RESIDENTIAL ENERGY APPLIANCE CLASSIFICATION

Duy Tho Le
Minh Thai Nguyen
Duy Ngoc Anh Nguyen

MAY 28, 2021

Abstract

This work proposed two deep learning models that are capable of detecting the working condition (On - Off) of five appliances, including air conditioner, electric vehicle charger, oven, cloth washer and dryer. We also discuss imbalanced datasets and their effects on the performances of models, as well as methods to handle these data. The models were then submitted to Kaggle for evaluation. The whole project could be found at https://github.com/minhthai1995/FIT5149

Table of Contents

1 Introduction

Electricity is one of our most irreplaceable resources. It is used in our routine life tasks, complicated applications such as space centers or satellites. Due to its importance, people in recent years are still working with the study of the nature of electricity in order to create a way to generate electricity unprecedented in the past.

Throughout history, humans had hydroelectricity, thermal power, wind power, solar power, and even nuclear power. The nation that we are living in – Australia, is a thermoelectric country. And because each method of generating electricity has advantages and disadvantages, thermal power leads to air pollution, environmental degradation, and also ozone depletion. However, there are no alternatives for the Australian government.

Therefore, studying how to reduce electricity consumption works without delay. This research aims to analyse the usage of power for household appliances and predict the consumption for future application (when an appliance uses power in a specific minute) – to transmit electricity more rationally. For "testing the water", our scope is only for 5 appliances including Air conditioner, Electric vehicle charger, Oven, Cloth washer, and dryer in over 500,000 minutes (has been divided into training and testing data).

Because of the lack of time and human resources, the work for this project would be allocated to 3 people. The works including:

- Study the data, exploratory data analysis: Ngoc Anh Duy Nguyen
- Implement the model 1 + 2 (LSTM network): Minh Thai Nguyen
- Implement the model 3 (CNN network): Duy Tho Le

Then, we would summarise and write a report for each section. The whole process would be conducted in parallel with the goal to maximize the evaluation. Therefore, some parts of the research seemed not to be reasonable, but it brought a higher result. We also discussed regularly and made some contributions to other's parts to support and maximize the overall performance of this project.

2 Pre-processing and features used

2.1 Exploratory Data Analysis (EDA)

As mentioned in the EDA step in Jupyter Notebook, in this part we would summarise our exploration.

Our given data are training and testing with 417720 and 105540 data records in respectively, each of data record represented for one minute and they are continuous (time-series data). Our core features are **"load", "hourofday", "dayofweek"**, and 5 columns contain predicted values for 5 appliances whether at that moment they are used or not. Some default statistical values provided to us including **"dif", "absdif", "max", "var", "entropy", "nonlinear", "hurst"**:

Name	Description
Load	electrical load
Hourofday	what time the data record was captured in a day
Dayofweek	which day the data record was captured in a week

Dif	difference between two sequential load data points
Absdif	absolute value of diff
Max	maximum load in that moment
Var	variance of load over a neighbourhood time window of 30/64 minutes around each load data point
Entropy	the Shannon entropy that measures the "forecast ability" of a time series data
Nonlinear	the nonlinearity coefficient is used in Terasvirta 's nonlinearity test
hurst	the hurst is used as a measure of the long-term memory of a time series

Table 1: Input features definition

The class distribution for each of the appliance is presented below:

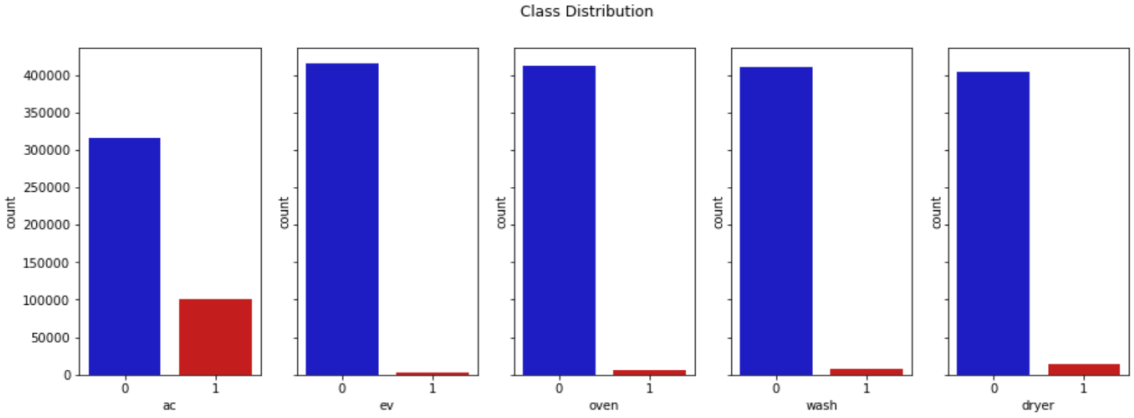

Figure 1: Class Distribution of Five Target Variables

As can be seen from the above figure, the target classes are heavily skewed, then the metric that is more suitable for this problem is the F1-score.

Due to the EDA process, we decided to keep the set of attributes as original because the new generated features using tsfresh library in Python are not useful since **some of them had low relation with the predicted values and some could be replaced by the default variables (because their correlation between them is high)** and we need to have **enough attributes to pass through the deep learning model**. Our working experience showed that although we have a potential set of features if its size is too small for the algorithm, the result should be noted as acceptable. Therefore, we would use 10 default features for both of our models.

With the above intuition, we introduce a **Convolutional neural network (CNN)** and **Recurrent neural network (RNN)**. The details of these algorithms would be discussed in the implementation step and the comparison between them would be explained in detail in this report. The key idea of pre-processing is "deep learning" by scrolling the data by the size of the window we prefer. There are two ways of scrolling: depth (for CNN) and breadth (for RNN). The window of rolling is based on the average running minutes of all 5 appliances for each running time (30 or 64 minutes).

2.2 Scaling

We used **preprocessing.StandardScaler()** from **sklearn** library to scale 10 variables to distributions with mean 0 and standard deviation of 1, the variables are **'absdif', 'dayofweek', 'dif', 'entropy', 'hourofday', 'hurst', 'load', 'max', 'nonlinear', 'var'**. Since 'id' is not used during training, thus no scaling needed.

2.3 Rolling

Data pre-processing is crucial to create appropriate input for the CNN model. A particular input of the model should look like a pseudo image. Therefore, we chose to use the function ***roll_time_series*** from **tsfresh** library to expand the number of observations. For example, we have data with n observations, with **time_shift** set to t, at every observation k^{th}, we add *t-1* observations *k-1,k-2,...k-t* above the k^{th} observation.
Here is an example of the rolling technique with $n = 5, t = 3$.

Figure 2: Example of rolling with time_shift = 3

The training data has 417720, when rolling with time_shift = 64, the data will have 417720*64 = 26734080 rows. Then the 26734080 rows will be splitted into 417720 pseudo-images with height = 64 and width = 10 (variables).

2.5 Fast Fourier transform (FFT)

In order to capture more features from the original input as well as speed up the training process of CNN (inspired by [9]), we decided to build another branch to apply FFT on the **x_train** data. Therefore, the CNN model now has two branches.
A NumPy function **numpy.fft.rfft** was used to compute the 1D discrete Fast Fourier transform domain of the input data. By merging every 2 consecutive observations in *n* observation, we can obtain $n/2 + 1$ FFT features.

2.5 Target variable merging

The idea here is that instead of building 5 different binary classifiers, we prefer a different approach which is to build only 1 multi-label classifier for each model type. To do that, 5 binary labels will need to be encoded into 1 column. For example: "**ca ev oven**" will be "**11100**"; "**ca oven dryer**" will be "**10101**"; "**ac ev oven wash dryer**" will be "**11111**". Then we used Label Encoder (from **sklearn** library) to encode those values to a range of 0 → 19. Later in the inference stage, the predictions will be decoded back.

