On The Use of Machine Learning Models for Home User Network Classification

Rushat Rai

Department of Information Engineering

The Chinese University of Hong Kong

Abstract—**DPI probe readings show that eight key network indicators in the downlink network side can tell us the kind of experience an internet user is having. Classified into two user labels (UGE having a good experience and UBE having a bad one), the goal of this work is to make a multivariate time series classification model that can use these indicator readings to predict a user into one of these buckets. Experiments with 3 different machine learning approaches (Traditional, Deep-Learning, TS-Char) show that deep-learning and TS-Char-based approaches are the most promising, and should be the primary focus of future testing with more data such that network operators may be able to discover potential complaining users in advance.**

I. INTRODUCTION

User broadband experience is quickly becoming a top priority for network operators worldwide in order to stay competitive in a market with growing demands for high performance internet. DPI probe readings show that eight key indicators on the downlink network side primarily dictate the kind of experience the user is having. These indicators, as qualified by ZTE, are [1]:

1. Indicator 1: In the first step of the three-way handshake, the time interval between the syn ack packet and the ack packet;
2. Indicator 2: In the second step of the three-way handshake, the time interval between the syn ack packet and the ack packet;
3. Indicator 3: The time interval between the ack packet and the first payload packet in the three-way handshake;
4. Indicator 4: The response delay of the first packet with payload after the establishment of TCP for multiple flows in the session;
5. Indicator 5: In TCP transmission, the actual delay of transmission from the DPI position to the user terminal;
6. Indicator 6: In TCP transmission, the transmission delay from the DPI position to the website;
7. Indicator 7: In TCP transmission, the percentage of downlink retransmitted packets in the current session;
8. Indicator 8: In TCP transmission, the percentage of upstream retransmission packets of the current session.

The problem-type is one of Multivariate Time Series Classification (hereby referred to as MTSC). MTSC is a machine learning task that seeks to capture relationships between the multiple parallel time series features in order to predict a class label for each given set of inputs. I was unable to find any existing literature that tackles the problem statement of home user network classification, but there are several studies associated with MTSC that extensively discuss and evaluate machine learning models suitable for it [2]. As such, these studies are used as the starting point for the experiments in this report.

The goal of this work is to find a model that best captures relationships between the eight indicators to classify them as belonging to either a user with a good experience (UGE), or as a user with a bad experience (UBE).

II. DATA OVERVIEW

The dataset provided by ZTE contains recorded indicator data for 500 unique users. Each record has a label identifying it as a UGE or UBE. For model development, 80% of the data (400 users) is used for training and 20% (100 users) of it is for validation. A class distribution of 50-50 UGE/UBE is maintained across the training and validation sets.

The data provided for this problem statement is quite challenging for a few reasons:

a) The sampling rate and time range of the records are not standard;

b) Some records are extremely dense while others are quite sparse. The largest record contains 43828 observations, whereas the smallest one has just 566. Uneven-length time series are incompatible with most state-of-the-art algorithms.

c) Some records contain multiple observations for a single time stamp;

d) We are limited to a sample of only 500 unique users for training and validation;

e) Indicators of users with good experiences are not completely free of outliers. This makes the distributions of UGE and UBE indicators quite similar [Fig. 3] and as a result are difficult for statistical models to differentiate between.

A rule of not removing outliers from UGE data is maintained across experiments in order to preserve the integrity of the problem statement: - in practical use, the indicators of users with good experiences are not completely free of outliers, and manual removal of outliers from UGE data would introduce bias.

