PREDICCIÓN DEL GASTO ENERGETICO EN EDIFICACIONES

POR:

Daniel David Sierra Moreno

MATERIA:

Introducción a la Inteligencia Artificial

PROFESOR:

Raul Ramos Pollan

UNIVERSIDAD DE ANTIOQUIA

FACULTAD DE INGENIERÍA

MEDELLÍN 2021

Contenido

INTRODUCCCIÓN

La inteligencia artificial es una herramienta que hoy en día puede ser usada en muchas aplicaciones y campos de trabajo. Es una herramienta que puede generar mucho valor a partir de los datos que tengan de algún proceso u operación. Usar estos datos para el entrenamiento de diferentes tipos de algoritmos de *Machine Learning*, permiten generar modelos que nos ayuden a observar comportamientos, reducir el tamaño de la información sin perder su significancia, o generar predicciones para la toma de decisiones en el corto, mediano o largo plazo.

En este trabajo se explorará la implementación de algoritmos de *Machine Learning* en la predicción del gasto energético en edificaciones debido a diferentes tipos de aplicaciones. Este es un tema de interés ya que reducir el costo energético no solamente implica un ahorro monetario, sino también un ahorro energético que va en línea con políticas de desarrollo sostenible ambiental.

1. Planteamiento del problema

Es importante cuantificar el gasto energético que implican los edificios debido al uso de agua caliente, vapor, electricidad, etc., ya es necesario implementar mejoras que permitan lograr mayores eficiencias energéticas para reducir el impacto ambiental y los costes. Es por esto por lo que se desea desarrollar un modelo que permita cuantificar el uso de la energía en edificios, ya que, al tener mejores estimaciones, se podrá generar un mayor interés por parte de inversores e instituciones financieras para la realización de inversiones que impliquen el ahorro energético.

1.1. Dataset

El dataset a utilizar proviene de una competencia de kaggle en la cual se proporcionan datos de tres años de lecturas en contadores de más de mil edificios en diferentes lugares del mundo. El dataset este compuesto por un conjunto de archivos .csv que proporcionan la información requerida, tal como los datos meteorológicos del lugar y los datos del edificio mismo.

El archivo que contiene los datos del edificio es nombrado como *building_meta* y contiene la siguiente información:

- ***site_id*** – Identificador para los archivos meteorológicos
- ***building_id*** – Identificador para el archivo *training.csv*
- ***primary_use*** – Identificador de la categoria principal de actividades para el edificio, basado en la clasificación de *EnergyStar property type definitions*
- ***square_feet*** – Área neta del edificio
- ***year_built*** – Año de apertura del edificio
- ***floor_count*** – Número de pisos del edificio

En cuanto al archivo con los datos meteorológicos, este se divide en dos, un archivo que será para entrenamiento de algoritmos y otro para pruebas, llamados *weather_train* y *weather_test*, respectivamente. La información que contienen ambos archivos es la siguiente:

- ***site_id***
- ***air_temperature*** – Temperatura del aire en grados Celsius
- ***cloud_coverage*** – Porción del cielo cubierto por nubes en oktas
- ***dew_temperature*** – Temperatura de rocio em grados Celsius

- ***precip_depth_1_hr*** - Milímetros
- ***sea_level_pressure*** - Millibar/hectopascales
- ***wind_direction*** – Dirección de la brujula (0-360)
- ***wind_speed*** – Metros por segundo

También hay un archivo llamado *train.csv* el cual tiene la siguiente información:

- ***building_id*** – Identificación para el archive *building_meta*.
- ***meter*** – Identificador del contador. Leído como: {0: electricidad, 1: agua fría, 2: vapor,3: agua caliente}. No todos los edificios tienen todos los tipos de contador.
- ***timestamp*** – Cuándo se hizo la medición
- ***meter_reading*** – Variable objetivo. Consumo de energía en kWh (o equivalente).

Por último, se tiene un archivo llamado *test.csv* que sirve para organizar correctamente las predicciones. Este archivo tiene la siguiente información:

- ***row_id*** – Identificador de fila en el archive de entrega
- ***building_id*** – Código de identificación del edificio
- ***meter*** – Código de identificación del medidor
- ***timestamp*** – Pasos temporales de los datos de prueba

1.2. Métrica

La métrica de evaluación principal para el modelo será el error logarítmico medio cuadrático (RMSLE), el cual se calcula mediante la siguiente expresión:

$$\in = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(log(p_i+1) - log(a_i+1)\right)^2}$$

Donde $\in$ es el valor del RMSLE, n es el número total de observaciones en el dataset, p_i es la predicción de la variable objetivo y a_i es el valor real de la variable objetivo.

Por otra parte, en cuanto a la métrica de negocio, se tiene interés en que las predicciones sean lo suficientemente confiables para saber el gasto energético de los edificios y así cuantificar los costos relacionados. Con esta información se pueden realizar análisis financieros para determinar que tan viables pueden ser los proyectos relacionados con el ahorro energético.

1.3. Variable Objetivo

Como ya menciono anteriormente, la variable objetivo que se desea predecir en este caso es “meter_reading”, la cual nos da el gasto energético que se produce en la edificación, se analizarán y posteriormente, se decidirá cuales variables serán las entradas para el entrenamiento de los algoritmos.

2. Exploración de variables

Para iniciar con la exploración de variables lo primero que se hace es juntar la información que esta distribuida en los dataset “building_metadata”, “weather_train” y “train”, de forma de que se obtiene un dataset con todas las variables disponibles, dicho dataset se deja con el nombre de “train” ya que será con el cual se trabajará. Luego de hacer este procedimiento, se analizan algunas variables que se consideran de interés.

2.1. Análisis de la variable objetivo

Para empezar con la exploración de variables, se analiza el comportamiento de la distribución que tiene la variable objetivo, dicho comportamiento de muestra en la *Figura 1,* donde se puede observar que la variable objetivo tiene una asimetría bastante pronunciada hacia la izquierda. Este problema puede ser arreglado mediante una transformación logarítmica, cuyo resultado se muestra en la *Figura 2*.

