SEDE: ORELLANA

FACULTAD: INFORMATICA Y ELECTRÓNICA

**CARRERA:** INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

**ASIGNATURA:** MINERIA DE DATOS

**PAO:** 6

**PARALELO:** A

## INFORME DEL PROYECTO

1. **DATOS GENERALES:**

**NOMBRES:** **CODIGOS:**

Loja Mancheno Angel Gino 2861

Maita Torres Tania Esperanza 2864

**FECHA DE REALIZACIÓN FECHA DE ENTREGA:**

11/12/2023 11/12/2023

1. **Tema:**

**Despliegue de un Modelo Predictivo para la Detección Temprana de la Enfermedad “Moniliophthora perniciosa” en Plantaciones de Cacao**

1. **Objetivo General**

Desarrollar y desplegar un Modelo Predictivo utilizando técnicas de modelado avanzadas y una arquitectura de API REST con el fin de detección temprana de la Enfermedad “Moniliophthora perniciosa” en Plantaciones de Cacao.

1. **Objetivos específicos:**

* Realizar una revisión bibliográfica mediante la búsqueda en bases de datos científicas que permitirá conocer las tendencias y enfoques de técnicas de modelado, arquitectura API REST y el estudio de la enfermedad "Moniliophthora perniciosa".
* Diseñar e implementar un modelo predictivo utilizando técnicas de machine learning capaz de predecir la presencia o ausencia de la enfermedad.
* Implementar una API REST empleando Fast API que permita el despliegue del modelo predictivo entrenado.
* Desarrollar una aplicación híbrida mediante arquitectura Hexagonal para el consumo de la API.
* Realizar pruebas de funcionamiento a la API REST, mediante la simulación de consultas, para analizar su rendimiento.

1. **Introducción**

La implementación de un modelo predictivo de vanguardia usando la biblioteca de Scikit-learn, anclado a una evaluación comparativa entre diversas categorías de algoritmos de Machine Learning, agrupándolos en cuatro conjuntos distintos entre ellas técnicas de la reducción de la dimensionalidad, tendencia de regulación, clasificación de valores atípicos con regresiones robustas y métodos de ensamble, con el fin de que el modelo predictivo se enfrente a desafíos relacionados con la complejidad o la simplicidad.

La fusión de datos será un componente esencial en el análisis, donde se integrarán los registros manuales de las fichas con los datos de los sensores. El análisis comprenderá una descripción detallada de las características de los datos, por lo tanto, al construir el modelo predictivo, se debe realizar la limpieza y normalización de los datos.

1. **Desarrollo**

La fusión de datos se la realizo a partir de los datos obtenidos de parte de sensores y las fichas que se caracterizan por una recolección de datos de manera manual, permitiendo una interpretación más completa y detallada de los patrones y correlaciones identificados entre las variables ambientales y la incidencia de la enfermedad. La técnica que se empleó para lograr dicha fusión fue la de Agregación la cual consiste en agregar, es decir, concatena datos de diferentes fuentes [3]. Dando como resultado un conjunto de datos con las siguientes variables de entrada y una de salida ver la **Table 1**:

**Table 1.** variables de estudio

|  |  |
| --- | --- |
| Variables de entrada | variable de salida |
| Fruto | Incidencia |
| Severidad |
| Rain |
| Temperature |
| RH |
| Dew Point |
| Wind Speed |
| Gust Speed |
| Wind Direction |

Cabe destacar que no es aconsejable tener un exceso de variables ya que algunas pueden resultar irrelevantes, esto podría generar valores faltantes, introduciendo sesgos significativos y comprometiendo la exactitud del modelo predictivo. Por esta razón, se implementan técnicas de reducción de dimensionalidad, para lo cual se ha empleado los algoritmos PCA, IPCA y KPCA. Este último incluye tres kernels comunes: lineal, polinomial y gaussiano. Así mismo, se tomará la decisión de determinar cuál de estos métodos es el óptimo para el análisis.

