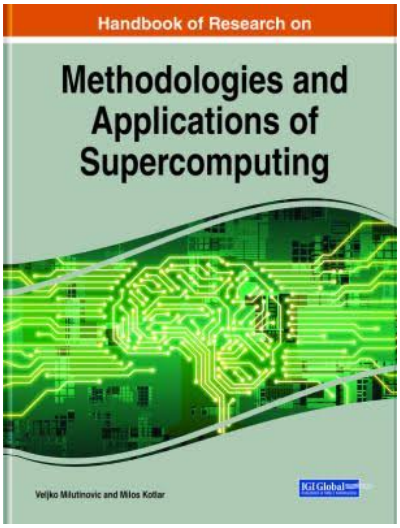


Chapter 16

Efficient End-to-End Asynchronous Time-Series Modeling With Deep Learning to Predict Customer Attrition

Victor Potapenko
Florida International University, USA

Malek Adjouadi
Florida International University, USA

Naphtali Rishe
Florida International University, USA

ABSTRACT

Modeling time-series data with asynchronous, multi-cardinal, and uneven patterns presents several unique challenges that may impede convergence of supervised machine learning algorithms, or significantly increase resource requirements, thus rendering modeling efforts infeasible in resource-constrained environments. The authors propose two approaches to multi-class classification of asynchronous time-series data. In the first approach, they create a baseline by reducing the time-series data using a statistical approach and training a model based on gradient boosted trees. In the second approach, they implement a fully convolutional network (FCN) and train it on asynchronous data without any special feature engineering. Evaluation of results shows that FCN performs as well as the gradient boosting based on mean F1-score without computationally complex time-series feature engineering. This work has been applied in the prediction of customer attrition at a large retail automotive finance company.

DOI: 10.4018/978-1-7998-7156-9.ch016

INTRODUCTION

Time-series (TS) data can be classified as synchronous vs. asynchronous, co-cardinal vs. multi-cardinal, and even vs. uneven (Wu et al., 2018). Synchronous TS data are aligned on the time dimension, while asynchronous events are not. Co-cardinal event sequences are aligned on the first and last elements. TS are classified as multi-cardinal if one of the sequences is longer than another. Multi-cardinal sequences are synchronous if the remaining elements are aligned. TS are defined to be even when data points are distributed evenly over time. Unless otherwise specified, we use asynchronous as a general term to describe asynchronous, multi-cardinal, and uneven sequences.

Our instantiation of this problem is a supervised multi-class classification with stationary and TS data. As a case study, we use a dataset comprising more than one million car loan histories. Our TS are asynchronous, multi-cardinal, and uneven. Stationary data is comprised of context data, such as vehicle and customer profiles. The target of the classification task in this case study is the customer attrition classified by type of contract termination within a 6-month horizon. The proposed approach includes database TS data extraction, imbalanced target sampling, time-series dimensionality reduction for XGBoost models, originally proposed by Chen and Guestrin (2016), that do not support multi-dimensional data, and supervised classification using a Fully Convolutional Network (FCN). (XGBoost is an open-source software library that provides a gradient boosting framework.)

TS extraction is performed with a technique called "streaming updates." The streaming updates method is used to extract value updates as TS data at variable time intervals defined by a frequency of database of field updates. This approach is discussed in Section 3a on Data Extraction. At any given time step, our extracted data is significantly imbalanced towards active accounts that are not terminated. In Section 3b, we outline a sampling approach that mitigates class imbalance in training set and prevents data leakage into the evaluation set by using different sampling techniques designed to simulate real-world conditions.

The input size and frequency of TS varies widely among features extracted with the streaming method. Some features frequently occur at regular time intervals throughout account lifetimes; others only occur a few times at highly irregular intervals. As a result, the products of join operations cause input space to increase dramatically, which leads to high memory and computational requirements for their manipulation and creates the necessity to use imputation techniques to eliminate sparsity. The scope of the present work is limited to the comparison of the non-deep learning approach that uses TS statistics to reduce time dimension, versus the deep learning approach with FCN trained on zero-imputed TS data. In Section 3c, we describe the XGBoost and FCN models. Section 4 outlines evaluation results followed by a discussion in Section 5.