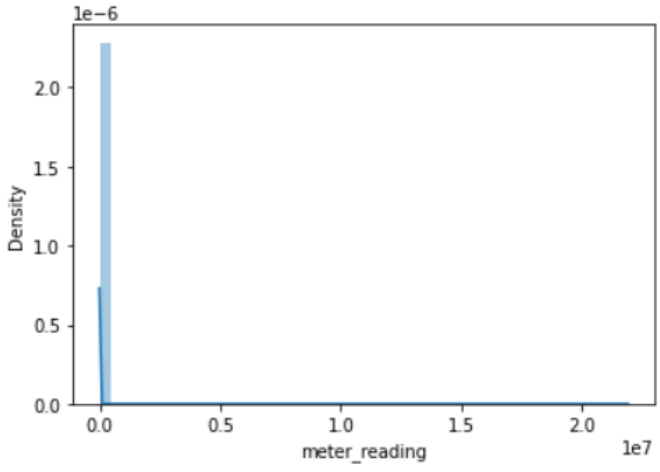

Figura 1. Distribución de la variable objetivo.

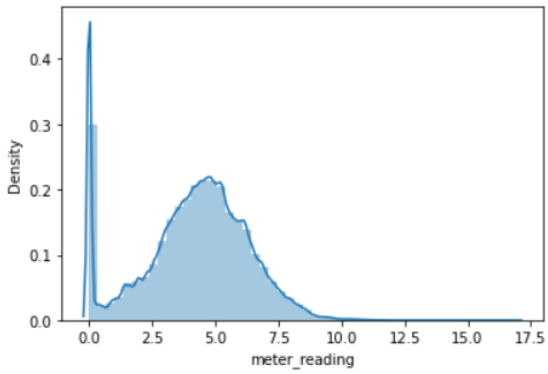

Figura 2. Distribución de la variable objetivo después de la transformación logarítmica.

En la *Figura 2*, se puede observar que la distribución de la variable objetivo luego de la transformación logarítmica ya tiene un comportamiento mas adecuado para realizar un análisis, y es esta variable transformada la que se usara en el entrenamiento y pruebas de los algoritmos. Cabe destacar que la asimetría o *skewness* de la variable objetivo antes de la transformación era de 104.81, y el valor luego de la transformación es de -0.27, lo cual refleja una gran reducción. También se observa que la variable objetivo posee muchas lecturas correspondientes a un valor de cero.

2.2. Tipos de medidores

En el dataset se brinda información acerca del tipo medidor que tiene cada edificación, esta variable es de interés ya que podría pensarse inicialmente que dependiendo del tipo de aplicación se pueden tener consumos más o menos altos de energía que se relacionen a un tipo de medidor especifico, ya sea de electricidad, vapor o calentadores de agua. En la *Figura 3* se muestra la frecuencia o cantidad de lecturas que se presentan para cada tipo de medidor, es claro que los medidores de electricidad son los más recurrentes en el dataset, mientras que los de agua caliente son los menos empleados.

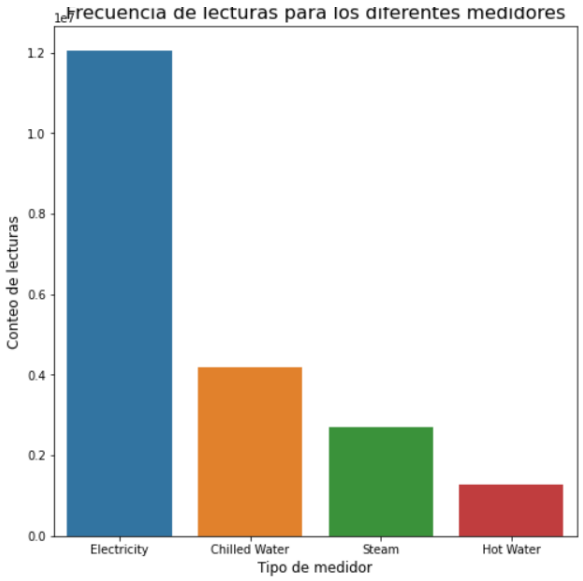

Figura 3. Frecuencia de cada tipo de medidor.

En la *Figura 4* se muestra el consumo energético dado para cada tipo de medidor. Se puede observar que los medidores de vapor son los que tienen las lecturas de mayores gastos energéticos, mientras que los medidores de agua caliente son los que presentan los menores consumos. También se observa que los gastos energéticos relacionados a la electricidad y aplicaciones de agua helada (chilled water) tienen valores semejantes, siendo un poco mas alto los de agua helada.

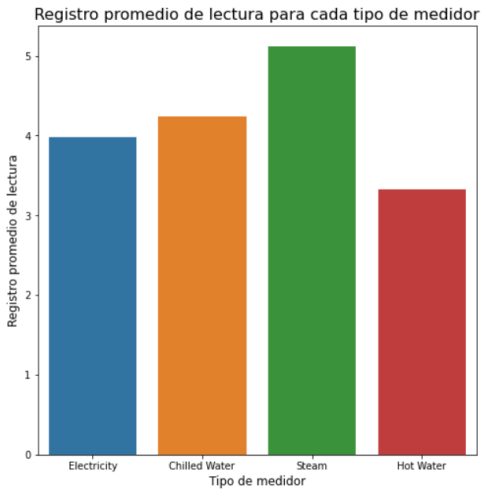

Figura 4. Consumo energético por tipo de medidor.

2.3. Uso primario de la edificación

Otra variable que se considera de interés en el análisis es el tipo de uso que se le da a la edificación. Esto se debe a que dependiendo del uso de la edificación

es más factible encontrar algún tipo de aplicación especifica, por ejemplo, en una escuela quizás no es tan factible contar con un sistema de agua caliente, pero si por ejemplo un gasto energético considerable.

En la *Figura 5* se muestra la frecuencia de cada uno de los usos primarios de las edificaciones. Se observa que las edificaciones de uso educacional son las más comunes, mientras que las religiosas son las que menos se presentan en el dataset.