  # Configuramos los datos de entrenamiento

        dt\_train = ipca.transform(X\_train)

        dt\_test = ipca.transform(X\_test)

        # Mandamos los data frames la la regresión logística

        logistic.fit(dt\_train, y\_train)

        # Calculamos nuestra exactitud de nuestra predicción

        # print("SCORE IPCA: ", logistic.score(dt\_test, y\_test))

        # print("SCORE IPCA: ", logistic.score(dt\_test\_ipca, y\_test),variables\_artificiales)

        if logistic.score(dt\_test, y\_test) > mayor\_ipca:

            mayor\_ipca = logistic.score(dt\_test, y\_test)

            variables\_ipca = variables\_artificiales

    # print(mayor\_pca, mayor\_ipca)

    # print(variables\_pca, variables\_ipca)

    print(

        "pca: ", [mayor\_pca, variables\_pca],

        "ipca: ", [mayor\_ipca, variables\_ipca]

    )

Evaluaremos dichos algoritmos con diferentes números de componentes también conocidas como variables artificiales, la elección de la cantidad óptima de componentes dependerá del número inicial de variables de entrada y estará orientada a preservar la máxima información relevante mientras se reduce la dimensionalidad del conjunto de datos.

**Table 2.** Algoritmos de reducción de dimensionalidad con datos normalizados

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algoritmos** | | **Resultado** | **Total de componentes Iterados** | **N componentes** | **Conclusión** |
| **PCA** | | 0.975041597 | 7 | 7 | Al analizar estos resultados se concluye que tanto el algoritmo PCA, IPCA y KPCA Linear obtienen el mejor y mismo resultado con el mismo número de componentes |
| **IPCA** | | 0.975041597 | 7 |
| **KPCA** | **Linear** | 0.975041597 | 7 |
| **Poly** | 0.961730449 | 7 |
| **Rbf** | 0.940099834 | 7 |

**Table 3.** Algoritmos de reducción de dimensionalidad con datos discretizados

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algoritmos** | | **Resultado** | **Componentes iterados** | **N componentes** | **Conclusión** |
| PCA | | 0.985024958 | 7 | 4 | Al analizar estos resultados se concluye que tanto el algoritmo PCA, IPCA y KPCA Linear, Poly obtienen el mejor y mismo resultado sin embargo el número de componentes es distinto |
| IPCA | | 0.985024958 | 4 |
| KPCA | Linear | 0.985024958 | 7 |
| Poly | 0.985024958 | 7 |
| Rbf | 0.96672213 | 3 |

Basándonos en los resultados obtenidos de los experimentos, se concluye que, al emplear datos normalizados, PCA, IPCA y KPCA lineal proporcionan resultados más favorables. Por lo tanto, se tiene la opción de seleccionar y utilizar cualquiera de estos tres algoritmos con el número de componentes necesarios para el proceso de entrenamiento de un modelo predictivo.

**Tendencia de regulación**

Continuando con las estrategias para evitar complicaciones relacionadas con la complejidad y simplicidad de los modelos, nos adentramos en la tendencia de regulación. Este enfoque implica la imposición de penalizaciones sobre las features que no contribuyen de manera positiva, es decir, que restan información al modelo predictivo. Dentro de esta técnica, destacan algoritmos como Lasso, Ridge y ElasticNet. Cada uno de ellos se evaluarán para determinar cuál es el óptimo, permitiendo seleccionar la técnica de regulación más adecuada para mantener y mejorar la exactitud predictiva del modelo.

import pandas as pd

import sklearn

# Importamos los modelos de sklearn

from sklearn.linear\_model import LinearRegression

from sklearn.linear\_model import Lasso

from sklearn.linear\_model import Ridge

from sklearn.linear\_model import ElasticNet

# Importamos las metricas de entrenamiento y el error medio cuadrado

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error  # error medio cuadrado

from sklearn.preprocessing import StandardScaler  # Normalizar los datos

if \_\_name\_\_ == "\_\_main\_\_":

    # Importamos el dataset del 2017

    dataset = pd.read\_csv('./data/DATASET.csv')

    # Mostramos el reporte estadistico

    # print(dataset.describe())

    # Reporte de la felicidad mundial (índice de corrupción)

    # Vamos a elegir los features que vamos a usar

    X = dataset[['FRUTO', 'SEVERIDAD', 'Temperature', 'RH',

                 'DewPoint', 'WindSpeed', 'GustSpeed', 'WindDirection']]