3 Models

3.1 Model 1: 5 distinct LSTM models

3.1.1. Model idea

While reading a story, human brains understand the current content based on the understanding that is in their memory from previous words. The traditional neural networks do not have the ability to connect previous knowledge to the current task. However, the recurrent neural network (RNN) can address this problem due to its internal memory, and it is ideally suited for machine learning issues involving sequential data. The below figure compares the difference between the structure of the RNN and the Feed Forward Neural Network:

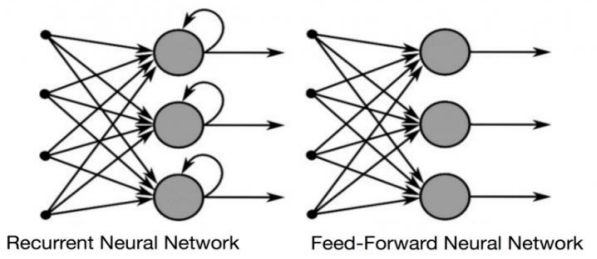

Figure 3: Structure of RNN versus FFNN

Clearly, a feed-forward neural network simply takes into account the present input, so it cannot remember anything and is not suitable for time series problems.

In RNN, the decision of the current input is made by considering what has been learned from the previous data. Therefore, our team decided to use LSTM (a special case of RNN) to build a model for the given problem.

LSTM, which stands for Long Short-Term Memory networks, has the ability to remember inputs for a long time. The LSTM can write, read, and even remove the memory information, much like a computer's memory.

3.1.2. Model development

After pre-processing data, we built five LSTM networks to predict the output of five features independently. The LSTM network is supported by Keras and the input data for the first layer must be a 3-dimensional shape (M, T, N) [3], where:

- M: number of samples
- T: sequence length (timestep)
- N: number of features

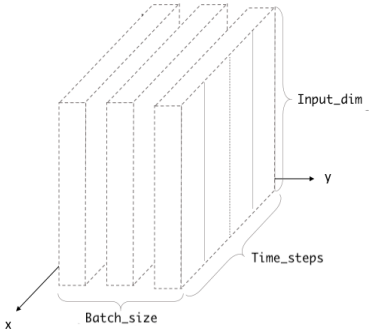

Figure 4: 3D shape training input for the LSTM

Firstly, we concatenate the training set and testing set into a big data frame. As we want to use T-minute windows of input data for predicting the target classes, T-1 last training records must be prepended to the beginning of the test set.

Secondly, we construct the 3D input using the NumPy array in Python for both the training and testing input data. The training set is further divided into two smaller subsets, one for training and one for validation.

Thirdly, 5 distinct LSTM networks will be constructed to predict the classes of five target variables. The final layer of each network would have only 1 neuron, to predict the class "0" or "1" with the activation function to be "sigmoid". The number of epochs, learning rate, lambda in L2 regularization, and the dropout rate will be carefully tuned based on the f1 score [5] on the training and validation set.

Finally, the trained models will be used to perform the classification tasks on the test set. The output would be saved in CSV format and submitted to Kaggle for performance evaluation.

3.1.3. Model assumption

Because LSTM networks are most suitable for time series data, we do not use any technique to shuffle the input data. Moreover, As the prediction is made in both current and previous data, then we need to append the T-1 last

training records to the beginning of the testing set. According to **Edstem**, the **"load"** data column in the test set is not ordered by time and it may affect the ability of the LSTM network to predict the target classes.

3.2 Model 2: 1 LSTM model with 5 output neurons

Similar to the first model, we also used LSTM networks to perform classification for the target variables. However, after getting the 3D input array for the training, validation, and testing set, this time we only built one LSTM network with 5 neurons at the output layer. With 5 neurons, we can make predictions for 5 target classes. The activation for each output neuron is a "sigmoid" function. With this model, we can save a lot of time and computation costs for the training and testing phases.

