Proyecto 1: Clasificación

Gabriel Vargas, Jesús Niño, Rafael Ramos, Angel Tribeño
Universidad de Ingeniería y Tecnología
Correos electrónicos: {gabriel.vargas, jesus.nino, rafael.ramos, angel.tribeno}@utec.edu.pe

***Abstract*—"Este proyecto consiste en el estudio y aplicación de modelos de clasificación sobre un conjunto de datos seleccionado de Kaggle. Se presenta tanto la parte teórica (regresión lineal y regresión logística) como la parte aplicada (implementación, experimentación y análisis de resultados)."**

***Index Terms*—Clasificación, Regresión Lineal, Regresión Logística, Aprendizaje Automático.**

I. Parte Teórica

1.1 Regresión Lineal (Mínimos Cuadrados Ordinarios)

Consideramos un conjunto de datos $\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, donde $x^{(i)} \in \mathbb{R}^{M+1}$ es el vector de características del i-ésimo ejemplo de entrenamiento (incluyendo una coordenada de sesgo $x_0^{(i)} = 1$) e $y^{(i)} \in \mathbb{R}$ es el objetivo. El vector de parámetros es $w = (w_0, w_1, \ldots, w_M)^\top$.

Función objetivo.: El costo de mínimos cuadrados ordinarios es

$$J(w) \;=\; \tfrac{1}{2}\sum_{i=1}^{N}\Big(y^{(i)} - w^\top x^{(i)}\Big)^2.$$

Residuales.: Para cada punto de datos definimos el residual

$$e^{(i)} \;=\; y^{(i)} - w^\top x^{(i)}.$$

Entonces

$$J(w) = \tfrac{1}{2}\sum_{i=1}^{N}(e^{(i)})^2.$$

Derivada.: La derivada parcial con respecto a w_k es

$$\frac{\partial J}{\partial w_k} = -\sum_{i=1}^{N} x_k^{(i)}\Big(y^{(i)} - \sum_{j=0}^{M} w_j x_j^{(i)}\Big).$$

Ítem 1: Expresión para w_k.: Igualando la derivada a cero y despejando w_k se obtiene

$$w_k \sum_{i=1}^{N}(x_k^{(i)})^2 = \sum_{i=1}^{N} x_k^{(i)} y^{(i)} - \sum_{j\neq k} w_j \sum_{i=1}^{N} x_k^{(i)} x_j^{(i)}.$$

Por lo tanto

$$\boxed{w_k \;=\; \frac{\sum_{i=1}^{N} x_k^{(i)} y^{(i)} - \sum_{j\neq k} w_j \sum_{i=1}^{N} x_k^{(i)} x_j^{(i)}}{\sum_{i=1}^{N}(x_k^{(i)})^2}}$$

Ítem 2: Número de coeficientes y ecuaciones.: El vector de parámetros w tiene $M+1$ coeficientes (incluyendo el sesgo w_0). Para cada $k = 0, \ldots, M$, la condición $\frac{\partial J}{\partial w_k} = 0$ produce una ecuación. Así obtenemos exactamente $M+1$ ecuaciones para $M+1$ incógnitas.

Forma matricial.: En notación matricial, sea $X \in \mathbb{R}^{N\times(M+1)}$ la matriz de diseño e $y \in \mathbb{R}^N$ los objetivos. El gradiente es

$$\nabla_w J = X^\top X w - X^\top y,$$

y las ecuaciones normales son

$$\boxed{X^\top X w = X^\top y}.$$

Si $X^\top X$ es invertible, la solución en forma cerrada es

$$w = (X^\top X)^{-1} X^\top y.$$

1.2 Regresión Logística (una característica, datos separables)

Consideramos un conjunto de datos de clasificación binaria $\{(x^{(i)}, t^{(i)})\}_{i=1}^{N}$ con $x^{(i)} \in \mathbb{R}$ y etiquetas $t^{(i)} \in \{0, 1\}$. El modelo logístico (incluyendo sesgo) es

$$\begin{aligned} p\big(t^{(i)} = 1 \mid x^{(i)}; w, w_0\big) &= \sigma(z^{(i)}),\\ z^{(i)} &= w x^{(i)} + w_0,\\ \sigma(z) &= \frac{1}{1+e^{-z}}. \end{aligned}$$