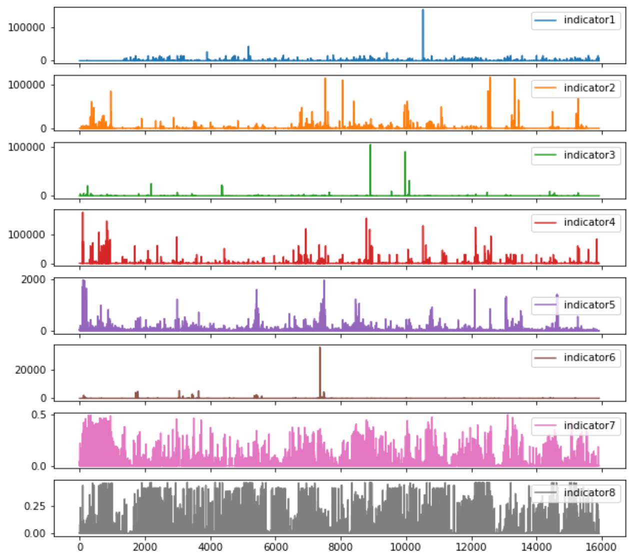


Fig. 1. Example UBE Indicators Plot

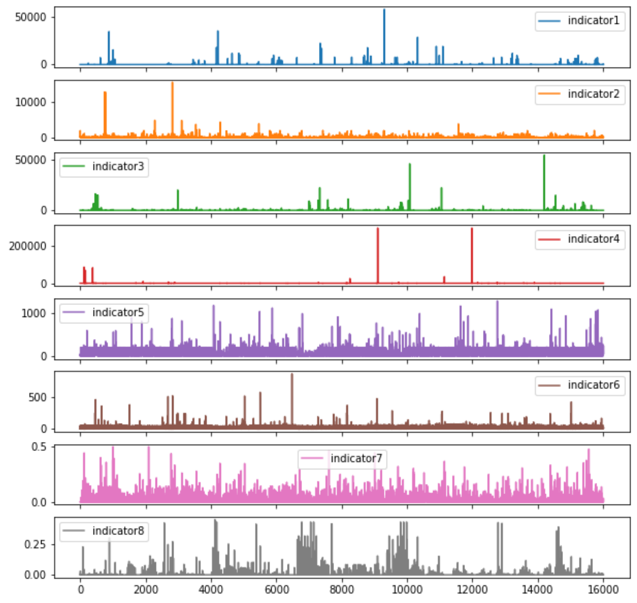
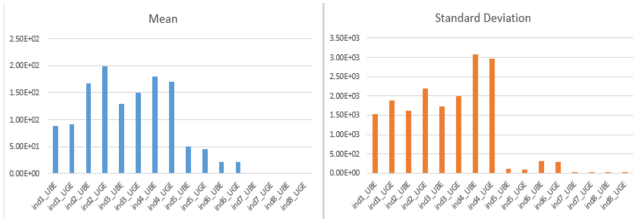
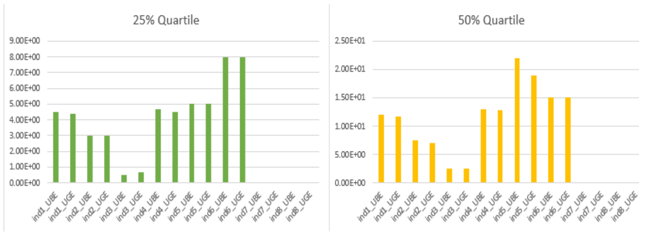


Fig. 2. Example UGE Indicators Plot





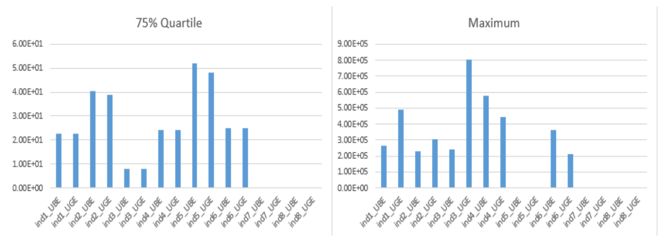


Fig. 3. Indicators Agg. Descriptive Statistics Comparison

III. EXPERIMENTS

The experiments conducted can be broadly divided into three main categories – traditional MTSC models, deep-learning-based MTSC models, and Time Series Characteristic models (hereby TS-Char models).

A. *Traditional MTSC Models*

Traditional MTSC models refer to machine learning models designed to classify multivariate time series signals directly.

