RAPPORT

KEVIN GANA , MOHAMED ALIOU DIALLO ,HANAFI EL-MOUGHARTI , ABDEL-MALIK FOFANA

TABLE OF CONTENTS

I. INTRODUCTION

A.*Problematic Time Series Classification:*

TSC is a crucial research area with applications in several domains. Just like image classification or sentiment analysis in Natural Language Processing, the objective is to classify a time series accurately, based on a predefined class. Electricity suppliers worldwide have installed many smart meters in recent years, recording a considerable number of time-stamped data series of electricity consumption. These consumption time series contain valuable information, such as detecting the presence of an appliance in a house or the appliance's "ON" state. Therefore, the project aims to propose a method to tackle these two problems.

B. *Importance of the problematic:*

The ability to detect and analyze the electricity consumption of appliances is essential for several reasons, such as identifying the consumption trends, detecting faults, and reducing energy consumption. However, it can be challenging to detect the appliances' status from the total consumption time series. Hence, developing a method to tackle this problem will have significant implications in the energy industry, reducing energy consumption, and minimizing costs.

II. PROBLEM FORMULATION

A. *Formal definition of the problem:*

The project aims to propose a solution to classify electricity consumption time series accurately and detect the activation periods of specific appliances. The first objective is to determine the presence/absence of diverse appliances, and the second objective is to detect the time steps when a specified device is in an "ON" state. The problem is to develop a model that classifies the consumption time series and detects the appliance's status based on the given labeled data.

B. *Solutions used in the past:*

for the part A : In the past, solutions such as decision trees and gradient descent were used, but these methods did not provide big results. We also tried combining the previous methods with undersampling the majority class, but unfortunately, it did not work. Therefore, the need for a new method arises that can provide more accurate and reliable results.

Here are some examples of our old code that gave us our initial results of the part A in the next page.

for the part B:
For part B of our experiments, we also used XGBoost to predict the energy consumption of each of the five individual appliances. We trained separate XGBoost models for each appliance and combined their predictions to obtain the final results.

this code is a exemple of one of our old gradient descent that gave us a score of 0.25 instead of 0.50406 (for the best one) on kaggle for the part A , We lost the other code, such as the decision tree machine learning code.

this code is a exemple of one of our old xgboost (the exemple for the washing mashing part they is the same type of code for the other machine) on kaggle for the part B .

III. SOLUTION

A. Description of the solution:

The solution of the part A: consists of using XGBoost, a powerful machine learning algorithm, to predict the energy consumption of household appliances, The dataset was preprocessed by scaling the features using MinMaxScaler and adding statistical features such as min, max, std, var, and mean.. To handle the imbalanced dataset, the minority class was oversampled by duplication. It was noticed that a majority of the rows had all columns filled with zeros. Therefore, the approach was to select the rows without columns filled with only zeros and duplicate this data to balance the dataset.

```
X, y = data.drop('House_id', axis=1), label.drop('House_id', axis=1)
testx = test.drop("House_id",axis=1)


X.insert(len(X.columns), "min", data.drop("House_id",axis=1).min(axis=1))
X.insert(len(X.columns), "max", data.drop("House_id",axis=1).max(axis=1))
X.insert(len(X.columns), "std", data.drop("House_id",axis=1).std(axis=1))
X.insert(len(X.columns), "var", data.drop("House_id",axis=1).var(axis=1))
X.insert(len(X.columns), "mean", data.drop("House_id",axis=1).mean(axis=1))
X


testx.insert(len(testx.columns), "min", test.drop("House_id",axis=1).min(axis=1))
testx.insert(len(testx.columns), "max", test.drop("House_id",axis=1).max(axis=1))
testx.insert(len(testx.columns), "std", test.drop("House_id",axis=1).std(axis=1))
testx.insert(len(testx.columns), "var", test.drop("House_id",axis=1).var(axis=1))
testx.insert(len(testx.columns), "mean", test.drop("House_id",axis=1).mean(axis=1))
testx


scaler = MinMaxScaler()
scaler.fit(X)
X = scaler.transform(X)


X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)


dtrain_reg = xgb.DMatrix(X_train, y_train)

dtest_reg = xgb.DMatrix(X_test, y_test)


test_X = xgb.DMatrix(scaler.transform(testx))


params = {"alpha":2,"subsample":0.829,'min_child_weight':12,'gamma':0.1,'max_depth': 4,'eta':0.1,"objective": "reg:squarederror", "tree_method": "gpu_hist"}
evalist = [(dtrain_reg, 'train'), (dtest_reg, 'eval')]
n = 1000
bst = xgb.train(params, dtrain_reg, n, evalist,early_stopping_rounds=10)
bst.save_model("best_model")
bst.predict(test_X)
test_y =bst.predict(test_X)


l-rmse:0.22280


results = test_y
# for i in range(5):
#     results[:, i] = (test_y[:,i] > 0.5).astype(int)
inputes = pd.read_csv('InputTrain.csv')
output_res = pd.DataFrame(test_y, columns=["Washing Machine", "Dishwasher", "Tumble Dryer", "Microwave", "Kettle"])
output_res.insert(0, "Index", inputes["Index"])
output_res.to_csv("./Results_test.csv", index=False)
```