Pérdida (log-verosimilitud negativa / entropía cruzada).: Asumiendo ejemplos independientes, la verosimilitud es

$$L(w, w_0) = \prod_{i=1}^{N} \big(\sigma(z^{(i)})\big)^{t^{(i)}} \big(1 - \sigma(z^{(i)})\big)^{1-t^{(i)}}.$$

Minimizamos la log-verosimilitud negativa (entropía cruzada):

$$\ell(w, w_0) = -\sum_{i=1}^{N}\Big[t^{(i)} \log \sigma(z^{(i)}) + (1-t^{(i)}) \log(1-\sigma(z^{(i)}))\Big].$$

Gradiente.: Usando la identidad $\sigma'(z) = \sigma(z)(1-\sigma(z))$, la derivada con respecto a w es

$$\frac{\partial \ell}{\partial w} = \sum_{i=1}^{N}\big(\sigma(z^{(i)}) - t^{(i)}\big)\, x^{(i)}, \qquad \frac{\partial \ell}{\partial w_0} = \sum_{i=1}^{N}\big(\sigma(z^{(i)}) - t^{(i)}\big).$$

Estas expresiones son exactamente las direcciones de actualización usadas en el descenso de gradiente:

$$w \leftarrow w - \eta \sum_{i=1}^{N}(\sigma(z^{(i)}) - t^{(i)}) x^{(i)}, \qquad w_0 \leftarrow w_0 - \eta \sum_{i=1}^{N}(\sigma(z^{(i)}) - t^{(i)}).$$

Datos separables (definición).: El conjunto de datos es (linealmente) *separable* si existen parámetros (w^*, w_0^*) tales que

$$t^{(i)} = 1 \implies w^* x^{(i)} + w_0^* > 0, \qquad t^{(i)} = 0 \implies w^* x^{(i)} + w_0^* < 0,$$

para todo i. En otras palabras, un único límite afín clasifica todos los ejemplos correctamente con margen positivo.

Ítem 1: Por qué los pesos logísticos no regularizados tienden a infinito para datos separables (demostración).: Sea $t^{(i)} \in \{\pm 1\}$ las etiquetas con signo (donde $t = 2y - 1$). Asumamos separabilidad y sea

$$m := \min_i \big(t^{(i)}(w^* x^{(i)} + w_0^*)\big) > 0$$

el margen positivo más pequeño logrado por algún (w^*, w_0^*) separador. Consideremos escalar (w^*, w_0^*) por $\alpha > 0$: $(w(\alpha), w_0(\alpha)) = (\alpha w^*, \alpha w_0^*)$. Entonces para cada i

$$t^{(i)}\big(w(\alpha)x^{(i)} + w_0(\alpha)\big) \geq \alpha m.$$

Usando la pérdida logarítmica por ejemplo en la forma $t \in \{\pm 1\}$,

$$\ell_i(\alpha) = \log\big(1 + e^{-t^{(i)}(w(\alpha)x^{(i)} + w_0(\alpha))}\big) \leq \log\big(1 + e^{-\alpha m}\big).$$

Sumando sobre i da

$$\ell\big(w(\alpha), w_0(\alpha)\big) \leq N \log\big(1 + e^{-\alpha m}\big).$$

Cuando $\alpha \to \infty$, $e^{-\alpha m} \to 0$ y por lo tanto $\ell\big(w(\alpha), w_0(\alpha)\big) \to 0$. Por lo tanto, el ínfimo de la pérdida es 0, y se aproxima en el límite $\|(w, w_0)\| \to \infty$. En consecuencia, el problema de máxima verosimilitud no regularizado no tiene un minimizador finito: el optimizador siempre puede reducir la pérdida escalando los parámetros arbitrariamente grandes, por lo que los pesos divergen al infinito.

Ítem 2: Cómo la regularización ℓ_1 y ℓ_2 previenen esto y sus efectos cualitativos.: Se añade una penalización para obtener un objetivo regularizado.