    # Definimos nuestro objetivo, que sera nuestro data set, pero solo en la columna

   # score

    y = dataset[['INCIDENCIA']]

    # Imprimimos los conjutos que creamos

    # En nuestros features tendremos definidos 155 registros, uno por cada pais, 7

    # colunas 1 por cada pais

    # print(X.shape)

    # Y 155 para nuestra columna para nuestro target

    # print(y.shape)

    # Aquí vamos a partir nuestro entrenaminto en training y test, no hay olvidar el

    # orden

    # Con el test size elejimos nuestro porcetaje de datos para training

    X = StandardScaler().fit\_transform(X)  # Normalizamnos los datos

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, *test\_size*=0.25)

    # Aquí definimos nuestros regresores uno por 1 y llamamos el fit o ajuste

    modelLinear = LinearRegression().fit(X\_train, y\_train)

**Table 4.** evaluación de features con tendencia de regulación

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algoritmos** | **Resultado** | **Features** | **Score** | **Análisis** |
| **Lasso** | 0 | Fruto | 0.90977943 | Con los resultados obtenidos se observa que, una variable con el coeficiente Lasso es la que tiene mayor peso |
| 0.01043325 | Severidad |
| 0 | Temperature |
| 0 | RH |
| 0 | Dew Point |
| 0 | Wind Speed |
| 0 | Gust Speed |
| 0 | Wind Direction |
| Ridge | 1.8589115 | Fruto | 0.91116245 | Existen 4 variables con mayor peso Fruto, Severidad, Dew Point, Gust Speed |
| 1.05314035 | Sevridad |
| -1.93008979 | Temperature |
| -4.75459511 | RH |
| 1.50801143 | Dew Point |
| -1.41391233 | Wind Speed |
| 3.42225679 | Gust Speed |
| -3.9139604 | Wind Direction |
| ElasticNet | 0 | Fruto | 0.90965721 | En los coeficientes de las variables de ElastiNet solo se observa una con mayor peso |
| 0.01026525 | Sevridad |
| 0 | Temperature |
| 0 | RH |
| 0 | Dew Point |
| 0 | Wind Speed |
| 0 | Gust Speed |
| 0 | Wind Direction |

Podemos concluir que el algoritmo Ridge ha demostrado ser el más eficaz para el conjunto de datos normalizados. Con un score de 0.91116245, Ridge destaca al asignar pesos significativos a cuatro variables: Fruto, Severidad, Dew Point y Gust Speed. Este enfoque de regulación logra un equilibrio al considerar múltiples features, preservando la información relevante y brindando una mayor capacidad predictiva en comparación con Lasso y ElasticNet, que asignaron mayor peso solo a una variable.

Ahora analizaremos el resultado con los datos discretizados, el resumen del experimento se muestra en la siguiente tabla:

**Table 5.** evaluación de features con tendencia de regulación

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algoritmos** | **Resultado** | **Features** | **Score** | **Aanalisis** |
| **Lasso** | 0 | Fruto | 0.89493911 | Con los resultados obtenidos se observa que, una variable con el coeficientes Lasso es la que tiene mayor peso |
| 0.18738285 | Sevridad |
| 0 | Temperature |
| 0 | RH |
| 0 | Dew Point |
| 0 | Wind Speed |
| 0 | Gust Speed |
| 0 | Wind Direction |
| Ridge | 0.00208983 | Fruto | 0.89493883 | Existen 4 variables con mayor peso Fruto, Severidad, Dew Point, Gust Speed |
| 0.25198558 | Sevridad |
| -0.01357181 | Temperature |
| -0.01231162 | RH |
| -0.00850912 | Dew Point |
| -0.03196937 | Wind Speed |
| 0.02591034 | Gust Speed |
| -3.9139604 | Wind Direction |
| ElasticNet | 0 | Fruto | 0.48814893 | En los coeficientes de las variables de Elastinect solo se obseva una con mayor peso |
| 0.08119701 | Sevridad |
| 0 | Temperature |
| 0 | RH |
| 0 | Dew Point |
| 0 | Wind Speed |
| 0 | Gust Speed |
| 0 | Wind Direction |