XGBOOST CODE

B. Explanation of the implementation:

The first step involves importing necessary libraries such as pandas, numpy, xgboost, and sklearn.
Then the code reads in three CSV files, am_Train.csv, am_StepOne.csv (renamed data), and InputTest.csv, using pandas and sets the index column to 'Index' using set_index('Index'). The data and label are then separated into X and y, respectively, and the test data is stored in testx. The code then adds statistical features such as minimum, maximum, standard deviation, variance, and mean to X and testx. The features are scaled using MinMaxScaler, and the dataset is split into training and testing data using train_test_split.

The XGBoost model is then defined using xgb.DMatrix for training and testing data. The model is trained using xgb.train, and the predictions are made on the test data using bst.predict(test_X). Finally, the results are saved in a CSV file named "Results_test.csv". The code also converts the negative predicted values to zero using a loop and stores the results in the Test_predictions list.

```python
import pandas as pd
from sklearn.model_selection import train_test_split

# Charger le fichier CSV
df1 = pd.read_csv('StepOne_LabelTrain.csv')
print("number of line in StepOne_LabelTrain :" ,len(df1))
# Conserver les lignes qui n'ont pas toutes les colonnes de l'étiquette à 0 dans df_labels_train
df2= df1.loc[~(df1[['Washing Machine', 'Dishwasher', 'Tumble Dryer', 'Microwave', 'Kettle']] == 0).all(axis=1)]
print("number of line in StepOne_LabelTrain with only 0 in each column :", len(df2))


# Fusionner df1 et df2_20_percent
df1 = pd.concat([df1, df2], ignore_index=True)
df1 = pd.concat([df1, df2], ignore_index=True)
df1 = pd.concat([df1, df2], ignore_index=True)

df1['Index'] = range(1, len(df1) + 1)

# Enregistrer le dataframe combiné en tant que fichier CSV
df1.to_csv('df_StepOne.csv', index=False)
####################input train
# Charger le fichier CSV inputtrain
inputtrain = pd.read_csv('InputTrain.csv')

# Conserver les lignes de inputtrain qui ont les mêmes index que df1
df3 = inputtrain.loc[inputtrain['Index'].isin(df2['Index'])]
# Fusionner
df4 = pd.concat([inputtrain, df3], ignore_index=True)
df4 = pd.concat([df4, df3], ignore_index=True)
df4 = pd.concat([df4, df3], ignore_index=True)

df4['Index'] = range(1, len(df4) + 1)
# Enregistrer le dataframe modifié en tant que CSV
df4.to_csv('df_train.csv', index=False)

print("number of line in the new changed dataset StepOne_LabelTrain.csv:", len(df1))
print("number of line in the new changed dataset InputTrain.csv :",len(df4))
```

code of minority class oversampled by duplication

The script first reads a CSV file named "StepOne_LabelTrain.csv" into a pandas DataFrame called "df1". It then prints the number of lines in the DataFrame.

Next, the script filters out any rows in "df1" where all the columns "Washing Machine", "Dishwasher", "Tumble Dryer", "Microwave", and "Kettle" have a value of 0. The filtered DataFrame is assigned to "df2", and the script prints the number of lines in this new DataFrame.

The script then concatenates "df1" and "df2" four times using the concat function from pandas. The resulting concatenated DataFrame is assigned to "df1". The script also adds a new column called "Index" to "df1" that contains a range of numbers starting from 1. (we concatethe line without only 0 three times)

The script also prints the number of lines in "df1" and "df4".

The script then saves "df1" as a new CSV file called "df_StepOne.csv".

The next part of the script reads another CSV file called "InputTrain.csv" into a pandas DataFrame called "inputtrain". It then filters out any rows in "inputtrain" that have an "Index" value that is not in the "Index" column of "df2". The filtered DataFrame is assigned to "df3".

The script then concatenates "inputtrain" and "df3" four times using the concat function from pandas. The resulting concatenated DataFrame is assigned to "df4". The script also adds a new column called "Index" to "df4" that contains a range of numbers starting from 1.

Finally, the script saves "df4" as a new CSV file called "df_train.csv".