Regularización ℓ_2 (Ridge).

$$\ell_{\ell_2}(w, w_0) = \ell(w, w_0) + \frac{\lambda}{2}\big(\|w\|_2^2 + w_0^2\big), \qquad \lambda > 0.$$

La penalización cuadrática crece como $\|w\|_2^2$ cuando $\|w\| \to \infty$, por lo que el objetivo regularizado tiende a $+\infty$ para normas grandes. Esto hace que el objetivo general sea coercivo y asegura la existencia de un único minimizador finito. Efecto: los coeficientes se reducen suavemente hacia cero (pequeños pero típicamente no nulos).

Regularización ℓ_1 (Lasso).

$$\ell_{\ell_1}(w, w_0) = \ell(w, w_0) + \lambda\big(\|w\|_1 + |w_0|\big), \qquad \lambda > 0.$$

La penalización de valor absoluto también crece sin límite cuando $\|w\| \to \infty$, por lo que existe un minimizador finito. Debido a que ℓ_1 no es diferenciable en cero, a menudo produce soluciones dispersas (algunos coeficientes exactamente cero), lo que puede interpretarse como selección de características.

Gradiente regularizado (como referencia).: Con penalización ℓ_2:

$$\frac{\partial \ell_{\ell_2}}{\partial w} = \frac{\partial \ell}{\partial w} + \lambda w, \qquad \frac{\partial \ell_{\ell_2}}{\partial w_0} = \frac{\partial \ell}{\partial w_0} + \lambda w_0.$$

Con penalización ℓ_1, usar subgradientes:

$$0 \in \frac{\partial \ell}{\partial w} + \lambda\, \partial \|w\|_1,$$

lo que admite soluciones con algunos componentes de w exactamente cero cuando se cumple la condición del subgradiente.

II. Parte Aplicada

A. Introducción

Esta sección presenta la implementación práctica de métodos de clasificación para el conjunto de datos de Reconocimiento de Actividad Humana. El flujo de trabajo incluye la extracción de características de los datos brutos de series temporales de sensores utilizando dos bibliotecas: `PyTS`, basada en gráficos de recurrencia, y `TsFresh`, que realiza ingeniería de características automatizada para series temporales. Las características extraídas se utilizan luego para entrenar y evaluar múltiples clasificadores con el fin de comparar su rendimiento.

La implementación se llevó a cabo en Python y está disponible en el siguiente repositorio: Proyecto en GitHub.

B. Conjunto de Datos

El conjunto de datos utilizado en este proyecto es el conjunto de datos de Reconocimiento de Actividad Humana (HAR), que contiene señales de series temporales multivariadas registradas desde sensores de smartphones. Cada observación corresponde a una ventana de 2.56 segundos de datos del sensor, capturada con una frecuencia de muestreo de 50Hz. Los sensores incluyen un acelerómetro triaxial (aceleración corporal y total) y un giroscopio triaxial, resultando en nueve señales brutas de entrada.

El conjunto de datos se proporciona en dos archivos: `train.h5` y `test.h5`. El conjunto de entrenamiento contiene 7352 muestras con sus correspondientes etiquetas de actividad, mientras que el conjunto de prueba contiene 2947 muestras sin etiquetar reservadas para evaluación. Cada muestra está estructurada como un arreglo tridimensional de forma (128×9), donde 128 representa el número de pasos de tiempo en cada ventana y 9 corresponde a los canales del sensor. Las etiquetas cubren seis clases de actividad: *caminando*, *subiendo escaleras*, *bajando escaleras*, *sentado*, *de pie* y *acostado*.

El conjunto de datos presenta un escenario desafiante pero atractivo para la clasificación: aunque las seis actividades están bien definidas, las señales brutas de series temporales multivariadas son complejas y de alta dimensionalidad. Esta complejidad dificulta la aplicación directa de clasificadores tradicionales, y motiva la necesidad de métodos de extracción de características que puedan revelar patrones más discriminativos. En este trabajo, se investigaron dos enfoques: gráficos de recurrencia usando `PyTS`, que transforman las señales en imágenes capturando dinámicas temporales, e ingeniería de características automatizada con `TsFresh`, que produce un gran conjunto de descriptores estadísticos de las series temporales.

