INFORME FINAL

Introducción a la Inteligencia Artificial

"*Predicting Molecular Properties*".

Integrantes:
Andres David Medina Herrera, Miguel Arias Londoño
CC.1003562902, CC.1017272204
andres.medina2@udea.edu.co - miguel.ariasl@udea.edu.co

Facultad de Ingeniería

Universidad de Antioquia

Sede Medellín

25/10/2023

Contenido

Introducción

La constante de acoplamiento escalar es un parámetro en espectroscopia que indica la interacción entre núcleos atómicos o partículas subatómicas. En teoría de campos, física de partículas, física cuántica y demás ciencias exactas relacionadas con lo nuclear, es usado para realizar predicciones y cálculos, ya que guarda estrecha relación con la distancia entre núcleos atómicos, ángulos de enlace, configuración estereoquímica, etc.}

Actualmente existen técnicas que permiten calcular tal constante de acoplamiento, pero debido a la complejidad y costos de estos procedimientos, resultan considerablemente limitantes en los flujos de trabajo del día a día. Partiendo de ello, el presente trabajo busca obtener predicciones de la constante de acoplamiento a partir de algoritmos de inteligencia artificial. Para ello se usarán datos provenientes de kaggle, de la competición "*Predicting Molecular Properties".* (https://www.kaggle.com/competitions/champs-scalar-coupling/overview). Con los datasets presentes en la competición se espera tener la información suficiente para crear un modelo capaz de predecir la constante de acoplamiento escalar entre los diferentes pares de átomos presentes en las diferentes moléculas proporcionadas en los datos.

En base a un conjunto de datos de entrenamiento y de prueba, se busca predecir la interacción magnética entre dos átomos en una molécula es decir, la constante de acoplamiento escalar,la predicción de esta constante para entre pares de átomos en moléculas,el tipo de acoplamiento y cualquier característica que se pueda crear a partir de los archivos dados de estructura de la molécula,es el objetivo de este reto.

No se predecirá todos los pares de átomos en cada molécula, sino solo los pares que se enumeran explícitamente en los archivos de entrenamiento y prueba,además se excluye cualquier predicción de la constante escalar que involucre átomos de flúor.

El desempeño del modelo será evaluado con base en el logaritmo del error absoluto medio, se calculará para cada tipo de acoplamiento escalar y luego se promedian entre los tipos, de modo que una disminución del 1 % en MAE para un tipo proporciona la misma mejora en la puntuación que una disminución del 1 % para otro tipo.

$$score = \frac{1}{T}\sum_{t=1}^{T}\log\left(\frac{1}{n_t}\sum_{i=1}^{n_t}|y_i - \hat{y}_i|\right)$$

Dónde:

- T es el número de tipos de acoplamiento escalar
- n_t es el número de observaciones de tipo t
- y_i es la constante de acoplamiento escalar real para la observación
- $\hat{y}_i$ es la constante de acoplamiento escalar predicha para la observación

Para este caso un métrica de negocio no tiene explicación aparente,debido a que se está trabajando con estructuras moleculares y su predicción las aplicaciones para este tipo de modelos afectan desde las ciencias farmacéuticas hasta la ingeniería de alimentos o materiales,por ende se considera que si bien tiene un efecto económico no es posible definir una métrica de negocio relacionable aparentemente

Descripción y estructura de los datasets

La competición da los datos en un zip, el cual contenía los siguientes archivos:

- potential_energy.csv
- magnetic_shielding_tensors.csv
- structures.csv
- scalar_coupling_contributions.csv
- test.csv
- dipole_moments.csv
- train.csv
- sample_submission.csv
- mulliken_charges.csv

Cada uno de los datasets contiene información importante de las diferentes moléculas, tal información se encuentra dispersa en cada uno de los CSV, por lo cual se deberá entender y estudiar cada archivo, para disponer de la data adecuadamente.

Potential_energy.csv:

Este data frame contiene información sobre la energía potencial presente en la molécula.

	molecule_name	potential_energy
0	dsgdb9nsd_000001	-40.523680
1	dsgdb9nsd_000002	-56.560246
2	dsgdb9nsd_000003	-76.426077
3	dsgdb9nsd_000004	-77.335268
4	dsgdb9nsd_000005	-93.428488
5	dsgdb9nsd_000006	-114.510216

Figura 1. Dataframe potencial_energy.csv

Como se observa en la figura 1, este posee dos filas, una que identifica la molécula y la respectiva energía potencial de esta. Posee en total 130789 filas, por lo cual tendremos ese mismo número de moléculas para análisis.

magnetic_shielding_tensors.csv:

Este data frame contiene información sobre los tensores de blindaje magnéticos, los cuales físicamente hablando, en términos simples son como un “escudo” que rodea el núcleo atómico y protege a los electrones circundantes de los efectos del campo magnético externo, guardando cierta relación con la constante de acoplamiento.

	molecule_name	atom_index	XX	YX	ZX	XY	YY	ZY	XZ	YZ	ZZ
0	dsgdb9nsd_000001	0	195.3147	0.0000	-0.0001	0.0000	195.3171	0.0007	-0.0001	0.0007	195.3169
1	dsgdb9nsd_000001	1	31.3410	-1.2317	4.0544	-1.2317	28.9546	-1.7173	4.0546	-1.7173	34.0861
2	dsgdb9nsd_000001	2	31.5814	1.2173	-4.1474	1.2173	28.9036	-1.6036	-4.1476	-1.6036	33.8967
3	dsgdb9nsd_000001	3	31.5172	4.1086	1.2723	4.1088	33.9068	1.6950	1.2724	1.6951	28.9579
4	dsgdb9nsd_000001	4	31.4029	-4.0942	-1.1793	-4.0944	34.0776	1.6259	-1.1795	1.6260	28.9013
5	dsgdb9nsd_000002	0	275.6345	0.0003	0.0133	0.0003	275.6364	-0.0003	0.0161	-0.0004	237.4966

Figura 2. magnetic_shielding_tensor.csv

Este data frame contiene 2358875 filas, ya que contiene información de cada átomo presente en cada una de las moléculas.

structures.csv:

El dataframe de estructuras posee información de la molécula, como el tipo de átomos presente en esta y la ubicaciones de los mismos, al tiempo que da un índice a cada uno de los átomos respecto a la molécula.

	molecule_name	atom_index	atom	x	y	z
0	dsgdb9nsd_000001	0	C	-0.012698	1.085804	0.008001
1	dsgdb9nsd_000001	1	H	0.002150	-0.006031	0.001976
2	dsgdb9nsd_000001	2	H	1.011731	1.463751	0.000277
3	dsgdb9nsd_000001	3	H	-0.540815	1.447527	-0.876644
4	dsgdb9nsd_000001	4	H	-0.523814	1.437933	0.906397
5	dsgdb9nsd_000002	0	N	-0.040426	1.024108	0.062564
6	dsgdb9nsd_000002	1	H	0.017257	0.012545	-0.027377

Figura 3. structure.csv

Al poseer información de cada unos de los átomos de cada una de las moléculas, es de esperarse que este data frame tenga el mismo número de filas que el anterior, 2358875.

scalar_coupling_contributions.csv:

Este data frame me da información sobre las contribuciones de acoplamiento escalar, estas contribuciones se relacionan con la interacción entre los núcleos atómicos en una misma molécula, lo que proporciona información valiosa sobre la estructura y la conectividad de los átomos en la molécula, siendo entonces parámetros que no se pueden ignorar para el modelo. Entre las filas se tiene: sd, el cual se refiere al acoplamiento escalar; fc, acoplamiento escalar de contacto fermi; pso, acoplamiento escalar paramagnético spin-orbit; y el dso, acoplamiento escalar diamagnético spin-orbit.

	molecule_name	atom_index_0	atom_index_1	type	fc	sd	pso	dso
0	dsgdb9nsd_000001	1	0	1JHC	83.02240	0.254579	1.258620	0.272010
1	dsgdb9nsd_000001	1	2	2JHH	-11.03470	0.352978	2.858390	-3.433600
2	dsgdb9nsd_000001	1	3	2JHH	-11.03250	0.352944	2.858520	-3.433870
3	dsgdb9nsd_000001	1	4	2JHH	-11.03190	0.352934	2.858550	-3.433930
4	dsgdb9nsd_000001	2	0	1JHC	83.02220	0.254585	1.258610	0.272013
5	dsgdb9nsd_000001	2	3	2JHH	-11.03170	0.352932	2.858560	-3.433950
6	dsgdb9nsd_000001	2	4	2JHH	-11.03240	0.352943	2.858530	-3.433870
7	dsgdb9nsd_000001	3	0	1JHC	83.02410	0.254634	1.258560	0.272012
8	dsgdb9nsd_000001	3	4	2JHH	-11.03190	0.352943	2.858560	-3.433930
9	dsgdb9nsd_000001	4	0	1JHC	83.02430	0.254628	1.258560	0.272012
10	dsgdb9nsd_000002	1	0	1JHN	30.61160	0.059952	1.949350	0.067923

Figura 4.scalar_coupling_contributions.csv

Como se observa en la figura 4 se permutan cada uno de los átomos en la molécula con los otros, es decir que habrá tantas filas por molécula como el número de permutaciones posibles entre sus átomos, siendo en total 4659075 filas.