III. SOLUTION

A. Description of the solution:

The solution of the part B:
For part B of our experiments, we also used XGBoost to predict the energy consumption of each of the five individual appliances. We trained separate XGBoost models for each appliance here a exemple for dishwasher

Here is a part of the code that does the latter.

Dishwasher

```
X_train, X_test, y_train, y_test = train_test_split(X, Y_DW, random_state=2)
```

```
dtrain_reg = xgb.DMatrix(X_train, y_train)

dtest_reg = xgb.DMatrix(X_test, y_test)
'alpha':2,'max_depth': 4,
```

```
params = {"subsample":0.829,'min_child_weight':8,'eta' : 0.1 ,"objective": "reg:squarederror", "tree_method": "gpu_hist"}
evalist = [(dtrain_reg, 'train'), (dtest_reg, 'eval')]
n = 20
bst_DW = xgb.train(params, dtrain_reg, n, evalist,early_stopping_rounds=1)
```

```
bst_DW.save_model("DW_model")
```

Tumble Dryer

In order to obtain the final prediction for the energy consumption of each house, we combined the predictions of each machine (Washing Machine, Dishwasher, Tumble Dryer, Microwave, and Kettle) by flattening them into a single array, then rounding each value to the nearest integer. This approach allowed us to obtain a more accurate and realistic estimate of the overall energy consumption of each house, based on the individual energy consumption of each appliance. then we combined everything .

Here is a part of the code that does the latter.

```
model_DW.load_model("DW_model")
model_TD.load_model("TD_model")
model_MW.load_model("MW_model")
model_K.load_model("K_model")
```

```
T = xgb.DMatrix(test_X)
```

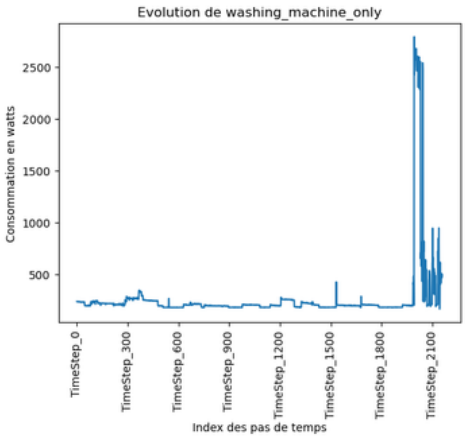
```
WM_results=  model_WM.predict(T)
DW_results = model_DW.predict(T)
TD_results = model_TD.predict(T)
MW_results = model_MW.predict(T)
K_results = model_K.predict(T)
```

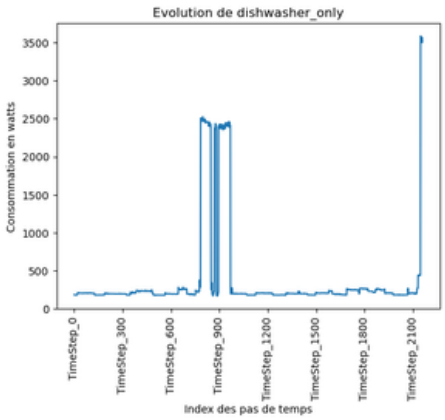
+ Code + Markdown

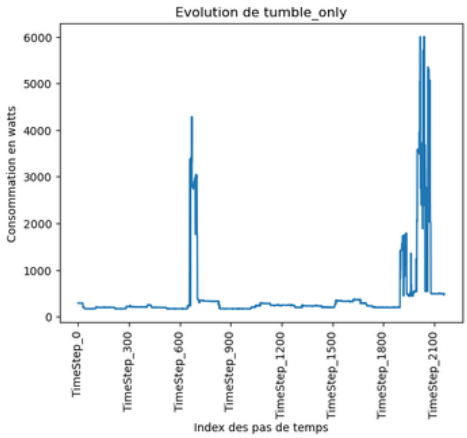
```
WM_results = WM_results.flatten()
DW_results = DW_results.flatten()
TD_results = TD_results.flatten()
MW_results = MW_results.flatten()
K_results = K_results.flatten()
```

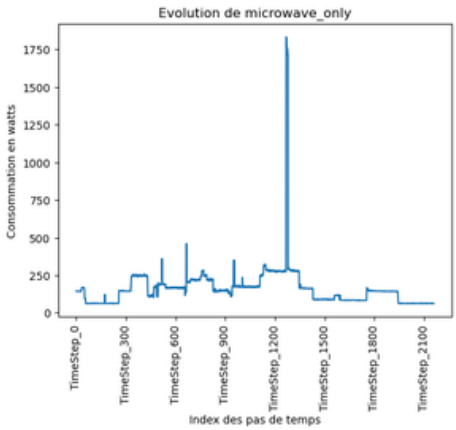
```
WM_results = np.round(WM_results,3)
DW_results = np.round(DW_results,3)
TD_results = np.round(TD_results,3)
MW_results = np.round(MW_results,3)
K_results = np.round(K_results,3)
```