C. Metodología

La metodología de este proyecto se centra en analizar cómo las diferentes estrategias de extracción de características influyen en el rendimiento de la clasificación en el conjunto de datos de Reconocimiento de Actividad Humana. Los datos brutos de series temporales multivariadas son de alta dimensionalidad y difíciles de procesar directamente, por

lo que se investigaron dos enfoques: transformaciones de gráficos de recurrencia con la biblioteca `PyTS`, y extracción de características estadísticas automatizada con la biblioteca `TsFresh`. Estos métodos fueron seleccionados porque representan dos perspectivas complementarias: uno convierte las series temporales en imágenes que enfatizan las dinámicas temporales, mientras que el otro produce descriptores numéricos que resumen propiedades estadísticas de las señales.

Para evaluar la calidad de las características extraídas, se aplicó un conjunto de clasificadores conocidos utilizando las bibliotecas `scikit-learn` y `xgboost`. El propósito de esta evaluación comparativa no era estudiar los modelos en detalle en sí mismos, sino obtener evidencia empírica de cuán efectivamente cada método de extracción de características respalda la clasificación. Esta evaluación tuvo dos objetivos principales: (i) comparar PyTS y TsFresh en términos de su capacidad para producir características discriminativas, y (ii) identificar qué clasificadores logran un rendimiento competitivo y, por lo tanto, son buenos candidatos para la implementación manual posterior en el proyecto.

Los experimentos se llevaron a cabo bajo condiciones controladas con semillas aleatorias fijas para garantizar la reproducibilidad, y el rendimiento se midió utilizando precisión (accuracy), puntuación F1 ponderada y matrices de confusión. Estas métricas permitieron una visión equilibrada tanto de la corrección general como del comportamiento específico por clase.

Los resultados experimentales confirmaron que `TsFresh` alcanzó un mejor desempeño global en la extracción de características, al proporcionar representaciones más discriminativas para la clasificación. Entre los modelos evaluados, los que lograron mayor rendimiento fueron `XGBoost` (*accuracy*: 0.98, *F1*: 0.98) y `Random Forest` (*accuracy*: 0.97, *F1*: 0.97). Sin embargo, para las siguientes etapas del proyecto se optó por trabajar con `Decision Tree` (*accuracy*: 0.94, *F1*: 0.94) y `K-Nearest Neighbors (KNN)` (*accuracy*: 0.95, *F1*: 0.95). Esta decisión se basó en la simplicidad y comodidad de implementación de estos algoritmos, que aun así alcanzaron porcentajes de predicción elevados y resultados satisfactorios para el problema planteado.

III. Implementación

Se implementó ambos algoritmos elegidos, con el fin de poder hacer la experimentación, predicción y análisis. Repositorio en GitHub.

IV. Experimentation

A. Decision Tree

La implementación manual del algoritmo `Decision Tree` alcanzó una exactitud global de 0.92 y una puntuación F1 ponderada de 0.92, lo que representa un desempeño sólido y competitivo frente a los modelos evaluados en fases previas. El informe de clasificación muestra resultados consistentes en todas las clases. En particular, las actividades dinámicas como "caminando" (clase 0), "subiendo escaleras" (clase 1) y "bajando escaleras" (clase 2) lograron métricas equilibradas, con F1 de 0.86, 0.85 y 0.89 respectivamente. Por otro lado, las actividades estáticas exhibieron rendimientos aún más altos: "sentado" (clase 3) y "de pie" (clase 4) alcanzaron valores de F1 cercanos a 0.95, mientras que "acostado" (clase 5) fue clasificada de manera perfecta, con precisión, recall y F1 de 1.00.

El desempeño global de 0.92 pone de manifiesto la capacidad del árbol de decisión para manejar tanto actividades dinámicas como posturas estáticas, logrando un balance notable entre clases. La matriz de confusión refleja una baja tasa de errores, con la mayoría de las equivocaciones concentradas en la diferenciación entre las clases dinámicas (0, 1 y 2). En contraste, las posturas estáticas se reconocieron casi sin ambigüedad, lo que confirma que el modelo capta adecuadamente los patrones característicos de este tipo de señales.