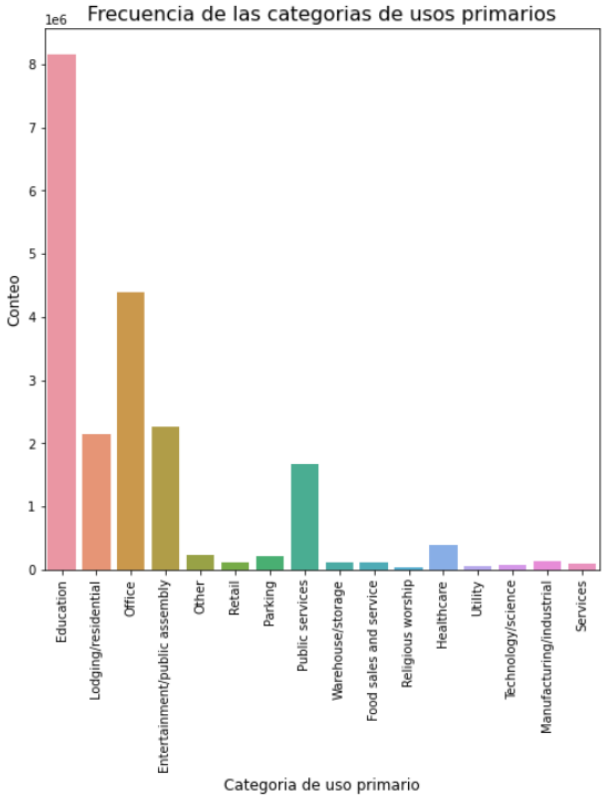

Figura 5. Frecuencia de los diferentes usos primarios de edificaciones.

En la *Figura 6* se presenta el consumo energético para diferentes sitios y usos de edificación. El sitio 13 es el que posee, en general, las lecturas promedio más altas de gasto energético para los medidores. Por su parte, el sitio 11 es el que tiene las menores lecturas promedio en sus medidores, y dicho sitio solo posee edificaciones de uso educacional. También se observa que en el sitio 13 las lecturas más altas corresponden a aplicaciones industriales seguido de centros de salud.

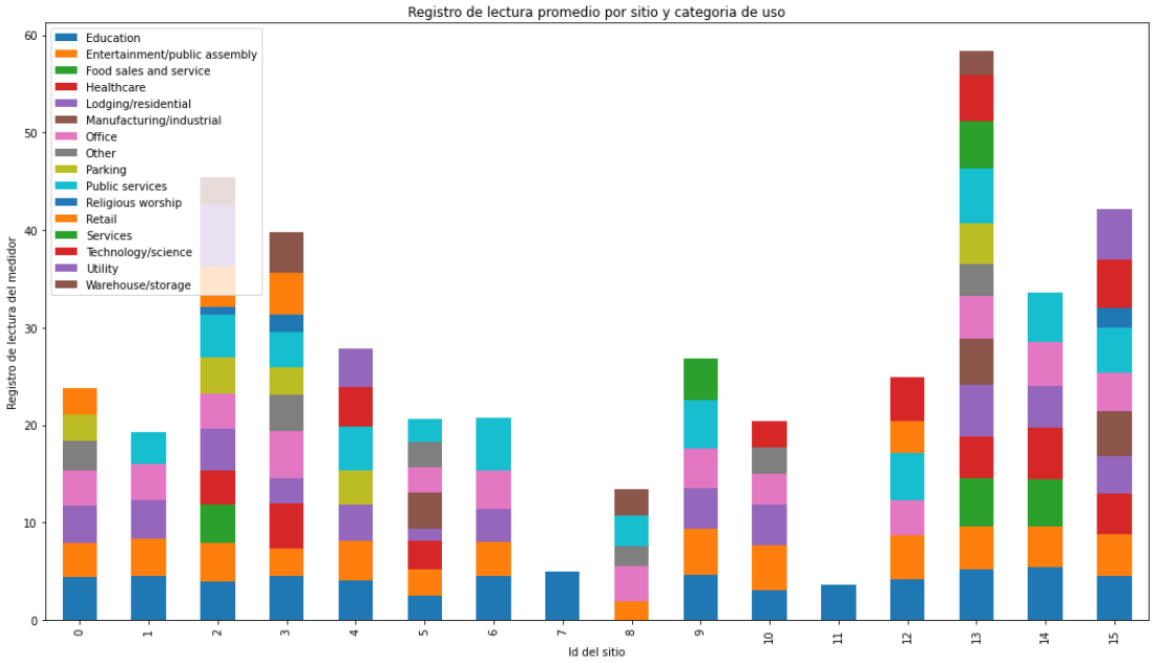

Figura 6. Gasto energético por sitio y tipo de uso de edificación.

2.4. Consumo de energía por mes y hora

Como es bien sabido, en Python si se tiene la información de tiempo, es posible convertir la variable a una serie temporal y realizar manipulaciones. En este caso el dataset se manipulo para obtener información de hora y mes y así obtener las graficas que se presentan a continuación.

La *Figura 1* muestra el comportamiento de la variable objetivo en función de la hora, es decir, el consumo energético a lo largo del día. Se puede observar que el consumo de energía cae en un rango aproximado desde la 1 AM hasta las 4 AM, presentándose el valor mínimo cerca de las 4 AM. Por otra parte, en el rango de las 12 PM a 3 PM, se observa que el consumo de energía se máxima, presentando un valor pico alrededor de la 1 PM. Este comportamiento es de

esperarse ya que en estos rangos de horas las edificaciones están en plena operación.

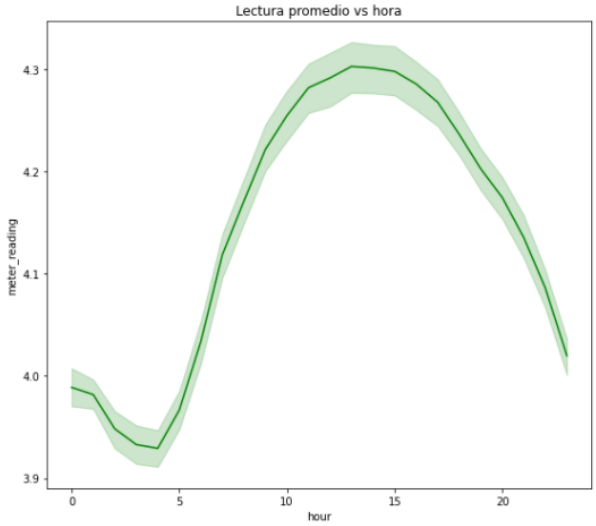

Figura 7. Consumo energético por hora.

La *Figura 8* muestra como se comporta el gasto energético en función de los meses del año. Se observa que el consumo de energía es muy bajo en el periodo de enero a marzo, presentando un valor mínimo en el mes de marzo. Además, el consumo de energía presenta un crecimiento gradual desde el mes de abril hasta que alcanza su valor pico en el mes de agosto.