I. *Data Preprocessing Pipeline*

1. Linear Interpolation

Linear interpolation is a regularization technique that resamples time series to a fixed interval. Applying linear interpolation to this dataset is key since it helps to deal with both the challenge of overly dense time series and uneven sampling rates by resampling them to standard 5-minute intervals. Figure 3 shows an example of linear interpolation resampling a time series to 15-minute intervals (the gray marks are the original data points and the orange marks are the resampled ones).

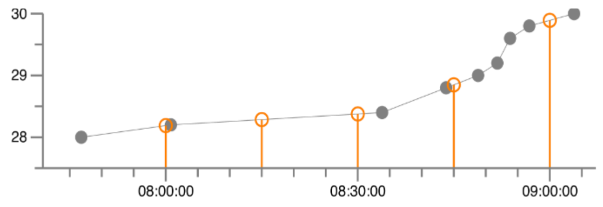


Fig. 4. Linear Interpolation Visualization [3]

After linear interpolation, the indicator records contain only a single observation per timestamp with no more than ~2000 total unique observations (up to 95% reduction in length while preserving important characteristics for classification).

2. Z-Normalization

Z-normalization is a technique in which all elements of the input vector are transformed into an output vector whose mean is approximately 0 while the standard deviation close to 1.

$$x_i' = \frac{x_i-\mu}{\sigma}, where\ i\ \epsilon\ N \qquad (1)$$

According to most recent work concerned with analyzing time series structural patterns, z-normalization is an essential preprocessing step that allows a model to focus on structural similarities/dissimilarities rather than on the amplitude-driven ones [4]. Since the units used in the indicator columns are not the same and the scales differ, z-normalization is necessary for this problem.

3. Low-Noise Padding

Low-noise padding is a technique which adds a low amplitude noise to the suffix of shorter time series to obtain sequences as long as the longest one. It is similar to zero-noise padding where the padding is done with zero values. Zero-noise padding was considered for evaluation as well, however, given its historically worse results, it was skipped. A padding with amplitude $1e^{-6}$ is applied to fix the issue of uneven-length time series in the data.

In comparison to alternatives such as uniform scaling and ARIMA-based future value forecasting, low-noise padding is the easiest to implement and gives the best performance for this type of problem [5]. The main drawback of low-noise padding is that it tends to be unsuccessful when the length of time series differs by a lot [6], but this issue is already dealt with by linear interpolation, which makes it a perfect next-step in the pipeline.

Strictly speaking, there are some classifiers capable of directly processing varying length time series, so this step is not required for all models. However, they tend to underperform compared to their alternatives [2]. Research suggests that almost any preprocessing strategy is better than none for time series that differ due to differing fixed sampling frequencies, which is the case for this dataset [6].

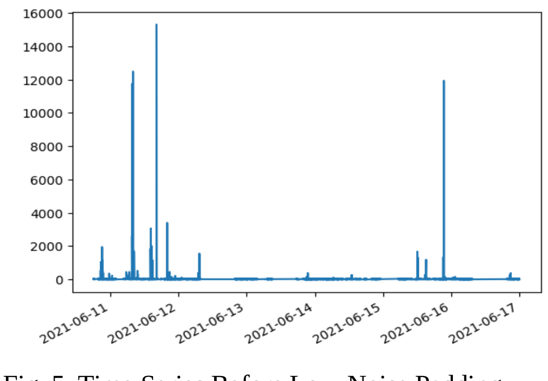


Fig. 5. Time Series Before Low-Noise Padding

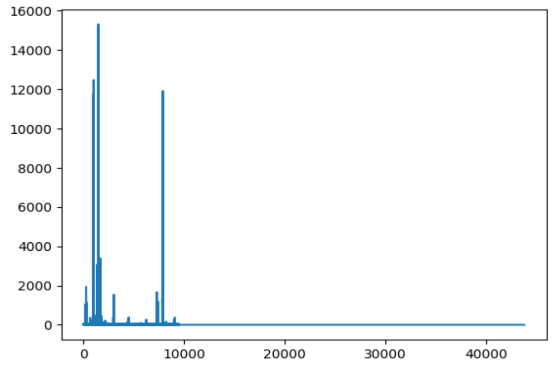


Fig. 6. Time Series After Low-Noise Padding

II. Results

ROCKET is the best ranked and fastest classifier that is the recommended choice for MTSC problems [2]. As such, the sktime implementation [7] was used as a baseline for MTSC models. Unfortunately, this model only yielded an accuracy of 0.4 on the validation set. The speculated reason is that this because of the UGE outliers making the statistical distributions of UGE and UBE very similar. Tests with other MTSC models such as DTW-KNN and HIVE-COTE also yielded similarly poor results, reinforcing this hypothesis. As such, traditional MTSC methods are not a recommended method to model this dataset.