B. KNN

La implementación del algoritmo `KNN` alcanzó una exactitud global de 0.53 y una puntuación F1 ponderada de 0.45. Estos valores reflejan un rendimiento limitado en comparación con las implementaciones utilizadas en experimentos previos. El informe de clasificación reveló un comportamiento heterogéneo según la clase: las actividades dinámicas como "caminando" (clase 0) y "subiendo escaleras" (clase 1) lograron métricas aceptables, con F1 de 0.83 y 0.61 respectivamente. De manera similar, la clase "bajando escaleras" (clase 2) mostró un recall alto (0.88) y un F1 de 0.65, lo que indica que el modelo identifica bien este tipo de actividad. En contraste, las actividades estáticas, particularmente "sentado" (clase 3) y "de pie" (clase 4), obtuvieron resultados muy bajos, con F1 de apenas 0.11 y 0.08 respectivamente, mientras que "acostado" (clase 5) alcanzó un desempeño intermedio con F1 de 0.54

El desempeño global de 0.53 de `KNN` pone en evidencia limitaciones importantes del enfoque cuando no se aplican técnicas de preprocesamiento ni ajustes más finos. El patrón observado en el informe de clasificación muestra que el modelo es capaz de reconocer de manera más confiable actividades con señales dinámicas y repetitivas, como caminar o subir y bajar escaleras, pero tiene serias dificultades para diferenciar posturas estáticas. Esto se refleja en la matriz de confusión, donde las clases 3 y 4 presentan un gran número de errores de clasificación, siendo común que se confundan con la clase 5.

V. Discusión

La implementación manual de `Decision Tree` alcanzó grandes resultado, lo que refleja un desempeño competitivo en comparación con implementaciones más sofisticadas. Desde el punto de vista teórico, este resultado está en línea con las ventajas conocidas de los árboles de decisión: su capacidad para realizar divisiones jerárquicas en el espacio de características les permite manejar datos heterogéneos y capturar patrones tanto de actividades dinámicas como de posturas estáticas. La matriz de confusión mostró un bajo nivel de errores y una clasificación casi perfecta de las posturas, lo que confirma que

el modelo logra identificar reglas claras que diferencian estos estados de manera robusta.

La implementaciónde KNN, por otro lado, no alcanzo unos resultados muy positivos. Este rendimiento limitado coincide con lo que predice la teoría: KNN es un algoritmo altamente dependiente del preprocesamiento de datos, sensible a la escala de las variables y a la elección de hiperparámetros como el valor de k o la métrica de distancia. La fuerte confusión entre clases posturales ("sentado", "de pie" y "acostado") confirma que, sin una adecuada normalización o un conjunto enriquecido de características, el algoritmo no logra capturar diferencias sutiles en señales de baja variabilidad. Esto se traduce en un desempeño inestable entre clases, con un sesgo hacia actividades dinámicas que generan patrones más diferenciados en los sensores.

VI. Conclusiones

El presente trabajo permitió implementar y comparar diferentes enfoques de clasificación aplicados al problema de Reconocimiento de Actividad Humana. A partir de la extracción de características mediante PyTS y TsFresh, se observó que este último produjo un mejor rendimiento de los clasificadores evaluados. Entre los modelos probados, XGBoost (accuracy: 0.98, F1: 0.98) y Random Forest (accuracy: 0.97, F1: 0.97) lograron los resultados más altos, confirmando el potencial de los métodos de ensamble para capturar patrones complejos en datos de series temporales multivariadas. Sin embargo, por motivos de simplicidad y facilidad de implementación, se optó por continuar el análisis con Decision Tree y KNN, que aun así obtuvieron métricas competitivas (accuracy y F1 entre 0.94 y 0.95).

Las implementaciones manuales demostraron las diferencias esperadas desde la teoría. El KNN alcanzó apenas una exactitud del 0.53 y un F1 ponderado de 0.45, esto evidencio su fuerte dependencia de la escala de los datos, la selección de hiperparámetros y la calidad de las características. Por otro lado, el Decision Tree manual alcanzó un desempeño sólido con 0.92 en accuracy y F1, evidenciando su habilidad para construir reglas estructuradas que separan con claridad las actividades en movimiento de las posturas estáticas.

Como recomendaciones para trabajos futuros, se sugiere profundizar en la optimización de hiperparámetros y explorar métricas de distancia alternativas para KNN.

VII. Anexos

- Repositorio en GitHub.