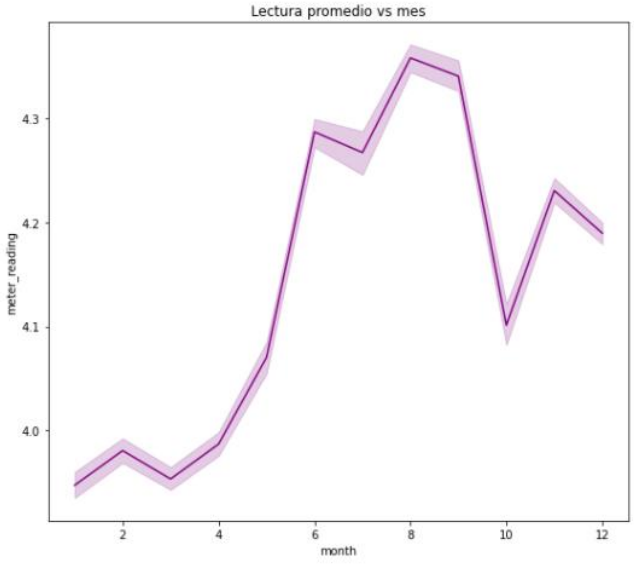

Figura 8. Consumo energético por mes.

2.5. Datos faltantes

Algo que también es importante analizar antes de entrenar un algoritmo de *machine learning* es buscar datos faltantes en el dataset. En la *Figura 9 y Tabla 1* se muestran los porcentajes de datos faltantes de las variables que poseen valores NaN. La variable que más datos faltantes tiene es *floor_count* con un 82.65%, seguida de *year_built* con aproximadamente 60%. Por lo que debe tenerse en cuenta a la hora realizar el entrenamiento del modelo. Más adelante se discutirá el tratamiento de estos datos faltantes.

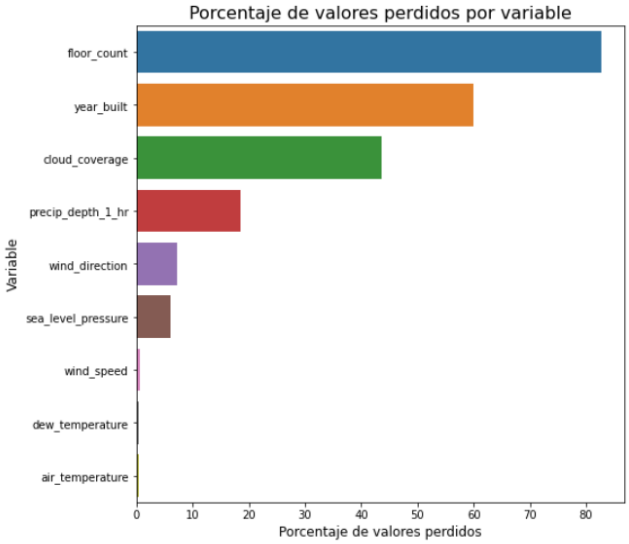

Figura 9. Datos faltantes por variable.

Tabla 1. Porcentaje de datos faltantes por variable.

	Total	Percent
floor_count	16709167	82.652772
year_built	12127645	59.990033
cloud_coverage	8825365	43.655131
precip_depth_1_hr	3749023	18.544739
wind_direction	1449048	7.167792
sea_level_pressure	1231669	6.092515
wind_speed	143676	0.710701
dew_temperature	100140	0.495348
air_temperature	96658	0.478124

2.6. Correlación de variables

La *Tabla 2* muestra los valores de correlación que existen entre las diferentes variables con la variable objetivo. Se puede observar que *square_feet* es la variable que tiene la mayor correlación. También se puede observar que, en general, las variables relacionadas al clima tienen una correlación tan baja que puede decirse que no están para nada relacionadas con la variable objetivo.

Tabla 2. Correlación de las variables con la variable objetivo.

	meter_reading
meter_reading	1.000000
square_feet	0.366016
floor_count	0.342052
site_id	0.139902
building_id	0.126395
year_built	0.103332
meter	0.064360
dayofyear	0.047776
month	0.047386
hour	0.033277
dew_temperature	0.008357
day	0.007223
precip_depth_1_hr	0.002923
air_temperature	-0.004705
sea_level_pressure	-0.008588
wind_direction	-0.014726
dayofweek	-0.025848
wind_speed	-0.032390
cloud_coverage	-0.033119

2.7. Distribución de las variables numéricas

En la *Figura 10* se muestran las distribuciones de cada variable. Se puede observar que algunas variables presentan una distribución semejante a la distribución normal. Se destaca que la variable *square_feet* tiene una asimetría marcada hacia la izquierda. Nuevamente este problema podría tratarse mediante una transformación logarítmica obteniéndose una distribución como la presentada en la *Figura 11*.