dipole_moments:

Este data frame posee información de la molécula y del momento dipolar de la misma descompuesto en las coordenadas cartesianas. Representa la distribución de cargas eléctricas en una molécula y la diferencia entre las cargas positivas y negativas en la misma.

	molecule_name	X	Y	Z
0	dsgdb9nsd_000001	0.0000	0.0000	0.0000
1	dsgdb9nsd_000002	-0.0002	0.0000	1.6256
2	dsgdb9nsd_000003	0.0000	0.0000	-1.8511
3	dsgdb9nsd_000004	0.0000	0.0000	0.0000
4	dsgdb9nsd_000005	0.0000	0.0000	-2.8937
5	dsgdb9nsd_000006	-2.1089	0.0000	0.0000
6	dsgdb9nsd_000007	0.0000	0.0000	0.0000

Figura 5. dipole moments.csv

Como se observa en la figura 5, el momento dipolar es dado por molécula, por ende el data frame deberá tener 130789 filas.

mulliken_charges.csv:

Este parámetro entrega información sobre el valor de distribución de carga eléctrica de una molécula, teniendo estrecha relación con el momento dipolar.

	molecule_name	atom_index	mulliken_charge
0	dsgdb9nsd_000001	0	-0.535689
1	dsgdb9nsd_000001	1	0.133921
2	dsgdb9nsd_000001	2	0.133922
3	dsgdb9nsd_000001	3	0.133923
4	dsgdb9nsd_000001	4	0.133923
5	dsgdb9nsd_000002	0	-0.707143
6	dsgdb9nsd_000002	1	0.235712
7	dsgdb9nsd_000002	2	0.235712

Figura 6. mulliken_charges.csv

La figura 6 muestra las primeras 6 filas del data frame, donde se observa el valor de la variables mencionada por cada átomo de cada molécula, es decir teniendo en total 2358875 filas, el mismo número que structures y magnetic shielding, dataframes que dan información por átomos.

train.csv y test.csv:

Train y test contiene información de las moléculas, todas las posibles permutaciones de los átomos de la misma y la variables a predecir, constante escalar de acoplamiento (train).

	id	molecule_name	atom_index_0	atom_index_1	type	scalar_coupling_constant
0	0	dsgdb9nsd_000001	1	0	1JHC	84.80760
1	1	dsgdb9nsd_000001	1	2	2JHH	-11.25700
2	2	dsgdb9nsd_000001	1	3	2JHH	-11.25480
3	3	dsgdb9nsd_000001	1	4	2JHH	-11.25430
4	4	dsgdb9nsd_000001	2	0	1JHC	84.80740
5	5	dsgdb9nsd_000001	2	3	2JHH	-11.25410
6	6	dsgdb9nsd_000001	2	4	2JHH	-11.25480
7	7	dsgdb9nsd_000001	3	0	1JHC	84.80930

Figura 7. train.csv

Test tiene la misma organización de datos que train, exceptuando evidentemente la columna de la constante escalar de acoplamiento.

Iteraciones de desarrollo

Procesamiento

Una vez visualizados cada uno de los data frame que se tenían, analizadas sus columnas, filas, significado físico de las variables que daban, análisis de la importancia de las mismas, etc., se procedió con la selección y limpieza de los datos que se usarán para el entrenamiento del modelo.

Se optó por usar todas las variables que se tienen para el proceso de predicción, para lo cual primero se deben tener los datos organizados en un solo dataframe y no de manera dispersa:

Los dataframe de potencial_energy y dipole_moments contiene información por molécula, es decir que su número de filas será el mismo, así entonces, se creó una nueva tabla con la información de las dos.

	molecule_name	dipole_X	dipole_Y	dipole_Z	potential_energy
0	dsgdb9nsd_000001	0.0000	0.0	0.0000	-40.523680
1	dsgdb9nsd_000002	-0.0002	0.0	1.6256	-56.560246
2	dsgdb9nsd_000003	0.0000	0.0	-1.8511	-76.426077
3	dsgdb9nsd_000004	0.0000	0.0	0.0000	-77.335268
4	dsgdb9nsd_000005	0.0000	0.0	-2.8937	-93.428488

(130789, 5)

Figura 8. Energía potencial y momento dipolar por molécula.

Por otro lado los dataframe magnetic_shielding_tensors.csv, mulliken_charges.csv y structures.csv poseen información de cada uno de los átomos de las diferentes moléculas, por lo cual la data de estos puede integrarse, haciendo coincidir en cada una los nombres de las moléculas, y los indices atomicos. Quedando un nuevo data frame con cada una de las columnas de los anteriores, sin repetir evidentemente las columnas referentes a los índices químicos y a la molécula perteneciente.