3.3 Model 3: Multi-label Convolutional Neural Network Classifier

3.3.1. Convolutional neural network (CNN) in solving time-series problems

Convolutional neural networks (CNN) are well-known for their ability to operate with spatial or 2-dimensional data. There are convolutions for 1D data, which is less well-known. This enables CNN to be used to a broader range of data types, such as texts as well as other time-series data. CNNs are able to learn features from sequential data autonomously, accommodate multiple-variate data, and generate a vector directly for time-series forecasting and classifying. As a result, one-dimensional CNNs have been shown to perform effectively on difficult time-series prediction challenges, even achieving state-of-the-art results. This inspired us to use the approach of CNN when dealing with the problem in this assignment.

A one-dimensional CNN is a CNN with a convolutional hidden layer that operates on a 1-D data sequence. Multiple convolutional layers followed by pooling layers that condense the output of the convolutional layer to the most important features. A flattening layer is then used to reduce the feature maps to a single one-dimensional vector, which acts as input for later fully connected layers.

Flatten layers are then followed by a dense fully connected layer that interprets the features extracted by the convolutional part of the model.

3.3.2. Model Architecture

The final CNN model has 2 branches (inspired by [10][11]), one uses raw training data, another uses FFT data generated from the pre-processing steps. Each Separable 1D Convolution block has 4 layers including **SepConv1D, Activation (ReLU), Batch Normalization (optional), and Drop out layer**. The output of Convolutional layers of each branch will go through Fully connected layers before being merged together. The 2 last layers in the model are fully connected layers for making predictions.

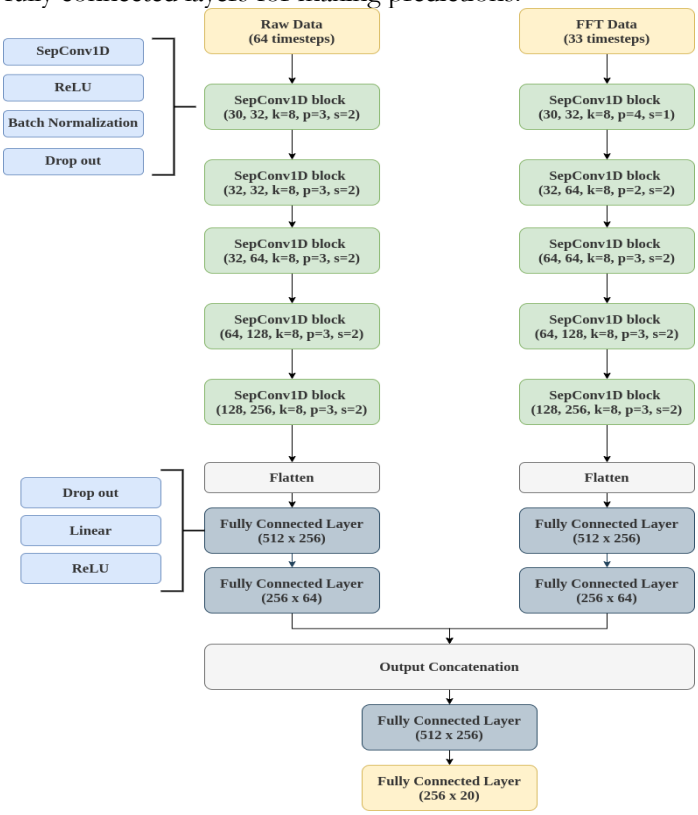

Figure 5: CNN model where k: convolution kernel size, p: padding, s: stride size

3.3.2. Model Assumption

- CNNs are less computationally expensive than RNNs because they learn in batches rather than sequentially. As a result, RNN is unable to leverage parallelization since earlier calculations must be completed first.
- CNNs are more robust when dealing with missing data in time-series because they learn data patterns within a time-window, thus making it less dependent on the data completeness.
- CNNs (with shuffling) can learn the input data in different viewpoints, while RNNs are only able to learn from the data before the target timestep.
- CNNs have been intensively studied, making RNN and LSTM obsolete.

3.4 Discussion of model difference(s)

We used CNN and LSTM in this problem, and the main differences between these two neural networks are:
+ **LSTM** makes use of its internal memory and processes sequential data to make better predictions and classification. As the training set is time-series data, the performances of the LSTM models on the training set are excellent. However, they did not maintain a high f1-score when coming to the testing set, it is understandable as the order of data in the testing set does not strictly follow the time series.