**Valores atípicos con regresiones robustas**

La identificación de valores atípicos nos puede realizarse a través de la visualización, aplicando fórmulas matemáticas al conjunto de datos o empleando métodos estadísticos. En este contexto, se explorará la detección de valores atípicos de manera visual con un diagrama de caja y bigote en donde se analizará la variable WindSpeed, facilitando una comprensión más intuitiva y efectiva de la presencia de valores atípicos en el conjunto de datos.

import pandas as pd

import warnings

import matplotlib.pyplot as plt

from sklearn.linear\_model import (

    RANSACRegressor, HuberRegressor

)

# Modelo de Máquinas de soporte de vectores, sub modelo La

# Regresión de Vectores de Soporte

from sklearn.svm import SVR

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

from sklearn.preprocessing import StandardScaler  # Normalizar los datos

if \_\_name\_\_ == "\_\_main\_\_":

    dataset = pd.read\_csv('./data/DATASET.csv')

    print(dataset.head(5))

    X = dataset.drop(['FRUTO' ,'INCIDENCIA', 'RH'], *axis*=1)

    y = dataset[['INCIDENCIA']]

    X = StandardScaler().fit\_transform(X)  # Normalizamnos los datos

    #y = StandardScaler().fit\_transform(y)  # Normalizamnos los datos

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,

*test\_size*=0.3, *random\_state*=42)

    # Dataset de la felicidad

    estimadores = {

        'SVR': SVR(*gamma*='auto', *C*=1.0, *epsilon*=0.1),

        'RANSAC': RANSACRegressor(),

        'HUBER': HuberRegressor(*epsilon*=1.35)

    }

    warnings.simplefilter("ignore")

    for name, estimator in estimadores.items():

        # entrenamiento

        estimator.fit(X\_train, y\_train)

        # predicciones del conjunto de prueba

        predictions = estimator.predict(X\_test)

        print("="\*64)

        print(name)

        # medimos el error,datos de prueba y predicciones

        print("MSE: "+"%.10f" % float(mean\_squared\_error(y\_test,

                                                         predictions)))

        plt.ylabel('Predicted Score')

        plt.xlabel('Real Score')

        plt.title('Predicted VS Real')

        plt.scatter(y\_test, predictions)

        plt.plot(predictions, predictions, 'r--')

        plt.show()

**Fig. 2.** Valores atípicos en WindSpeed

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

Al observar el diagrama de bloxpot, se puede identificar de inmediato que los valores que superan el valor de 0.19 se comportan como atípicos, por ende, al estar presentes en nuestros datos, esto podría originar sesgos significativos en el modelo predictivo, para abordar este problema específico se lo hace a través de las regresiones robustas, en donde Sci-kit learn nos brinda RANSAC y Huber Regressor.

**Table 6.** Comparación de algoritmos de regresiones robustas con datos normalizados

|  |  |  |
| --- | --- | --- |
| **Algoritmos** | **Score** | **Análisis** |
| Ransac | 0.3344425957 | El modelo RANSAC muestra  un puntaje mayor a Huber |
| Huber | 0.0271937894 | Huber presenta un puntaje menor y a la vez prometedor. |

En los algoritmos de regresiones robustas, Huber ha demostrado un desempeño superior, obteniendo un valor de 0.0271, donde un resultado menor se considera más favorable.

Ahora se realizará lo mismo, pero con datos discretizados para poder comparar con método es donde obtengo el mejor resultado.