También se tiene que los dataframe train y scalar_coupling_contributions poseen una organización de los datos similar, como se mencionó antes, estos poseen información de las posibles permutaciones entre átomos de una misma molécula, por ende se procede igual que en los dos casos anteriores. Resultando en un nuevo data frame con el mismo número de filas que los dos que se tenían, pero con las columnas de importancia de ambos.

Debido a que tenemos información relevante acerca de cada átomo de y nuestro dataset de entrenamiento posee 2 índices atómicos,estos son los que definen la configuración de átomos presentes en esa molécula,por ende debemos tener datos relacionados de nuestros dataframes magnetic_shielding_tensors.csv, mulliken_charges.csv y structures.csv para

cada 1,por ende, se les agrega el sufijo de AT0 o AT1 para separar la información de cada átomo presente

```
df_train_merged_final.csv:
Number of rows: 4659076
Number of Columns: 41
Data types:
molecule_name               object
atom_index_0                 int64
atom_index_1                 int64
dipole_moment_x            float64
dipole_moment_y            float64
dipole_moment_z            float64
potential_energy           float64
type                        object
scalar_coupling_constant   float64
fc                         float64
sd                         float64
pso                        float64
dso                        float64
XX_AT0                     float64
YX_AT0                     float64
ZX_AT0                     float64
XY_AT0                     float64
YY_AT0                     float64
ZY_AT0                     float64
XZ_AT0                     float64
YZ_AT0                     float64
ZZ_AT0                     float64
mulliken_charge_AT0        float64
atom_AT0                    object
```

```
x_AT0                      float64
y_AT0                      float64
z_AT0                      float64
XX_AT1                     float64
YX_AT1                     float64
ZX_AT1                     float64
XY_AT1                     float64
YY_AT1                     float64
ZY_AT1                     float64
XZ_AT1                     float64
YZ_AT1                     float64
ZZ_AT1                     float64
mulliken_charge_AT1        float64
atom_AT1                    object
x_AT1                      float64
y_AT1                      float64
z_AT1                      float64
dtype: object
```

Figura 9. Dataframe de entrenamiento con toda la información procesada

```
float64    85.0%
object     10.0%
int64       5.0%
dtype: object
```

Figura 10.Porcentaje de tipos de datos en nuestro Data Frame final.

Se tienen más de 4 millones de datos,con 41 columnas de las cuales 4 de ellas son variables categóricas por lo cual el dataset cumple con las especificaciones solicitadas en el proyecto

```
4 # Select 6 random columns
5 cols = df_final_train.sample(n=6, axis=1).columns
6
7 # Replace 5% of the data in those columns with NaN
8 for col in cols:
9     df_final_train.loc[df_final_train.sample(frac=0.05).index, col] = np.nan
10
```

Figura 11. Código para eliminar el 5% de los datos en 6 columnas

```
# Calculate the percentage of missing values for each column
missing_percentages = (df_final_train_test.isnull().sum() / len(df_final_train_test) * 100).round(0).astype(str) + '%'
# Printing only the colums that have missing values
print(missing_percentages[missing_percentages != '0.0%'])
```

```
dipole_moment_z             5.0%
scalar_coupling_constant    5.0%
fc                          5.0%
sd                          5.0%
ZZ_AT0                      5.0%
ZZ_AT1                      5.0%
dtype: object
```

Figura 12.Verificación de datos faltantes

De esta manera se cumplen todos los requisitos necesarios para continuar con el proyecto.

```
# Crear el objeto LabelEncoder
encoder = ps.LabelEncoder()

# Codificar las columnas no numéricas
df_final_train_test['atom_AT1'] = encoder.fit_transform(df_final_train_test['atom_AT1'])
df_final_train_test['type'] = encoder.fit_transform(df_final_train_test['type'])
df_final_train_test['atom_AT0'] = encoder.fit_transform(df_final_train_test['atom_AT0'])
```

```
#replace the missing values with the mean of the column
df_final_train_test.fillna(df_final_train_test.mean(), inplace=True)
```

Figura 13.Conversión de variables categóricas y definición de entradas vacías por el método del promedio de la columna

Modelos supervisados:

Una vez listo el dataset se procede con la creación de los respectivos modelos supervisados, y la optimización de sus parámetros, para lo cual se optó por usar un algoritmo de RandomForetsRegressor y un DecisionTreeRegresor.