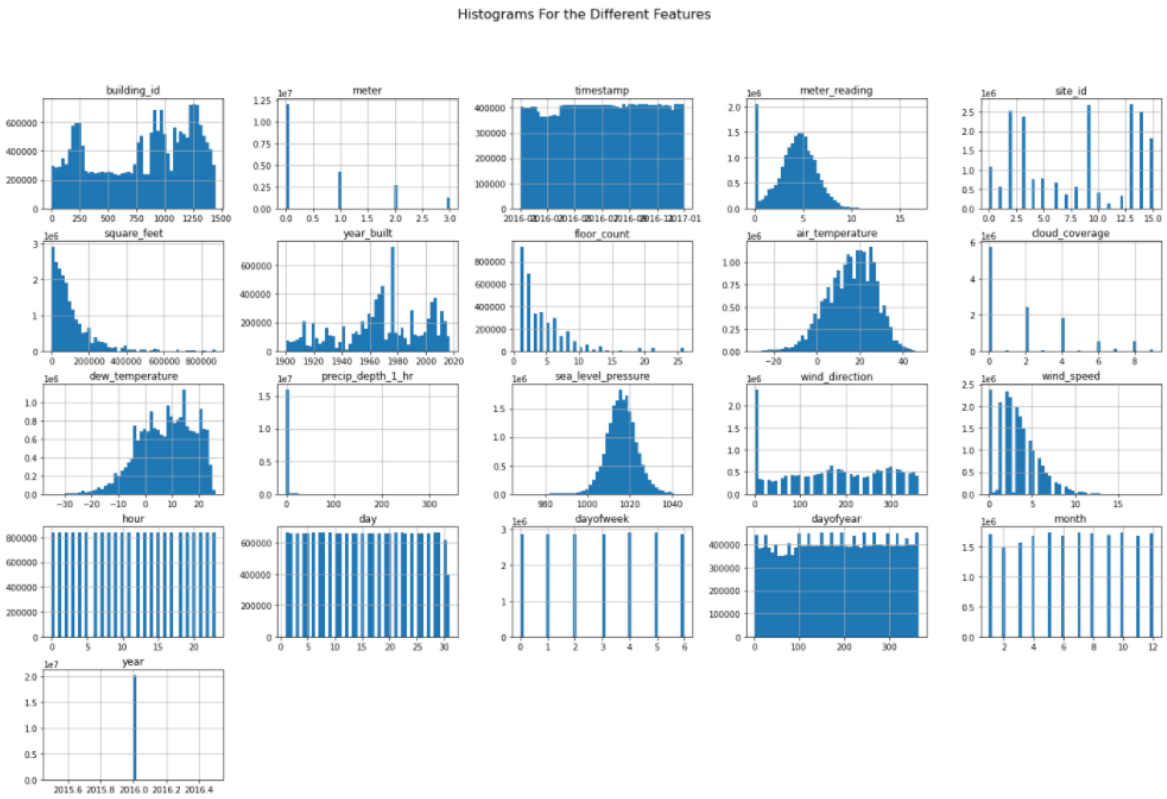

Figura 10. Distribución de las variables.

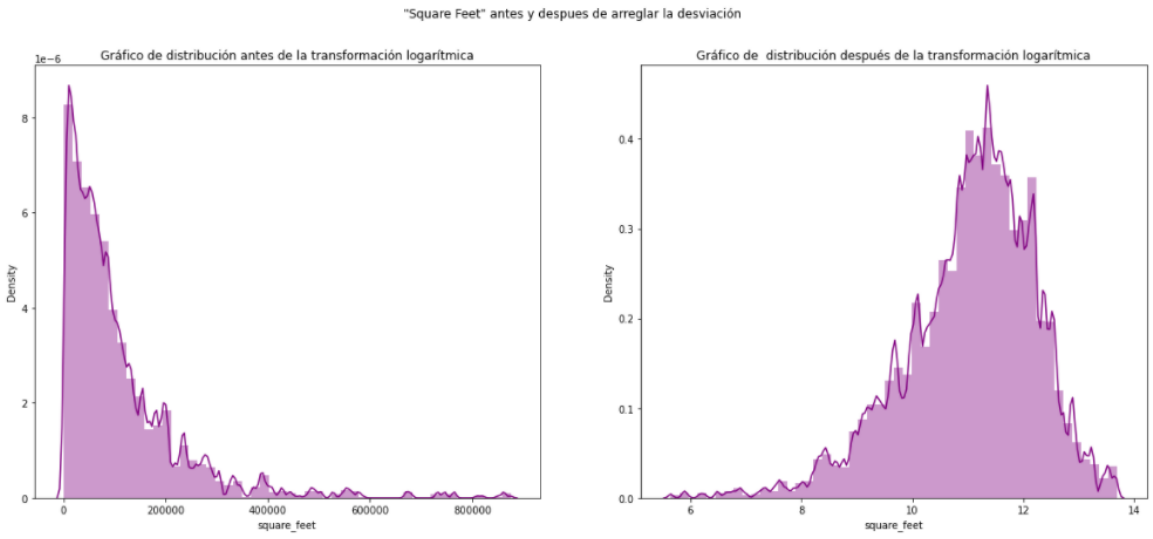

Figura 11. Distribución de probabilidad de la variable "square_feet" antes y después de la transformación logarítmica.

3. Tratamiento de datos

- **Remoción de lecturas cero en la variable objetivo:**
  Como se mencionó anteriormente, la variable objetivo presenta muchas lecturas correspondientes a un valor de cero. Esto es un problema para el análisis ya que al tener tantos datos correspondientes a este valor puede hacer que se genere un algoritmo sesgado a la hora de entrenar un modelo y no permitiría generar predicciones adecuadas. Por otro lado, también ese necesario tener en cuenta que una lectura de cero en un medidor no tiene sentido, ya que significaría que no hay gasto energético, por lo que lo más probable es que se trate de un error de lectura y dichos datos no aportan un valor.

```
meter_readings_zero = list(train[train.meter_reading == 0].index) #lista con los indices de los valores que tienen lecturas 0
train.drop(meter_readings_zero, axis = 0, inplace = True) #eliminar filas con lecturas cero en la variable objetivo
print('Nuevo tamaño de los datos: ',train.shape)
```

Figura 12. Código para eliminar las lecturas cero de la variable objetivo.

- **Eliminación de las columnas con muchos datos faltantes:**
  Como se encontró en el análisis exploratorio, existen algunas variables que tienen datos faltantes. En concreto, para este caso se considerará que una variable que tenga mas del 50% de datos faltantes debe ser eliminada ya que no aportará la suficiente información al modelo. De esta manera las variables que son eliminadas del dataset son *floor_count* y *year_built.* La *Figura 13* muestra el código implementado para eliminar dichas variables.

```
criterio = len(train) * 0.5 #criterio para eliminar la columna (50% de las filas que se tienen)
train.dropna(axis=1, thresh = criterio, inplace = True) #eliminación de las columnas con 50% o más de datos faltantes
print('New Shape of Train Data:',train.shape)
```

Figura 13. Código para eliminar las variables deseadas.

- **Relleno de datos faltantes:**
  Para el resto de las variables que contengan datos faltantes que no son eliminadas se opta por rellenar dichos valores faltantes con la media de los datos de cada una de ellas mismas. La *Figura 14* muestra el código usado para rellenar dichos valores.

```
train['cloud_coverage'].fillna(train.cloud_coverage.median(), inplace=True)
train['sea_level_pressure'].fillna(train.sea_level_pressure.median(), inplace=True)
train['precip_depth_1_hr'].fillna(train.precip_depth_1_hr.median(), inplace=True)
train['wind_direction'].fillna(train.wind_direction.median(), inplace=True)
train['wind_speed'].fillna(train.wind_speed.median(), inplace=True)
train['dew_temperature'].fillna(train.dew_temperature.median(), inplace=True)
train['air_temperature'].fillna(train.air_temperature.median(), inplace=True)
```

Figura 14. Código implementado para rellenar los datos faltantes.