+ **CNN** with FFT and shuffling is more efficient in terms of computing speed and generalisation (shuffling gives models better viewpoints). In addition, since the test dataset is not following time series (sometimes), making CNN more suitable for solving the problem. Since the input features are mainly focused on the "load", it isn't necessary to build an extremely deep CNN.

4 Experiment setups

4.1 Model 1 and 2 (multi-label RNN and 5 binary RNN classifiers)

For the LSTM models, we choose T (timesteps window) to be 30 minutes after trying different values of T. Next, after carefully tuning all the relevant hyperparameters, including the number of layers and neurons in each layer, epochs, learning rate for the gradient descent, lambda for the L2 regularization, and the dropout rate. The best values for each of the hyperparameters is presented below:

- Timestep window T: 30 mins
- Number of features: 10
- Number of hidden layers: 3
- Number of neurons for each hidden layer: 8, 8, 8
- Number of neurons for output layer (for 5 distinct LSTM model): 1
- Number of neurons for output layer (1 combined LSTM model): 5
- Learning rate for the gradient descent: 5e-2
- Lambda in L2 regularization: 3e-2
- Dropout rate: 0

In terms of cross-validation, we further split the training set into two subsets, 70% of them were used for the training purpose and the rest 30% were the validation set. The reason why we set the ratio to be 7:3 because we need the last 30% of the training data to make sure that all the appliances are turned on for some period of time. As the data was in time series format, we kept the original order and prepended (T-1) last training record to the beginning of the validation set.

As we mentioned in the EDA part, the target classes in the training set are super imbalanced, then the main indicator we used in this problem is the F1 score. To overcome the problem of the imbalanced dataset, many different techniques have been used. According to the TensorFlow document [1], we compute the bias of each target output classes using the above formula:

$$b_0 = log_e(\frac{pos}{neg})$$

This bias is then fed into the LSTM network to reduce the effect of imbalanced data. Moreover, we also modify the training algorithm to deal with the skewed distribution of target classes. In this scenario, we allocate different

weights to the minor and major classes. To be specific, we increased the weight of the minor class while decreasing the weight for the major class [2]. The overall goal is to punish the misclassification made by the minority class severely. The **class_weight** parameter is available for the Keras model. Assume that we have **900 samples of class 0** and **300 samples of class 1**, then the **class_weight is set to be {0: 1, 1: 3}**, meaning that the model will treat the weight of class 1 three times the weight of class 0. The macro f1-score of the models after adding this parameter has improved a lot. For instance, the f1-score of the **"ev" model** has increased from **0.55 to 0.82.**

4.1 Model 3 (multi-label CNN classifier)

For the CNN models, we choose T (timesteps window) to be 64 minutes after trying different values of T. After many trials and errors with different hyperparameters such as optimizer, learning rate scheduler, number of convolution layers, kernel, padding, stride size, batch normalization, dropout rate, activation functions, etc. We came to the best setting which is:

- Optimizer: Adam [8]
- Learning rate scheduler: One cycle with [6]
- Loss function: Focal loss [7]
- Activation function: Rectified Linear Unit (ReLU)
- Dropout rate: 0.5
- Kernel, padding, and stride size: see Figure 5.

Using cross-validation would require more computing power. Since the training data is big enough, we train the CNN model on 85% of the data, and the remaining 15% is used for validating. It is possible to shuffle the data when training the CNN model; however, we got better results by not shuffling. As discovered in the EDA part, the imbalance dataset is crucial, and we believe Focal loss is suitable to handle the problem. it is robust in rescaling the loss of both dominant classes and minor classes. Focal loss can be calculated as:

$$FL\left(p_t\right) = -\alpha\left(1-\rho_t\right)^r \log\left(p_t\right)$$

where p_t is a class proportion, alpha and gamma are scaling hyperparameters.