B. Deep-Learning Models

Deep learning models are known to be able to perform automatic feature engineering, which could prove to be helpful for this dataset where besides outliers and frequency components there could also be deeper predictors not immediately obvious to humans or traditional statistical methods.

I. Data Preprocessing Pipeline

The data preprocessing steps for the deep-learning models are the same as for the MTSC models since the data requirements are the same.

II. Model Architecture

I opted to use a Long-Short-Term-Memory Recurrent-Neural-Network (LSTM RNN) architecture for this problem. It is a well-established model for deep-learning-based time series model and can model deeper relationships than MTSC models. The hyperparameters of the best model were as follows:

- Epochs = 136. Running the model up to 1500 epochs showed no improvement in validation accuracy;
- Hidden layers = 3;
- Hidden layer size = 256;
- Dropout = 0.75. High dropout tends to show better results for MTSC problems;
- Loss = Cross Entropy Loss;
- Optimizer = Adam;
- Learning rate = $1e^{-3}$;
- Batch size = 10.

The total parameters in the final model are 1.3 million. The model takes ~1 hour to find the best checkpoint on a single P5000 (including data preprocessing and model training).

III. Results

The validation accuracy of the best checkpoint is 0.58. This is already a vast improvement over traditional MTSC models, presumably due to deep neural networks being able to map more underlying features than traditional statistical approaches. An interesting pattern can be observed in the model performance graphs:

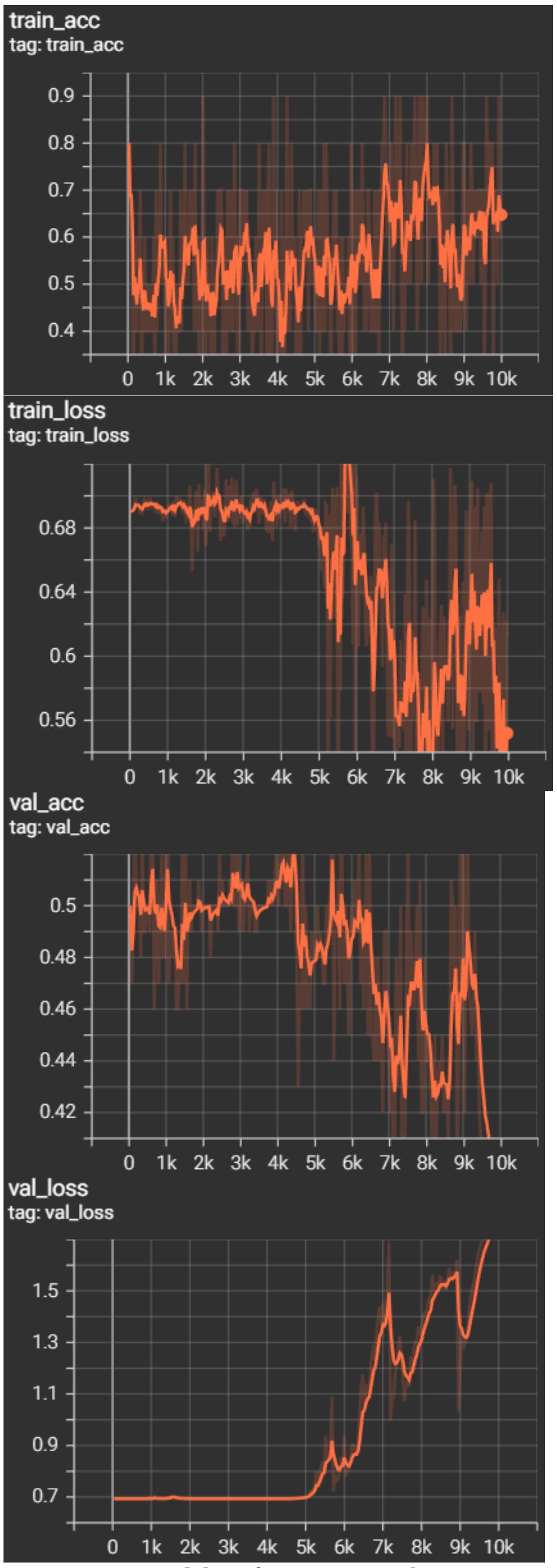


Fig. 7. RNN Model Performance Graphs

After ~4000 steps the training accuracy and loss stagnates while the validation loss begins rising rapidly- the model is unable to find more predictive features and begins overfitting. This observation might imply that the small number of unique user samples is limiting this model. Overall, the LSTM RNN is promising, and given more data, it might be able to achieve a much higher accuracy

C. *Time Series Characteristic Models*

TS-Char models rely on extracting descriptive statistics such as maximum, minimum, median, mode, number of peaks, Fourier transform coefficients, etc. from the time series and then plugging those as input features into traditional machine learning models not typically designed for MTSC. This is a particularly robust approach since it can work on time series of any length or sampling frequency without any preprocessing required and minimal computational overhead.

I. *Data Preprocessing*

As mentioned above, this method does not require any data preprocessing. There are some preprocessing steps required for PCA which is used in the TSFresh approach, and those are discussed in the relevant section below.

II. *Model Architecture*

To test the TS-Char method, I used two different approaches to extract characteristics from the time series. The first is a manual feature extraction approach, where I extract a defined set of features. The list of features extracted from each indicator for each user is as follows:

1. Mean;
2. Median;
3. Standard deviation;
4. 25th quartile;
5. 50th quartile;
6. 75th quartile;
7. Maximum;
8. Skew
9. Kurtosis;
10. Top 10 Fourier coefficients;
11. Outlier percentage as detected by Local Outlier Factor (number of neighbors set to 5).

The second approach is with the help of the library TSFresh [8]. TSFresh automatically extracts thousands of features from the data to give new insights into the time series and their dynamics. To keep the run time reasonable, the set of features to calculate was set to "EfficientFCParameters," which calculates all features except the ones which are marked with the "high_comp_cost" attribute. Running TSFresh with these parameters on the dataset results in 6264 extracted features within 2 hours on an 8 core CPU machine. For this model only, a train-val-test split of 60-20-20 is used (the test set is the same as the validation set used in previous approaches as defined by ZTE). This brings the final dimensions of the training data to 300x6264. This is a classic case of the Big-p, Little-n (p>>n) problem, where the number of features is much greater than the number of observations, which violates the default assumptions of most machine learning algorithms. To deal with this, a version of this approach with min-max normalization followed by Principal Component Analysis (PCA) for dimensionality reduction is also evaluated. After PCA, the shape of the training data is 300x397, which is more reasonable.

For both approaches, XGBoost with default hyperparameters is used as the classifier due to its well-known robustness and good performance on all sorts of classification problems. Hyperparameter tuning attempts for all methods did not offer improvement worth the computation time.

III. Results

1. Manual Feature Extraction + XGBoost

This model achieves a 10-fold cross validation accuracy of 0.51 and a test accuracy of 0.46. This is likely due to the same problem of UGE outliers that we saw in the MTSC method.

2. TSFresh Feature Extraction + XGBoost

This model achieves a test accuracy of 0.54. Other metrics are in the table below:

	Precision	Recall	F1Score	Samples
UBE	0.53	0.66	0.59	50
UGE	0.55	0.42	0.48	50

Table 1. TSFresh + XGBoost Metrics

The high recall and low precision on UBE are a good sign, since it means the system is getting most predicted labels correct when compared to the training labels, even though there are very few results.

On UGE the performance clearly tanks because of the outliers, and the recall is higher than the precision, which suggests the system is returning many results but with most predicted results incorrect.