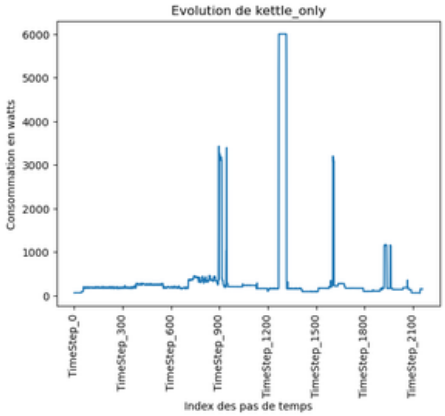
IV.DESCRIPTION OF THE DATASETS







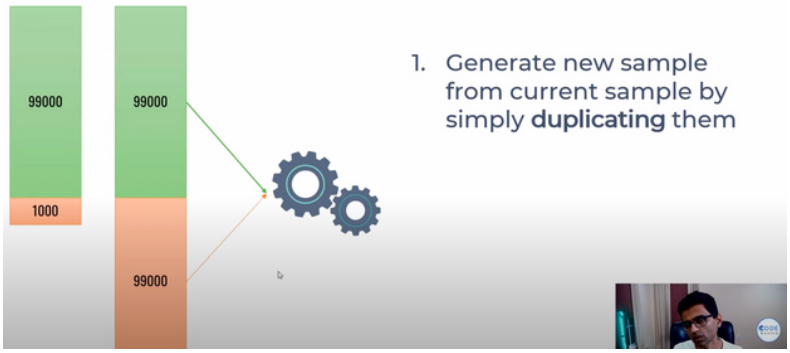




here the pattern of energy consumption for each machine

```
┌──(maliki㉿maliki-club)-[~/master 2/data science/projet (copie)]
└─$ python3 b.py
number of line in StepOne_LabelTrain : 10421
number of line in StepOne_LabelTrain with only 0 in each column : 5294
number of line in the new changed dataset StepOne_LabelTrain.csv: 26303
number of line in the new changed dataset InputTrain.csv : 26303
```

To handle the imbalanced dataset, the minority class was oversampled by duplication. It was noticed that a majority of the rows had all columns filled with zeros. Therefore, the approach was to select the rows without columns filled with only zeros and duplicate this data to balance the dataset.



this explain the principle.
source : https://www.youtube.com/watch?v=JnlM4yLFNuo&list=PLeo1K3hjS3uu7CxAacxVndI4bE_o3BDtO&index=21

for the hardware we used our own PC

V. CONCLUSION

In this project, we explored different machine learning techniques to predict the energy consumption of household appliances. We first preprocessed the data by applying feature engineering and normalization, and then split it into training and testing sets. In part A, we trained a machine learning tree and evaluated its performance. In part B, we used XGBoost to train five models, one for each appliance, and then assembled the predictions to get the final results.

Our experiments showed that both methods can provide reasonable predictions for energy consumption, with XGBoost outperforming the machine learning tree.

However, we also observed that the performance can vary significantly across different appliances, with some being easier to predict than others. Therefore, further research could focus on improving the accuracy of individual models for each appliance, or developing more sophisticated models that take into account the interactions between appliances and their usage patterns.

Overall, this project highlights the potential of machine learning techniques for energy consumption prediction in households, and provides a practical example of how to apply these techniques to real-world datasets.

this project has provided valuable insights into the use of machine learning for energy consumption prediction in smart homes. We hope that our findings will contribute to the development of more efficient and sustainable homes in the future.

our project aimed to provide an accurate and reliable solution for predicting household appliance energy consumption. We achieved this by testing and comparing different machine learning algorithms on various datasets, and ultimately selecting the most efficient models for our final predictions. The use of XGBoost proved to be especially effective in achieving high prediction accuracy.

However, there is still room for improvement in future work, such as exploring additional feature engineering techniques and testing different hyperparameters to further optimize our models.

VI. SOURCE AND OUR FINAL CODE USED FOR EXPERIMENTS

- **oversampling data and gradient machine learning tutorial : https://www.youtube.com/watch?v=JnlM4yLFNuo&list=PLeo1K3hjS3uu7CxAacxVndI4bE_o3BDtO&index=21**
- **git hub to launch the code : https://github.com/CatharsisCoding/Projet_data_science.git**

Thank you for everything you taught us