datos originales se proyectan sobre estos k vectores propios para obtener los nuevos vectores característicos. La proyección se realiza mediante la siguiente fórmula:

$$\mathbf{Z} = \mathbf{X}\mathbf{W}$$

donde $\mathbf{Z}$ son los datos transformados, $\mathbf{X}$ es la matriz de datos original y $\mathbf{W}$ es la matriz de los primeros k eigenvectors. Esta transformación permite reducir la dimensionalidad de los datos, facilitando su procesamiento y mejorando la capacidad del modelo para generalizar sobre nuevos datos.

IV-B. Justificación

En el proyecto se empleó PCA sobre otras técnicas para la reducción de la dimensionalidad por los siguientes criterios:

- **Eficiencia computacional**: PCA es una técnica lineal en su ejecución, lo que la hace más eficiente que otras técnicas no lineales cuando se utiliza con grandes cantidades de datos.
- **Mejores resultados con los modelos**: PCA proporciona vectores que, según la experimentación realizada, resultaron ser más rápidos en términos computacionales durante el entrenamiento.
- **Mejores predicciones**: Durante la experimentación, PCA arrojó mejores resultados en las predicciones. Esto se debe a la regularización implícita que realiza PCA al devolver los primeros k vectores, lo que evita el sobreajuste (overfitting).

V. Implementación

La implementación de los modelos se encuentra en la carpeta models la cuál es importada en los diferentes archivos .ipynb cuando son necesarios. Es por ello, que el proceso de visualización de datos, extracción de features, experimentación y entrenamiento de modelos se realizan en sus respectivos archivos .ipyn. Es importante destacar que la replicabilidad se puede lograr, ya que se emplea el número 42 como semilla en la función *train_test_split* y se establece el *np.random.seed* en 2024.

VI. Experimentación

Como se evidenció al momento de hacer la visualización de datos la cantidad de labels está distribuida de manera que no se observa un desbalance muy grande. Es por ello, que se utilizó la métrica del accuracy para poder evaluar los modelos desarrollados.

VI-A. Elección de liberia para extracción de features

Experimentamos con ambas librerías *PyTS* y *TsFresh*. Como comentado anteriormente usamos ROCKET de PyTS para nuestra submission por su eficiencia y el accuracy superior que obtuvimos. Dividimos 80 % del training set para entrenar y 20 % para testear, obteniendo los siguientes resultados al reducir la dimensionalidad a 10 con PCA, usando nuestros modelos:

modelo	accuracy ROCKET	accuracy TSFresh
Regresión Logística	84.37 %	71.4 %
Decision tree	79.10 %	52.35 %
SVM	47.99 %	22.64 %

En cuanto a la velocidad al momento de extraer features con las dos librerias obtuvimos lo siguiente:

	PyTS ROCKET	TSFresh
#features extraídos	1000	7047
Tiempo de ejecución	37s	47m22s

Observación: Probamos extraer 20.000 features con ROCKET lo cual nos tardó 12m47s, demostrando una eficiencia mucho más alta que TSFresh.

Hemos testeado varios valores para el hiperparametro *n_kernels* de ROCKET. Los accuracies obtenidos fueron usando SVC de *Sklearn* (para poder probar de forma rápida). El número de features extraídos es $n\_kernels*2$ (e.g. 100 kernels nos resulta en un dataset con 200 features).

n_kernels	training time	accuracy
100	2s	94.69 %
500	6s	95.24 %
1000	15s	95.37 %
10000	9m29s	95.44 %

Al aumentar *n_kernels* el accuracy casi no cambia, pero el tiempo de entrenamento del modelo aumenta rapidamente, el cual se vería aun más alto en nuestro modelo que no es tán optimizado como el de *sklearn*. Por estos motivos hemos elejido *n_kernels = 500*, que extrae 1000 features.

VI-B. Experimentación de modelos

Como se evidenció al momento de hacer la visualización de datos la cantidad de labels está distribuida de manera que no se observa un desbalance muy grande. Es por ello, que se utilizó la métrica del accuracy para poder evaluar los modelos desarrollados.

Para la experimentación, como se señaló anteriormente, se utilizaron los datos provenientes de los features obtenidos por la libreria PyTS utilizando el método Rocket, y también se usó el método PCA para la reducción de dimensionalidad.

Asimismo, es importante destacar que, aunque los métodos principales empleados en este proyecto fueron el Árbol de Decisión y la Regresión Logística, también se probó el método de Máquinas de Soporte Vectorial (SVM). Sin embargo, como se detallará más adelante, este último no se incluyó en el informe final debido a su bajo rendimiento en la métrica de accuracy.