**Table 7.** Comparación de algoritmos de regresiones robustas con datos normalizados

|  |  |  |
| --- | --- | --- |
| **Algoritmos** | **Score** | **Aanalisis** |
| Ransac | 0.3344426 | El modelo RANSAC muestra  un puntaje mayor a Huber |
| Huber | 0.0266223 | Huber presenta un puntaje menor lo cual hace superior a Ransac |

**Métodos de ensamble**

En este análisis, se evaluará y comparará el desempeño de Dos de los enfoques de ensamble más populares son Bagging y Boosting. Los resultados permitirán determinar qué enfoque de ensamble funciona mejor para el conjunto de datos, así como qué algoritmos individuales obtienen mayor precisión al ser combinados. A través de esta evaluación comparativa, se seleccionará la mejor estrategia de ensemble learning para la tarea predictiva en cuestión**.**

**Bagging**

En este método, Bagging entrena cada modelo en un subconjunto distinto de los datos originales y luego los agrega a través de votación mayoritaria. A continuación, se presenta una descripción de esta estrategia, utilizando diferentes clasificadores base, incluyendo K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Regresión Logística, Árboles de Decisión y además se pretende encontrar el mejor estimador para cada clasificador en el conjunto de ensamble de bagging.

 estimators = {

        'LogisticRegression': LogisticRegression(),  # 10

        'SVC': SVC(),  # 10

        'LinearSVC': LinearSVC(),

        'SGD': SGDClassifier(*loss*="hinge", *penalty*="l2", *max\_iter*=5),

        'KNN': KNeighborsClassifier(),

        'DecisionTreeClf': DecisionTreeClassifier(),

        'RandomTreeForest': RandomForestClassifier(*random\_state*=0)

    }

    estimators\_mayor = {

        'LogisticRegression': {"n\_estimators": 0,

                               "valor": 0},

        'SVC': {"n\_estimators": 0,

                "valor": 0},

        'LinearSVC': {"n\_estimators": 0,

                      "valor": 0},

        'SGD': {"n\_estimators": 0,

                "valor": 0},

        'KNN': {"n\_estimators": 0,

                "valor": 0},

        'DecisionTreeClf': {"n\_estimators": 0,

                            "valor": 0},

        'RandomTreeForest': {"n\_estimators": 0,

                             "valor": 0} # Nuevo algoritmo

    }

    estimators\_item = range(2, 100, 2)

    for name, estimator in estimators.items():

        for i in estimators\_item:

            bag\_class = BaggingClassifier(*base\_estimator*=estimator,

*n\_estimators*=i).fit(X\_train, y\_train)

            bag\_predict = bag\_class.predict(X\_test)

            # print('='\*64)

            # print('SCORE Bagging with {} : {}'.format(name,

            # accuracy\_score(bag\_predict, y\_test)))

            if accuracy\_score(bag\_predict, y\_test) > estimators\_mayor[name]["valor"]:

                estimators\_mayor[name]["valor"] = accuracy\_score(

                    bag\_predict, y\_test)

                estimators\_mayor[name]["n\_estimators"] = i

    # print(estimators\_mayor)

    for name, estimator in estimators\_mayor.items():

        print(name, estimator)

**Table 8.** clasificadores de baggin

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algoritmo** | **Clasificador** | **Total de componentes Iterados** | **n\_stimator** | **score** |
| BaggingClassifier | LogisticRegression | 300 | 36 | 0.8356546 |
| SVC | 4 | 0.6908078 |
| LinearSVC | 16 | 0.84401114 |
| SGD | 54 | 0.69637883 |
| KNN | 2 | 0.7994429 |
| DecisionTreeClf | 6 | 0.98328691 |
| RandomTreeForest | 10 | 0.98885794 |

El mejor desempeño se observa con RandomTreeForest compuesto por 10 estimadores, confirmando la efectividad de los árboles de decisión dentro de los métodos de ensamble de Boosting. DecisionTreeClf y LinearSVC también alcanzaron buenos resultados mientras que SGD tuvo el puntaje más bajo.

Veremos que valores tendríamos si en ves de normalizar discretizamos los datos:

**Table 8.** clasificadores de baggin con datos discretizados

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algoritmo** | **Clasificador** | **Total de componentes Iterados** | **n\_stimator** | **score** |
| BaggingClassifier | LogisticRegression | 300 | 2 | 0.98002853 |
| SVC | 10 | 0.98145506 |
| LinearSVC | 2 | 0.9828816 |
| SGD | 2 | 0.9828816 |
| KNN | 22 | 0.97146933 |
| DecisionTreeClf | 6 | 0.9828816 |
| RandomTreeForest | 6 | 0.9828816 |