Antes de la creación de los modelos, se procedió a reemplazar los valores NaN con el promedio de los valores numéricos de la columna en la cual se encuentra, y respecto a las variables categóricas se usó método LabelEncoder de la librería sklearns, convirtiendo todo valor string en un dato numérico:

Se hizo la respectiva división entre el target y el resto de datos, es decir el ‘x’ y ‘y’.

Posteriormente se realiza una búsqueda de hiperparámetros para los modelos de árbol de decisión (DecisionTreeRegressor) y bosque aleatorio (RandomForestRegressor). Para lo cual primeramente se definen los conjuntos de parámetros para los modelos de árbol de decisión (parametros_dt) y bosque aleatorio (parámetros_rf). Estos conjuntos incluyen diversas combinaciones de hiperparámetros como la profundidad máxima (max_depth), el número mínimo de muestras para dividir un nodo (min_samples_split), y el número mínimo de muestras en una hoja (min_samples_leaf). Para luego crear los respectivos objetos de búsqueda de hiperparámetros (grid_dt y grid rf) y hacer el ajuste de los respectivos modelos.

```
# Definir los parámetros para el árbol de decisión
parametros_dt = {'max_depth': list(range(1, 20, 5)), 'min_samples_split': [2,3,5], 'min_samples_leaf': [1,2,3]}

# Definir los parámetros para el bosque aleatorio
parametros_rf = {'n_estimators': list(range(1, 100, 20)), 'max_depth': list(range(1, 20, 5)), 'min_samples_split': [2,3,5],
                 'min_samples_leaf': [1,2,3]}

# Crear los modelos
dt = DecisionTreeRegressor()
rf = RandomForestRegressor()

# Crear las búsquedas de cuadrícula
grid_dt = RandomizedSearchCV(dt, parametros_dt,scoring = 'neg_mean_squared_error', cv=5,random_state=0)
grid_rf = RandomizedSearchCV(rf, parametros_rf,scoring = 'neg_mean_squared_error',cv=5,random_state=0)

# Ajustar los modelos
grid_dt.fit(X_train, y_train)
grid_rf.fit(X_train, y_train)

# Imprimir los mejores parámetros

print("Mejores parámetros para DecisionTreeRegressor: ", grid_dt.best_params_)
print("Mejores parámetros para RandomForestRegressor: ", grid_rf.best_params_)
```

```
Rdm_forest = RandomForestRegressor(n_estimators =grid_rf.best_params_["n_estimators"]
                                   ,max_depth = grid_rf.best_params_["max_depth"],
                                   min_samples_leaf=grid_rf.best_params_["min_samples_leaf"],
                                   min_samples_split=grid_rf.best_params_["min_samples_split"],
                                   random_state=0)
Rdm_forest.fit(X_train, y_train)

DT_Regressor = DecisionTreeRegressor(
                                   max_depth = grid_dt.best_params_["max_depth"],
                                   min_samples_leaf=grid_dt.best_params_["min_samples_leaf"],
                                   min_samples_split=grid_dt.best_params_["min_samples_split"],
                                   random_state=0)
DT_Regressor.fit(X_train, y_train)
```

Ajustados los modelos, se evalúan estos entonces:

```
result_rf = evaluador_modelos("RandomForestRegressor", Rdm_forest, X_train, y_train, X_test, y_test)
result_dt = evaluador_modelos("DecisionTreeRegressor", DT_Regressor, X_train, y_train, X_test, y_test)
result_df = result_df.append(result_rf, ignore_index=True)
result_df = result_df.append(result_dt, ignore_index=True)
```

```
<ipython-input-136-67028d29a939>:3: FutureWarning: The frame.append method is deprecated and will be re
  result_df = result_df.append(result_rf, ignore_index=True)
<ipython-input-136-67028d29a939>:4: FutureWarning: The frame.append method is deprecated and will be re
  result_df = result_df.append(result_dt, ignore_index=True)
```

	Model	MAE	MSE	RMSE	R2 Score	MAE Ratio	RMSE Ratio
0	RandomForestRegressor	1.906688	7.923451	2.814863	0.994192	0.110038	0.162450
1	DecisionTreeRegressor	1.937796	8.587639	2.930467	0.993705	0.111834	0.169122

Modelos no supervisados + métodos supervisados:

Para los modelos no supervisados se optó por usar algoritmos de kmeans, es decir uso de clusterings y PCA, estos se usaron como complemento para los modelos usados

anteriormente, quedando entonces kmeans + DecisionTreeRegressor y PCA + RandomForestRegressor, así:

Primero se realiza una búsqueda de hiperparámetros utilizando la técnica de búsqueda aleatoria (RandomizedSearchCV) para ambos procesos de modelado: aquel basado en la combinación de KMeans y DecisionTreeRegressor, y otro basado en la combinación de PCA y RandomForestRegressor.

```
# Definir los parámetros para KMeans + DecisionTreeRegressor
parametros_kmeans_dt = {'kmeans__n_clusters': range(2, 10), 'dt__max_depth': list(range(20, 61, 20)),
                        'dt__min_samples_split': [2, 5, 10], 'dt__min_samples_leaf': [1, 2, 4]}

# Definir los parámetros para PCA + RandomForestRegressor
parametros_pca_rf = {'pca__n_components': range(2, 10), 'rf__n_estimators': [10, 50, 100, 200],
                     'rf__max_depth': list(range(20, 61, 20)), 'rf__min_samples_split': [2, 5, 10],
                     'rf__min_samples_leaf': [1, 2, 4]}

# Crear los pipelines
pipeline_kmeans_dt = Pipeline([
    ("kmeans", KMeans(n_init=10)),
    ("dt", DecisionTreeRegressor())
])

pipeline_pca_rf = Pipeline([
    ("pca", PCA()),
    ("rf", RandomForestRegressor())
])
```

```
# Crear las búsquedas aleatorias
random_kmeans_dt = RandomizedSearchCV(pipeline_kmeans_dt, parametros_kmeans_dt,
                                      scoring='neg_mean_squared_error', cv=5, random_state=0)
random_pca_rf = RandomizedSearchCV(pipeline_pca_rf, parametros_pca_rf,
                                   scoring='neg_mean_squared_error', cv=5, random_state=0)

# Ajustar los modelos
random_kmeans_dt.fit(X_train, y_train)
random_pca_rf.fit(X_train, y_train)

# Imprimir los mejores parámetros
print("Mejores parámetros para KMeans + DecisionTreeRegressor: ", random_kmeans_dt.best_params_)
print("Mejores parámetros para PCA + RandomForestRegressor: ", random_pca_rf.best_params_)
```

```
Mejores parámetros para KMeans + DecisionTreeRegressor:  {'kmeans__n_clusters': 4, 'dt__min_samples_spli
Mejores parámetros para PCA + RandomForestRegressor:  {'rf__n_estimators': 10, 'rf__min_samples_split': 1
```

Luego, se definen conjuntos de parámetros para ambas combinaciones de modelos (parametros_kmeans_dt y parametros_pca_rf). Estos conjuntos incluyen hiperparámetros específicos de KMeans y DecisionTreeRegressor para el primer modelo, y hiperparámetros específicos de PCA y RandomForestRegressor para el segundo modelo. Establecidos los conjuntos se crean pipelines (pipeline_kmeans_dt y pipeline_pca_rf) que combinan las etapas de procesamiento, en este caso KMeans o PCA, con los modelos DecisionTreeRegressor o RandomForestRegressor, respectivamente.

Posteriormente se crean los objetos de búsqueda de hiperparámetros (random_kmeans_dt y random_pca_rf) utilizando RandomizedSearchCV. Estos objetos utilizan los pipelines definidos y los conjuntos de parámetros especificados para buscar las mejores combinaciones de hiperparámetros. Para proceder con el ajuste usando los datos de entrenamiento y la búsqueda aleatoria de hiperparametros, para luego imprimir los parámetros encontrados del proceso de optimización.

```
# Crear nuevos modelos con los mejores parámetros
mejor_kmeans = KMeans(n_clusters=mejores_parametros_kmeans_dt['kmeans__n_clusters'])
mejor_dt = DecisionTreeRegressor(max_depth=mejores_parametros_kmeans_dt['dt__max_depth'],
                                 min_samples_split=mejores_parametros_kmeans_dt['dt__min_samples_split'],
                                 min_samples_leaf=mejores_parametros_kmeans_dt['dt__min_samples_leaf'])

mejor_pca = PCA(n_components=mejores_parametros_pca_rf['pca__n_components'])
mejor_rf = RandomForestRegressor(n_estimators=mejores_parametros_pca_rf['rf__n_estimators'],
                                 max_depth=mejores_parametros_pca_rf['rf__max_depth'],
                                 min_samples_split=mejores_parametros_pca_rf['rf__min_samples_split'],
                                 min_samples_leaf=mejores_parametros_pca_rf['rf__min_samples_leaf'])

# Ajustar los modelos
X_train_kmeans = mejor_kmeans.fit_transform(X_train)
mejor_dt.fit(X_train_kmeans, y_train)

X_train_pca = mejor_pca.fit_transform(X_train)
mejor_rf.fit(X_train_pca, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of
  warnings.warn(
```

```
RandomForestRegressor(max_depth=20, min_samples_leaf=4, min_samples_split=10,
                      n_estimators=10)
```

Figura 14. Creación de los modelos con los mejores parámetros obtenidos anteiormente.