3. PCA + TSFresh Feature Extraction + XGBoost

This model achieves a test accuracy of 0.65, beating the LSTM RNN to be the best model by far. Using PCA gives a clear improvement in results. Other metrics are in the table below:

	Precision	Recall	F1Score	Samples
UBE	0.66	0.62	0.64	50
UGE	0.64	0.68	0.66	50

Table 2. PCA + TSFresh + XGBoost Metrics

This model's great performance in a way proves the hypothesis from the RNN's results: there are underlying features in the indicators that are difficult to map using surface level characteristics and traditional statistical models. It appears that PCA greatly improves the performance on UGE and brings the precision and recall much closer to each other.

Unfortunately, PCA makes it impossible to interpret feature importance to further analyze this model, but it is a worthwhile performance tradeoff.

IV. Conclusion

In this work, I experiment with various machine learning methods to best predict class labels of UGE/UBE given a set of eight network indicators. After evaluating approaches in three main categories (Traditional MTSC, Deep Learning, TS-Char), a TS-Char approach with a combination of PCA, TSFresh and XGBoost offers the best result with an accuracy of 0.65. The deep learning-based LSTM RNN approach is also promising, and a lot of evidence discovered in this report suggests that it will be able to achieve much better performance and maybe even beat out the best current approach given more unique user data. An ensemble-based approach that extracts specific features from each indicator based on domain knowledge could also be an interesting approach to explore.

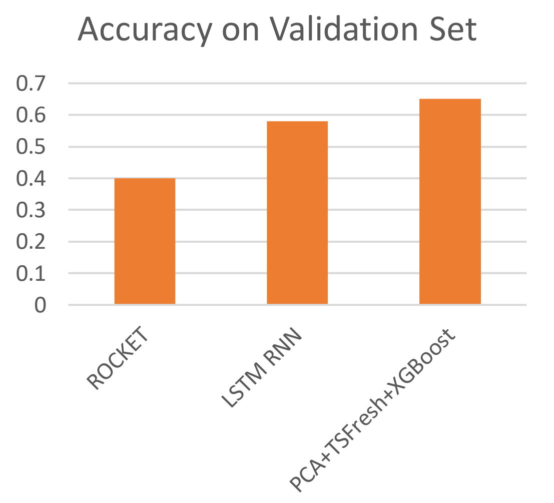


Fig. 8. Best Model from Each Approach Comparison

References

[1] ZTE, "ML5G-PS-012: Classification of Home Network Users to Improve User Experience," *ITU AI Challenge*. [Online]. Available: https://challenge.aiforgood.itu.int/match/matchitem/73.

[2] A. P. Ruiz, M. Flynn, J. Large, M. Middlehurst, and A. Bagnall, "The great multivariate time series Classification bake off: A review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 35, no. 2, pp. 401–449, 2020.

[3] Mstringer, "Python regularise irregular time series with linear interpolation," Stack Overflow, 27-Sep-2016. [Online]. Available: https://stackoverflow.com/a/39730384/6453452.

[4] P. Sinin, "Z-normalization of time series.," *Z-normalization | SAX-VSM*, 27-Oct-2016. [Online]. Available: https://jmotif.github.io/sax-vsm_site/morea/algorithm/znorm.html.

[5] A. Bier, A. Jastrzebska, and P. Olszewski, "Variable-length multivariate time series classification using Rocket: A Case Study of incident detection," *IEEE Access*, vol. 10, pp. 95701–95715, 2022.

[6] C.W. Tan, F. Petitjean, E. Keogh, and G. I. Webb: "Time series classification for varying length series", 2019; [http://arxiv.org/abs/1910.04341 arXiv:1910.04341].

[7] Markus Löning, Anthony Bagnall, Sajaysurya Ganesh, Viktor Kazakov, Jason Lines, Franz Király (2019): "sktime: A Unified Interface for Machine Learning with Time Series"

[8] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, "Time series feature extraction on basis of scalable hypothesis tests (tsfresh – a python package)," *Neurocomputing*, vol. 307, pp. 72–77, 2018.