The F1_score was computed during training using the function **f1_score()** from **sklearn** library which '**average'** parameter set to '**macro'**. The formula of **F1_score = 2 * (precision * recall) / (precision + recall).**
An additional problem when splitting the data is that we have to make sure that validating data has the same target cases as training data (sometimes validating has 16 classes but training has 20 classes). Thus we had to shuffle the data (use seed) and randomly select 10% as validation while ensuring that all possible cases are included.

5 Experimental results

+ Macro F1 score of model LSTM 1 on validate dataset: **0.74**

+ Macro F1 score of model LSTM 2 on validate dataset: **0.67**

The F1-score for the model 1 and 2 is acceptable, meaning that the models can predict target classes with low false negatives and low false positives. However, as we mentioned in the above sections, the LSTM networks work best in time series problems, but the output dataset is not properly ordered. This negatively affected the performance of these two LSTM models.

+ Macro F1 score of CNN model on validate dataset: **0.78**
CNN has higher performance than models 1 and 2 since it is not adversely affected by the validation/test dataset. Its accuracy on the validating data is 97%, which is considered as really good. On the testing data, the model can detect up to 16 classes (compared to 20 classes which it was trained on).

6 Conclusion

The optimal classifier that our team has found was CNN**.** Because this is the first time, we have to build CNN models for time-series classification, we want to **thank the FIT5149 teaching team** for giving us such a great dataset and challenge. After many trials and errors, the best feature set to us is the original dataset with 10 default features given. Extra features indeed help to improve the performance, but we have to sacrifice interpretability and

complexity. Thus, we decided not to use additional features created by the **tsfresh** library and just stick to the default and FFT features.

Our team understands the class imbalance and its effects on the performance of machine learning classification algorithms from this project. We also take into consideration the feature selections and feature scaling to maximise the classification accuracy and f1-score. Through the implementation of this project, we have chances to learn and test the performances of different machine learning algorithms, such as **LSTM** and **CNN**. In reality, we would meet many scenarios that the target class is imbalanced, such as fraud detection, medical diagnosis, or spam email classification. After this project, we gain deeper insights into classification problems and steps needed to resolve them.

For future work, we can try to rearrange the test data in time order and then apply our models to this new dataset to evaluate them. Moreover, we would like to discover the input data deeper and create more relevant features to improve the efficiency of the proposed machine learning models.

References

[1] Classification on imbalanced data | TensorFlow Core. (2021). Retrieved 30 May 2021, from https://www.tensorflow.org/tutorials/structured_data/imbalanced_data?hl=en

[2] Learning, H. (2021). How To Dealing With Imbalanced Classes in Machine Learning. Retrieved 30 May 2021, from https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/

[3] Understanding Input and Output shapes in LSTM | Keras. (2021). Retrieved 30 May 2021, from https://shiva-verma.medium.com/understanding-input-and-output-shape-in-lstm-keras-c501ee95c65e

[4] Classification of Time Series with LSTM RNN. (2021). Retrieved 30 May 2021, from https://www.kaggle.com/szaitseff/classification-of-time-series-with-lstm-rnn#-3.-Data-Pre-processing-for-LSTM-Model

[5] What is an F1 Score? - Definition | Meaning | Example. (2021). Retrieved 30 May 2021, from https://www.myaccountingcourse.com/accounting-dictionary/f1-score

[6] Smith, L. N., & Topin, N. (2019, May). Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications* (Vol. 11006, p. 1100612). International Society for Optics and Photonics.

[7] Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision* (pp. 2980-2988).

[8] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[9] Nair, V., Chatterjee, M., Tavakoli, N., Namin, A. S., & Snoeyink, C. (2020). Fast Fourier Transformation for Optimizing Convolutional Neural Networks in Object Recognition. *arXiv preprint arXiv:2010.04257*.

[10] Prithvi. (2019). *Starter Code for 3rd place Solution*. kaggle. Retrieved 05 30, 2021, from https://www.kaggle.com/prith189/starter-code-for-3rd-place-solution

[11] devforfu. (2019). *[PyTorch] Deep Time Series Classification*. kaggle. Retrieved 05 30, 2021, from https://www.kaggle.com/purplejester/pytorch-deep-time-series-classification