Una vez encontrados los parámetros, se procede a ajustar los modelos utilizando las técnicas seleccionadas (KMeans y DecisionTreeRegressor, PCA y RandomForestRegressor).

Una vez ajustados se procede a evaluar los mismos:

```
# Evaluar el modelo DecisionTreeRegressor + Kmeans
resultados_dt_km = evaluador_modelos("DecisionTreeRegressor + Kmeans", mejor_dt, X_train_kmeans, y_train, mejor_kmeans.transform(X_test), y_test)

# Evaluar el modelo RandomForestRegressor + PCA
resultados_rf_pca = evaluador_modelos("RandomForestRegressor + PCA", mejor_rf, X_train_pca, y_train, mejor_pca.transform(X_test), y_test)

# Agregar los resultados al DataFrame existente
result_df = result_df.append(resultados_rf_pca, ignore_index=True)
result_df = result_df.append(resultados_dt_km, ignore_index=True)
```

```
<ipython-input-33-d392f94b6aa4>:8: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pan
  result_df = result_df.append(resultados_rf_pca, ignore_index=True)
<ipython-input-33-d392f94b6aa4>:9: FutureWarning: The frame.append method is deprecated and will be removed from pandas in a future version. Use pan
  result_df = result_df.append(resultados_dt_km, ignore_index=True)
```

Figura 15 . Evaluación de los modelos.

Además, se concatenaron los resultados de los anteriores modelos con objetivo de visualización y posterior análisis y comparación.

```
result_df
```

	Model	MAE	MSE	RMSE	R2 Score	MAE Ratio	RMSE Ratio
0	RandomForestRegressor	2.214607	56.273937	7.501596	0.957326	0.122195	0.413915
1	DecisionTreeRegressor	2.470615	64.156605	8.009782	0.951348	0.136321	0.441955
2	RandomForestRegressor + PCA	6.868078	169.727112	13.027936	0.871292	0.378959	0.718841
3	DecisionTreeRegressor + Kmeans	6.322123	158.750003	12.599603	0.879616	0.348835	0.695207

Figura 15. Métricas de evaluación de los diferentes modelos generados.

Posteriormente, se hicieron las curvas de aprendizaje para los 4 modelos generados, quedando:

Curvas de aprendizaje:

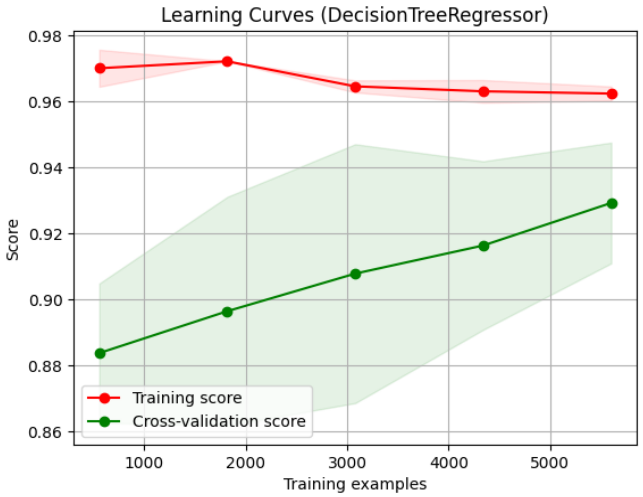

Figura 16. Curva de aprendizaje Arbol de decision

Figura 17. Curva de aprendizaje Bosque aleatorio

Nuestros 2 modelos supervisados tienen tendencias similares aunque el puntaje de los datos de validación de el árbol de decisión tienden a aumentar de manera significativa,casi lineal con respecto al conjunto de datos,se considera que el puntaje de ambos es muy bueno