**Boosting**

A diferencia de Bagging, Boosting entrena los clasificadores débiles secuencialmente, donde cada nuevo clasificador se enfoca en las muestras que fueron difíciles de predecir por el anterior. Esto permite corregir los errores y lograr modelos más robustos. Al evaluar GradientBoosting y AdaBoost, se obtuvieron los siguientes resultados.

if \_\_name\_\_ == '\_\_main\_\_':

    dt\_heart = pd.read\_csv('./data/DATASET.csv')

    x = dt\_heart.drop(['INCIDENCIA'], *axis*=1)

    y = dt\_heart['INCIDENCIA']

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, *test\_size*=0.35, *random\_state*=1)

    # Clasificador base (puede ser cualquier clasificador débil)

    base\_classifier\_adaboost = DecisionTreeClassifier(*max\_depth*=1)

    estimators = range(2, 300, 2)

    best\_result = {'result': 0, 'n\_estimator': 1}

    ada\_best\_result = {'result': 0, 'n\_estimator': 1}

    for i in estimators:

        # Crear el modelo AdaBoost para cada iteración

        adaboost\_model = AdaBoostClassifier( *n\_estimators*=i).fit(X\_train, y\_train)

        # Ajustar el modelo Gradient Boosting

        boost = GradientBoostingClassifier(*n\_estimators*=i).fit(X\_train, y\_train)

        # Realizar predicciones para ambos modelos

        boost\_pred = boost.predict(X\_test)

        adaboost\_predic = adaboost\_model.predict(X\_test)

        # Calcular la precisión para ambos modelos

        new\_accuracy = accuracy\_score(boost\_pred, y\_test)

        adaboost\_new\_accuracy = accuracy\_score(adaboost\_predic, y\_test)

        # Actualizar resultados si es necesario

        if new\_accuracy > best\_result['result']:

            best\_result['result'] = new\_accuracy

            best\_result['n\_estimator'] = i

        if adaboost\_new\_accuracy > ada\_best\_result['result']:

            ada\_best\_result['result'] = adaboost\_new\_accuracy

            ada\_best\_result['n\_estimator'] = i

    print("Mejor resultado para Gradient Boosting:", best\_result)

    print("Mejor resultado para AdaBoost:", ada\_best\_result)

**Table 6. Comparación de Resultados entre GradientBoosting y AdaBoost**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algoritmo** | **Total de componentes Iterados** | **n\_stimator** | **score** |
| GradientBoosting | 300 | 188 | 0.98050139 |
| AdaBoost | 280 | 0.95264624 |

Estos valores representan la puntuación de precisión de los modelos entrenados con los conjuntos de datos utilizados en la evaluación. En este caso, Gradient Boosting obtuvo la puntuación más alta con 188 estimadores.

1. **BIBLIOGRAFÍA**

[1] S. Marukatat, “Tutorial on PCA and approximate PCA and approximate kernel PCA,” *Artif Intell Rev*, vol. 56, no. 6, pp. 5445–5477, Jun. 2023, doi: 10.1007/s10462-022-10297-z.

[2] B. Dettmar, C. Peltier, and P. Schlich, “Beyond principal component analysis (PCA) of product means: Toward a psychometric view on sensory profiling data,” *J Sens Stud*, vol. 35, no. 2, Apr. 2020, doi: 10.1111/joss.12555.

[3] T. M. Tang and G. I. Allen, “Integrated Principal Components Analysis,” 2021. [Online]. Available: http://jmlr.org/papers/v22/20-084.html.

[4] F. Lajmi, L. Mhamdi, W. Abdelbaki, H. Dhouibi, and K. Younes, “Investigating Machine Learning and Control Theory Approaches for Process Fault Detection: A Comparative Study of KPCA and the Observer-Based Method,” *Sensors*, vol. 23, no. 15, Aug. 2023, doi: 10.3390/s23156899.

[5] W. Chango, J. A. Lara, R. Cerezo, and C. Romero, “A review on data fusion in multimodal learning analytics and educational data mining,” *Wiley Interdiscip Rev Data Min Knowl Discov*, vol. 12, no. 4, Jul. 2022, doi: 10.1002/widm.1458.