- **Adición de la variable estación y variable para saber si es de día o noche:** Para este análisis se encuentra interesante el hecho de saber de que estación del año es el cada dato, ya que normalmente en sistemas que ver con calentamiento de agua, o enfriadores de agua para sistemas de aire acondicionado, o sistemas de vapor, la estación del año puede influir en la carga que se impone en los sistemas aumentando o disminuyendo su consumo. También se puede tener que ciertas aplicaciones sean mas usadas en una estación que otra, como por ejemplo el uso del agua caliente en verano, el cual podría reducirse en esta estación. Además de esto la carga también tiene variaciones en el día o noche, y los consumos pueden presentar una variación dependiendo de esto. En la *Figura 15* se muestra el código empleado para añadir estas variables al dataset.

```
# Adicionamos la estación del año, lo cual puede mejorar el análisis

train['season'] = train['timestamp'].apply(lambda x: 'Spring' if x.month==3 or x.month==4 or x.month==5 else
                                                     'Summer' if x.month==6 or x.month==7 or x.month==8 else
                                                     'Autumn' if x.month==9 or x.month==10 or x.month==11 else
                                                     'Winter')

#Adicionamos una variable para saber si es de día, 1 es de día, 0 es de noche
train['isDayTime'] = train['timestamp'].apply(lambda x: 1 if x.hour >=6 and x.hour <=18 else 0)
```

Figura 15. Código para añadir variables de interés.

- **Transformación de variables categóricas:** Las variables categóricas no pueden ser usadas en el entrenamiento de un algoritmo, por lo que se debe hacer una transformación para convertirlas a variables numéricas que puedan ser usadas en entrenamiento. En este caso las dos variables categóricas que deben ser transformadas son *primary_use* y *season*. La *Figura 16* muestra como se realiza la transformación de dichas variables.

```
var_categoricas = ['primary_use', 'season']
encoder = preprocessing.LabelEncoder()

for i in var_categoricas:

    train[i] = encoder.fit_transform(train[i])
```

Figura 16. Transformación de variables categóricas a variables numéricas.

4. Métodos supervisados

4.1. Selección de modelos

4.1.1. Ajuste de la métrica:

Como se mencionó anteriormente, la métrica para medir el desempeño de los modelos será el RMSLE (Root Mean Squared Logarithmic Error), sin embargo, como realizamos una transformación logarítmica a la variable objetivo, la métrica que se puede implementar es el RMSE (Root Mean Squared Error), y a este

valor sacarle la raíz cuadrada, y de esta forma ya tendríamos el RMSLE. En la *Figura 17* se muestra la función mediante la cual se calcula el RMSLE.

```
#Función para calcular el RMSLE de los modelos implementados
def RMSLE(y_actual, y_pred):

    return np.sqrt(mean_squared_error(y_actual, y_pred))
```

Figura 17. Función de cálculo del RMSLE.

4.1.2. Partición de los datos

Para entrenar los modelos se usará el dataset que se generó, llamado *train.* De todo el conjunto de estos datos es necesario hacer una partición para entrenar y otra para prueba. Los datos que se guardan para prueba no serán utilizados en ningún momento en el entrenamiento. En cuanto a los datos que se usarán para entrenamiento, se hará una partición adicional en este conjunto de modo que se tenga una parte para entrenar y la otra parte para hacer validación. Es de esta forma que la partición de los datos de entrenamiento se usa en el ajuste del modelo, mientras que los que se guardan para prueba solo se usan al final, ya cuando el modelo ha sido entrenado, esto para simular un conjunto de datos completamente nuevos para el algoritmo y probar así su desempeño. En la *Figura 18* se muestra el código implementado para realizar estas particiones. Cabe destacar que para entrenar los algoritmos se opto por eliminar las variables del clima ya que no tenían casi correlación con la variable objetivo, así como las variables que dan información del día de la semana, el mes, el año. De esta manera se conservan solo aquellas que tienen algo de correlación con la variable objetivo.

```
#--------Partición de los datos-----------

from sklearn.model_selection import train_test_split

test_size = 0.3
val_size  = test_size/(1-test_size) # Elementos de validación


print (X.shape, y.shape)
print ("test size %.2f"%test_size)
print ("val size is %.2f (relative to %.2f) "%(val_size, 1-test_size))

#Xtv, ytv son los datos que se usan para entrenar el modelo
#Xts, yts son los datos que se usan para probar el modelo (solo se usan para el testeo final de los modelos que se seleccionen)
Xtv, Xts, ytv, yts = train_test_split(X, y, test_size=test_size)
print (Xtv.shape, Xts.shape)
```

Figura 18. Código para realizar la partición de datos.

4.1.3. Código de selección de modelos

Para desarrollar el análisis se plantean inicialmente tres modelos: Regresión lineal, Árbol de decisión y RandomForest. De las tres posibilidades se hará una selección inicial de los dos que den mejores resultados. En la *Figura 19* se

muestra el código implementado para la selección. Cabe resaltar que se implementó una metodología de *cross-validation* para realizar la selección de los modelos. De este análisis se encontró que los modelos con el mejor desempeño fueron el *Árbol de Decisión* y el *RandomFortest.*

```
#Selección de modelos

zscores = []
estimators = [estimator1, estimator2, estimator3]
for estimator in estimators:
    print("---------")
    z = cross_validate(estimator, Xtv, ytv, return_train_score=True, return_estimator=False,
                       scoring="neg_mean_squared_error", cv=ShuffleSplit(n_splits=10, test_size=val_size))
    report_cv_score(z)
    zscores.append(np.mean(np.sqrt(z['test_score']*(-1))))
best = np.argmin(zscores)
print ("Seleccionado: ", best)
best_estimator = estimators[best]
print ("\n Mejor modelo: ")
print (best_estimator)
```

Figura 19. Selección de modelos.