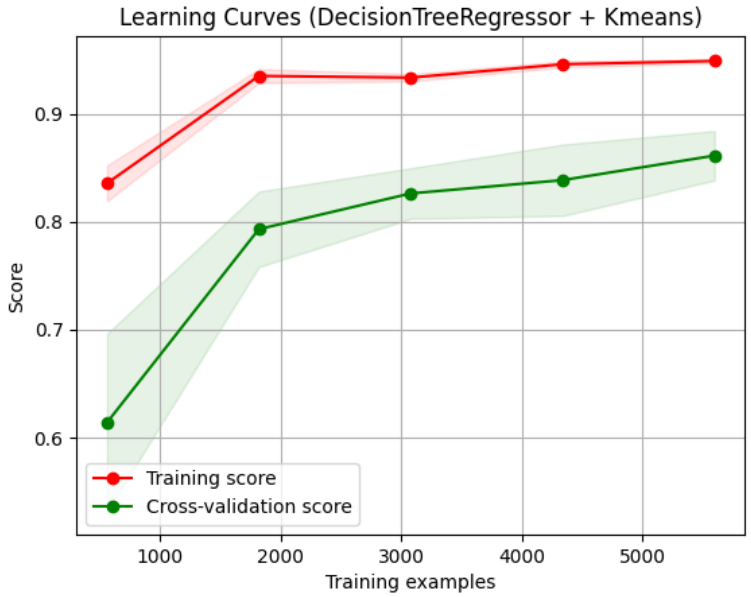

Figura 18. Curva de aprendizaje Bosque aleatorio +Kmeans

Figura 19. Curva de aprendizaje Bosque aleatorio + PCA

Si bien el uso de los algoritmos no supervisados o en este caso el “clustering de datos” y la reducción de dimensiones permite mejorar la eficiencia de los modelos y potencialmente su rendimiento,se tuvieron puntuaciones menores en estos modelos con respecto a los anteriores,esto también se ve en la tabla de resultados(fig 15),como el error medio absoluto es un poco más del doble en nuestro modelos no supervisados en el los supervisados

Retos y Consideraciones de Despliegue

Inicialmente, se planteó llevar a cabo el proyecto utilizando todas las filas del conjunto de datos, un poco más de 4 millones de registros. Sin embargo, trabajar con tal cantidad de información se convirtió en un desafío debido a las limitaciones de memoria RAM en el entorno de Google Colab. El proyecto se reiniciaba constantemente, resultando en la pérdida de cambios realizados. Aunque las tareas de limpieza y organización de los datos requerían periodos de tiempo no tan largos,los tiempos en los modelos de aprendizaje y la generación de curvas lo hacía prohibitivo en términos de tiempo y procesamiento.

Se optó por utilizar una muestra de datos más reducida para superar las dificultades mencionadas. Se decidió trabajar con exactamente 10004 filas, asegurándose de que la última molécula en el conjunto de datos estuviera completa y con información detallada sobre todos sus átomos.
Consideramos que la cantidad de datos empleados para el procesamiento es adecuada. Se cumple el requisito de tener al menos 5 mil filas y se garantiza que todas las moléculas utilizadas en el conjunto de datos tengan información completa sobre sus átomos,además se considera que para fines académicos y de aprendizaje el uso de conjuntos de datos muy grandes dificultan el proceso educativo

Observando las curvas de aprendizaje de cada uno de los modelos, es evidente los casos de overfitting, evidenciado al observarse el ajuste tan elevado en los datos; es necesario mencionar que tal comportamiento no es tan evidente en los métodos no supervisados, es decir, la introducción de métodos no supervisados que muestran una puntuación inicial más baja y luego mejoran con más datos podría indicar que los métodos no supervisados están inicialmente proporcionando información útil para la generalización del modelo.

En caso de desear mejorar el modelo se podría optar por un mejor manera de la información, es decir reducir la información sin pérdida relevante de datos, cambiar la complejidad de los respectivos modelos y hacer un análisis exhaustivo de los parámetros obtenidos en el proceso de optimización, tratando de evitar en lo posible el overfitting generado.

Conclusiones

- Limpieza y Comprensión de los Datos: La limpieza y el entendimiento precisos de los conjuntos de datos son cruciales para iniciar cualquier investigación. La calidad y organización de los datos influyen directamente en los resultados obtenidos, en nuestro caso era necesario conocer las dimensiones de los datasets para así generar un dataframe de entrenamiento correcto

- Modelos Supervisados y Ajuste de Hiper Parámetros: El uso de modelos supervisados requiere un entendimiento profundo de los datos. Los hiper parámetros influyen en la precisión del modelo; un ajuste adecuado es crucial para evitar desajustes o sobre ajustes. Buscar el equilibrio adecuado permite al modelo captar

el comportamiento de los datos sin "memorizarlos", asegurando su capacidad predictiva con nuevos datos.

- Importancia de los Métodos No Supervisados: Los métodos no supervisados son esenciales en el procesamiento de datos. Permiten presentar conjuntos de datos más compactos, destacando variables importantes para el entrenamiento. Además, facilitan la exploración y el análisis de datos sin necesidad de etiquetas previas. Estos métodos son especialmente útiles en ausencia de información de salida o para descubrir patrones ocultos en los datos.

- Análisis Detallado de Variables Relevantes: Realizar un análisis minucioso de las variables influyentes en los modelos es esencial. Este proceso puede reducir significativamente el error, ya que permite identificar y priorizar las variables que más impactan en el rendimiento de los modelos entrenados.