Es así que la primera experimentación que se tuvo con los tres modelos fue en base a su accuracy y a la cantidad de componentes a la que se reduce utilizando PCA. Dichos resultados se pueden ver en el siguiente gráfico:

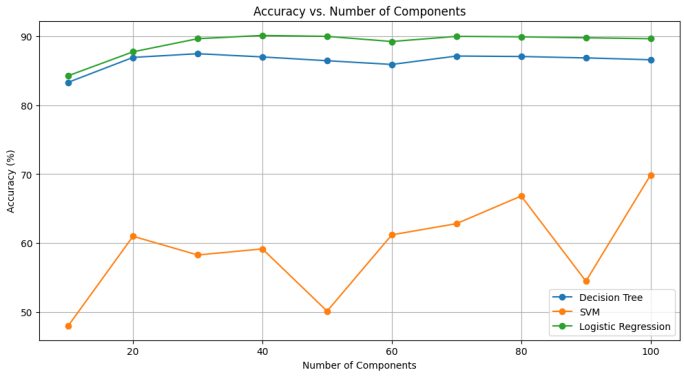

Figura 2. Experimento 01: Accuracy vs Number of Components

En un primer momento sostenemos que a medida de que se utilicen más componentes, la accuracy del modelo aumentará, razonamiento que es cierto para el caso del Árbol de Decisión y la Regresión Logística; sin embargo, se llega a un punto de saturación en donde por más que se aumente la cantidad de componentes no se llega a aumentar la accuracy del modelo. En el caso del SVM, también se puede observar esa tendencia a aumentar la accuracy conforme aumentamos la cantidad de componentes; sin embargo, se observa una gran diferencia en la accuracy de este modelo frente a los demás.

La segunda parte de la experimentación consistía en evaluar el tiempo de entrenamiento en relación a la cantidad de componentes utilizados. Los resultados de esta experimentación se pueden apreciar en la siguiente gráfica:

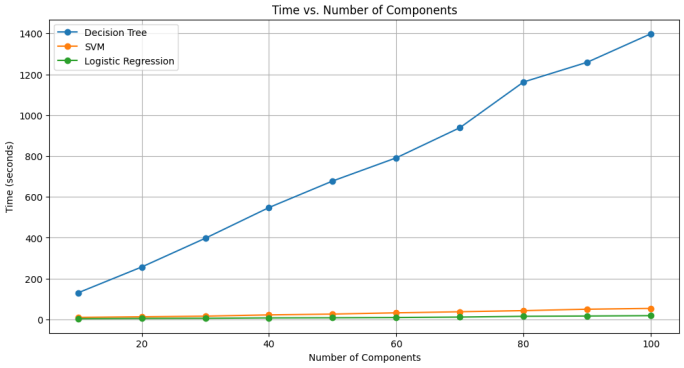

Figura 3. Experimento 02: Tiempo vs Number of Components

Se observa que el modelo de Árbol de Decisión toma un tiempo considerablemente más alto que la regresión logística y el SVM. Y que todos los modelos tienen tendencia lineal de aumentar su tiempo de entrenamiento a medida de que se aumenta la cantidad de característica a analizar.

El último experimento consistió en realizar la matriz de confusión del modelo de Árbol de Decisión y Regresión Logística. Ambos resultados se muestran a continuación:

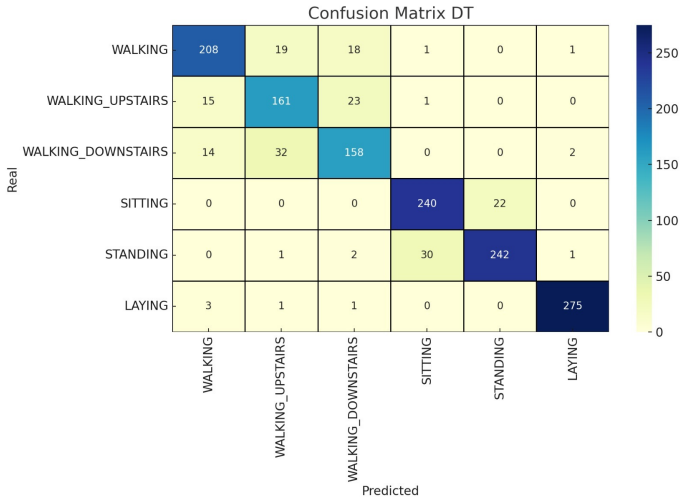

Figura 4. Matriz de Confusión del Árbol de Decisión

En base a la matriz de confusión de ambos modelos se puede observar que tienen un desempeño parecido casi parecido