4.2. Mejores hiperparámetros de los modelos

Una vez obtenidos los modelos a trabajar, se hace un estudio para obtener los mejores hiperparametros de las variables. Esto se realiza mediante un módulo de la librería de *sk.learn.model_selection*, llamado *GridSearchCV.* Este modulo permite hacer un análisis variando los hiperparámetros del algoritmo de interés implementando una metodología de *cross-validation*. Como resultado se obtienen los mejores hiperparámetros entre la selección que se la al módulo arrojando así el mejor estimador. Para el *Árbol de Decisión*, se dan unos valores para el hiperparámetro "*mas_depth*". Los valores definidos son 2,5,8,12, y 15.

Por su parte, los hiperparametros usados del *RandomForest* son *"n_estimators"* y *"max_depth"*. En la *Figura 20* se muestran los valores que se definieron para estos dos parámetros.

```
parametros = { 'n_estimators': [5,10,15],
               'max_depth':[5,7,9]}
```

Figura 20. Valores de los hiperparámetros para RandomForest.

En la *Figura 21,* se muestra el código que se usa para buscar los mejores hiperparametros.

```
parametros = { 'n_estimators': [5,10,15],
               'max_depth':[5,7,9]}

forest_reg = GridSearchCV(estimator = estimator3,
                          param_grid = parametros,
                          cv = ShuffleSplit(n_splits= 5, test_size=val_size),
                          scoring = 'neg_mean_squared_error',
                          verbose = 1,
                          return_train_score = True,
                          n_jobs = -1)
forest_reg.fit(Xtv, ytv)
```

Figura 21. Código para obtener los mejores hiperparámetros.

Finalmente, del análisis se obtuvo que el mejor hiperparámetro para el Árbol de Decisión es max_depth = 15. Mientras que para el RandomForest los valores para sus hiperparametros, n_estimators y max_depth, son 5 y 9 respectivamente.

5. Métodos no supervisados y supervisados

Una vez realizado el análisis con solamente métodos supervisados, se realizó un análisis implementando métodos no supervisados a modo de tratamiento de los datos antes del entrenamiento de los métodos supervisados. En este caso se usaron dos métodos no supervisados: PCA y NMF. De esta manera se podrá reducir la dimensionalidad del dataset y conservar aquellos datos que conserven la mayor cantidad de información y con estos entrenar los algoritmos supervisados.

5.1. PCA y RandomForest

Para implementar el PCA, se realizó un código mediante el cual se exploraron varias opciones de hiperparámetro de este algoritmo. Tomando valores para su hiperparámetros *n_components* de 1,3,5,7 y 9. Se entrenaron varios PCA con estos valores y luego se uso cada uno como el dataset de entrada para el algoritmo RandomForest, y así buscar cual de estos PCA daba el mejor desempeño. Se encontró que el mejor desempeño lo daba aquel modelo con $n_{components} = 5$. La *Figura 22* muestra la línea de código implementada para hacer este análisis.

```
from sklearn.decomposition import PCA
components = [1,3,5,7,9]
test_size = 0.3
val_size  = test_size/(1-test_size)
perf = [] #desempeños de los modelos
Rdm_forest = RandomForestRegressor(n_estimators = 5,max_depth = 9)
for i in components:
    pca = PCA(n_components = i)
    X_t = pca.fit_transform(X)

    #Partición de datos
    #Xtv, ytv son los datos que se usan para entrenar el modelo
    #Xts, yts son los datos que se usan para probar el modelo (solo se usan para el testeo final de los modelos que se selecci
    Xtv, Xts, ytv, yts = train_test_split(X_t, y, test_size=test_size)
    print (Xtv.shape, Xts.shape)

    Rdm_forest.fit(Xtv, ytv)
    perf.append(RMSLE(yts , Rdm_forest.predict(Xts)))
    print('RMSLE del modelo con ', i ,'elementos: ',"{:.5f}".format(RMSLE(yts , Rdm_forest.predict(Xts))))
    print('------------')

print('Mejor RMSLE: ',"{:.5f}".format(np.min(perf)),'  ; obtenido con ',components[np.argmin(perf)],' componentes para PCA')
```

Figura 22. Código de exploración del mejor PCA.

Una vez obtenido el mejor PCA, se realiza nuevamente el procedimiento descrito anteriormente para obtener los mejores hiperparámetros del RandomForest, usando *GridSearchCV.* De esta manera se obtiene que para este caso los mejores hiperparámetros son n_estimators = 15 y max_depth = 9.

5.2. NMF y Árbol de Decisión

En cuanto al NMF, el análisis es igual que con PCA. Se busca el mejor parámetro entre varias opciones dadas. En este caso se obtuvo que el mejor valor para el hiperparámetro de este algoritmo es n_components = 3. Usando el dataset transformado con el mejor NMF, se busca el mejor hiperparámetro para el Árbol de Decisión. Se obtuvo que el mejor hiperparámetro en este caso es max_depth = 15.

6. Curvas de aprendizaje

6.1. Métodos supervisados

Para obtener la curva de aprendizaje de los algoritmos se usa el modulo *learning_curves* de la librería de sk.learn.model_selection, la cual implementa una metodología de cross-validation para evaluar diferentes desempeños usando diferentes tamaños para el entrenamiento del modelo. La *Figura 1* muestra a gráfica de aprendizaje para el árbol de decisión obtenido anteriormente. Cabe destacar que la línea verde llamada *Cross-validation score*, es la curva de aprendizaje en prueba. Se puede observar que si bien, la curva de aprendizaje inicialmente presenta una mejora en el desempeño tanto en entrenamiento como en prueba, luego se presenta un comportamiento de *Bias*, puesto que a pesar de que se le dan más datos parece ser que el error se estabiliza y no disminuye a pesar de incrementar la cantidad de datos. Como

este algoritmo fue entrenado eliminando algunas de las columnas del dataset, es posible tratar de mejorarlo usando más columnas, si esto no da un cambio significativo, lo recomendable es incrementar la complejidad del modelo.

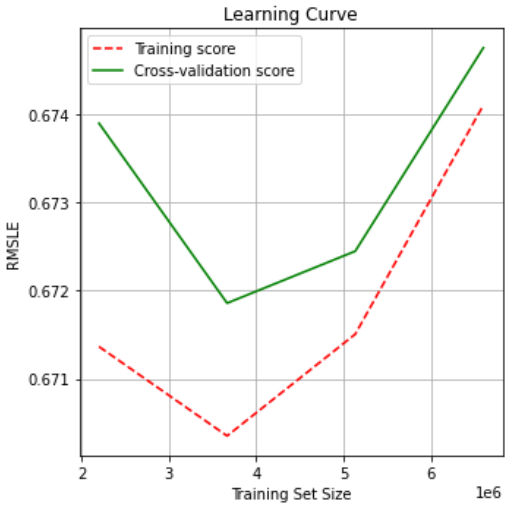

Figura 23. Curva de aprendizaje para el Árbol de Decisión.

En la *Figura 24* se muestra la curva de aprendizaje para el algoritmo *RandomForest* implementado. Se observa que el error incrementa al darle más datos al algoritmo, hasta que llega a un punto (aproximadamente 50% del tamaño del dataset), donde el error comienza a disminuir, y parece estabilizarse. Esto da una pista acerca de la cantidad de datos necesarios para entrenar el algoritmo de manera que su error sea bajo. En este caso no se evidencia tanto un Bias u overfitting.

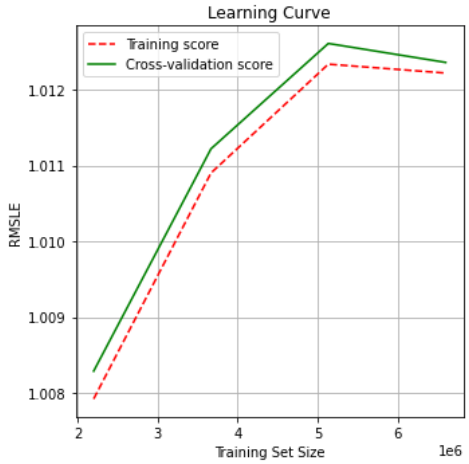

Figura 24. Curva de aprendizaje para el RandomForest.

6.2. Métodos no supervisados

Para obtener la curva de aprendizaje de la combinación de los métodos no supervisados y supervisados, se implementa la misma metodología descrita anteriormente. La *Figura 25* muestra la curva de aprendizaje para la combinación de *PCA* y *RandomForest*. Se observa que, a pesar de usar más datos para entrenar el modelo, el error en entrenamiento no disminuye y continúa incrementando. El modelo en prueba presenta un comportamiento igual. Este comportamiento puede deberse a que el modelo está presentando *Bias* (sesgo). Este error puede tratarse al incrementar la complejidad del modelo ó implementando más columnas en el entrenamiento del modelo, por lo que podría explorarse la posibilidad de implementar un PCA con una menor reducción de dimensionalidad.

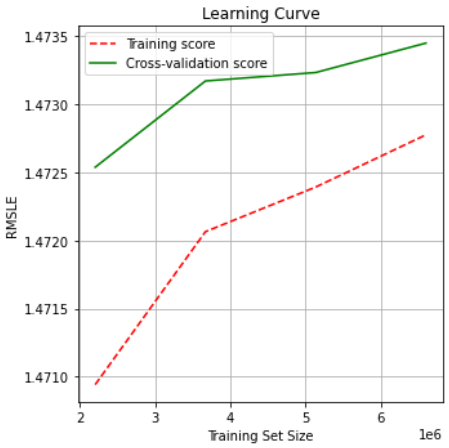

Figura 25. Curva de aprendizaje PCA+RandomForest.

La *Figura 26* muestra las curvas de aprendizaje en entrenamiento (roja) y prueba (verde) para la combinación de NMF y Árbol de Decisión. La gráfica muestra que, si bien el error es bajo en entrenamiento, la brecha que existe entre este valor y el error en prueba es muy grande, lo cual es un caso claro de *overfitting*, por lo que es posible que el modelo sea muy complejo o que se requieran más datos. En primera instancia para implementar una mejora lo más recomendable sería trabajar en la complejidad del modelo ya que obtener más datos implica el gasto de más recursos monetarios.

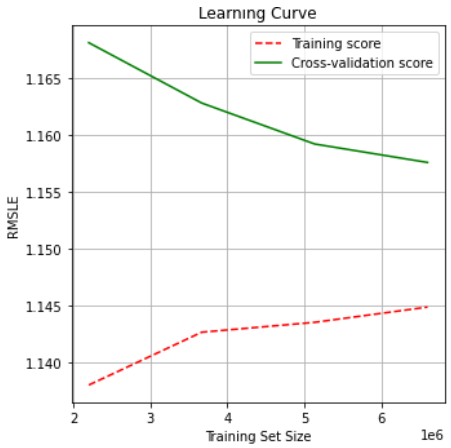

Figura 26. Curva de aprendizaje NMF+DecisionTree.

7. Retos y condiciones de despliegue del modelo

Para evaluar el desempeño mínimo con el cual se consideraría que el modelo genera un beneficio, se debe comparar el gasto energético generado por las predicciones del modelo con el posible ahorro monetario que se podría lograr al implementar estrategias que reduzcan ese gasto. Si el modelo permite obtener un nivel de ahorro objetivo mínimo, el cual puede ser establecido por los inversores. Administradores o gerentes de las edificaciones, se puede considerar que el modelo es apto para desplegarse en producción.

Para desplegar el modelo en producción, es necesario conectar todos los sensores que brinden la información necesaria, a un computador donde puedan ser usados los datos para generar las predicciones. Uno de los retos que se tiene para implementar este modelo, es que las edificaciones deben ser de alguna manera “inteligentes” ya que deben contar con la instrumentación necesaria para tomar los datos e implementar una tecnología del internet de las cosas (IOT), ya que es necesario que estos dispositivos se este comunicando para que el modelo puede ser usado. Así mismo también es necesario implementar herramientas de la nube que permitan guardar información y medir constantemente el desempeño del modelo, y en el caso en el que dicho desempeño caiga por debajo de los niveles establecidos, brindar información para reentrenar el modelo.

8. Conclusiones

- Es necesario hacer un análisis detallado de que variables tienen más peso en los modelos entrenados, ya que de esta manera se puede reducir el error.
- Es necesario aumentar la complejidad de algunos modelos ya que se presentan problemas de overfitting.
- Lo métodos no supervisados pueden ser implementados como una forma de preprocesado de datos, que nos permita generar un dataset más compacto guardando las variables más significativas.
- Se puede implementar un modelo de *Clustering* para analizar los patrones y comportamientos de cada tipo de edificación.

Bibliografía

- ASHRAE - Great Energy Predictor III | Kaggle. (2021). Retrieved 16 December 2021, from https://www.kaggle.com/c/ashrae-energy-prediction/overview/description