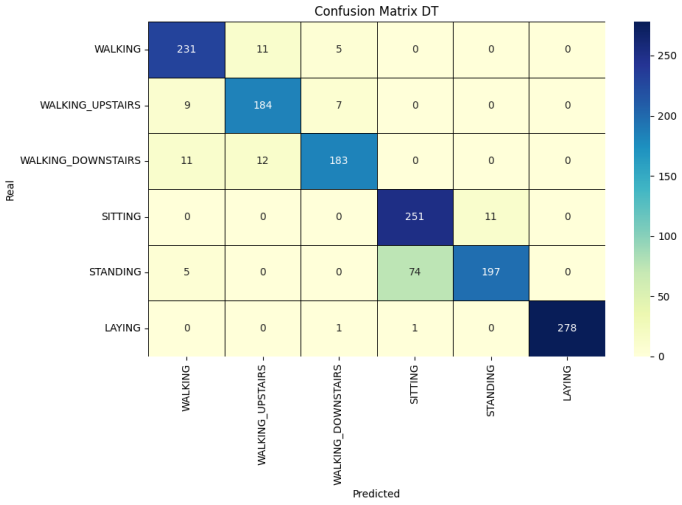

Figura 5. Matriz de Confusión de Regresión Logística

al momento de realizar la clasificación. Sin embargo, la Regresión Logistica tiene una pequeña superioridad.

Es así que, considerando el accuracy y el tiempo de entrenamiento, el modelo de Regresión Logistica es el mejor modelo de clasificación para este caso. Por lo tanto, este es el modelo que se entrenará y se enviará a la competencia de Kaggle

VII. Discución

Dentro de los resultados obtenidos, podemos observar en las figuras 4 y 5 que, en ambos modelos, la mayor cantidad de errores se encuentra en las etiquetas ”SITTINGz ”STANDING”. En particular, es en la regresión logística donde se aprecia mejor esta ligera confusión del modelo. Podemos interpretar que esta confusión es resultado de la similitud de movimientos plasmada en los datos, debido a que teoricamente estas acciones son estaticas durante la mayor parte del tiempo. Por lo que el mayor error se produce al predecir que alguien está de pie cuando en realidad está sentado.

El segundo error de predicción más común se encuentra en las acciones de subir y bajar escaleras en el árbol de decisión. Aunque este error es menos frecuente que el anterior, se puede interpretar que ocurre porque, aunque son acciones diferentes en dirección, el modelo deduce que las similitudes en los ejes causadas por el recorrido de las escaleras son similares en un porcentaje reducido de los datos, clasificándolos erróneamente.

En el resto de etiquetas, ambos modelos son capaces de identificar cada acción sin un error considerable, salvo algunas instancias en las que la etiqueta de caminar se confunde con la de subir escaleras en el árbol de decisión.

Ademas de lo ya mencionado, los resultados de ambos modelos son predicciones con gran presición, por lo que se puede concluir que es un buen modelo de clasificación,

VIII. Conclusión

Como se evidenció en el presente documento, se logró desarrollar correctamente un modelo de machine learning de clasificación, abarcando desde el preprocesamiento de los datos hasta el entrenamiento y la predicción de resultados.

Además de los logros tangibles como el modelo, se adquirió una mejor comprensión del funcionamiento de los modelos, técnicas de preprocesamiento, reducción de la dimensionalidad, etc. Se aprendió a discernir en base a criterios como la velocidad en tiempo de ejecucion o las entre las opciones para la extracción de las dimenciones con el fin de obtener el mejor resultado en base a las pruebas y experimentación. Entre las opciones de modelos de igual manera se dicernio en base a criterios como la presición, velocidad de ejecución, entre otros para encontrar los 2 que se adapten mejor a la situación planteada.

IX. Referencias

Las siguientes fuentes fueron consultadas durante la realización del trabajo:

- Johann Faouzi Github: johannfaouzi. *Transformation — pyts 0.11.0 documentation*. 2023. URL: https://pyts.readthedocs.io/en/latest/modules/transformation.html
- Blue Yonder. *tsfresh*. 2023. URL: https://github.com/blue-yonder/tsfresh
- Aledeluna. *Binary Time Series Classification Problem*. 2023. URL: https://www.kaggle.com/code/aledelunap/binary-time-series-classification-problem
- AKSEL UHR. *Fulltext01*. 2023. URL: https://www.diva-portal.org/smash/get/diva2:1782720/FULLTEXT01.pdf
- Nils Github: ni79ls. *Human Activity Recognition with Keras and CoreML*. 2023. URL: https://github.com/ni79ls/har-keras-coreml/blob/master/Human%20Activity%20Recognition%20with%20Keras%20and%20CoreML.ipynb
- Lin Fan; Zhongmin Wang; Hai Wang. *IEEE Xplore Document*. 2023. URL: https://ieeexplore.ieee.org/document/6824574
- Normalized Nerd. *Decision Tree Classification in Python*. 2021. URL: https://www.youtube.com/watch?v=sgQAhG5Q7iY