BACKGROUND

Time-series sequences have been modeled with Hidden Markov Models (HMMs) (Rabiner, 1989; Ephraim & Merhav, 2002) and Bayesian Networks (BNs) (Heckerman et al., 1995; Nielsen et al., 2009). However, HMMs and BNs are not designed for asynchronous sequences because they require specification of a constant time interval between consecutive events (Wu et al., 2018). Asynchronous data needs to be reshaped and synchronized to fit HMMs and BNs. TS can be reshaped to synchronicity at the data preprocessing step. Reshaping and synchronization methods often obfuscate original data or create artificial data points that are not initially present in the dataset. This results in information loss and requires

imputation techniques that incur significant computational and memory costs when datasets are large, and the degree of synchronicity is low.

Prior research by Wu et al. (2018) classifies asynchronous properties of TS and evaluates sequence synchronization methods and relative time representations by training unmodified Recurrent Neural Networks (RNN) with Long Short-Term Memory (LSTM) cells to perform comparative analysis. Synchronization via state (sync-state) transforms the sequences into synchronous and co-cardinal, but uneven. Synchronization via binning transforms the sequences into synchronous, even, and co-cardinal. Results demonstrate that, depending on the dataset, the sync-state or sync-bin synchronizations produce better results with elision, when all relative timing information is removed. These results are used as a baseline to compare to relative-time representations: Markov, Landmark-any, and Landmark-own. Markov and Landmark-any take the history parameter that defines how many prior values to consider. The Markov relative-time representation does not consider feature values, while the Landmark-any skips the features with zero values. Both Markov and Landmark-any capture bursts of activity. The Landmark-own relative-time representation only assumes some dependence between each of the features by taking relative times between non-zero values of each feature individually. Landmark-own has performed best on AUC on both datasets. (In Machine Learning, AUC is the area under the receiver operating characteristic curve.) However, a sync-bin baseline with elision has outperformed all other models on the PhysioNet dataset based on recall and F1 score. Unmodified LSTM with Landmark-own relative time representation has scored at AUC of 0.6464 on the PhysioNet dataset.

Another approach to modeling multivariate asynchronous data is to represent the problem as Value Missingness; the representation with missing values assumes that missingness is informative. GRU-D (Che et al., 2018) is an RNN-based approach to informative missingness in asynchronous data using modifications to a Gated Recurrent Unit (GRU) (Cho et al., 2014) and masking the data representation. TS are represented by four vectors: feature values, time stamps, missingness indicator, and time interval. GRU cells are modified with a hidden state and input decay terms. The modification aims to capture the property of a feature to return to some default value after a number of steps in the absence of new observations. In GRU-D, the influence of input variables fades away as more missing values are encountered. This model is specific to data with some assumed default values. We cannot make default value and fading influence assumptions uniformly across all sequences in our dataset. The proposed GRU-D model performs at AUC of 0.8370 when evaluated on the PhysioNet dataset that contains 8000 records of intensive care unit (ICU) records.

Phased LSTM (Neil et al., 2016) extends the LSTM unit by adding a time gate. A rhythmic oscillation is specified by three learnable parameters that control the opening and closing of the time gate. Phased LSTM performs updates at irregular time points on asynchronous data. Phased LSTM takes timestamps as a separate input tensor that controls the time gate. This architecture has outperformed LSTMs and Batch-normalized LSTMs on a set of artificial benchmarks, such as the sine wave frequency discrimination at high resolution and asynchronous sampling rates. This architecture does not require any additional preprocessing steps, because it fits asynchronous time-series data by design.

Convolutional Neural Networks (CNN) are state-of-the-art in feature extraction from images. The input structure of a typical CNN allows for multi-dimensional input, which fits the data structure of TS. A Significance-offset Convolutional Network (SOCNN) (Binkowski et al., 2017) uses a representation of asynchronous TS with feature indicator vectors, feature value vectors, and duration vectors. Duration vectors contain values that indicate time elapsed since the previous observation of any feature in the entire feature space. The network scheme consists of significance and offset convolutional subnetworks.

Outputs of subnetworks are combined via the Hadamard Product. This network performs better on asynchronous data when benchmarked against unmodified CNN, ResNet, and LSTM on the same input data, but performs worse on synchronous time-series data.

In recent years, deep learning architectures such as RNNs and CNNs have been adapted to modeling asynchronous TS. By design, RNNs and CNNs ingest multi-dimensional input and can be fit to TS. In our experience, RNNs are challenging to train with asynchronous data due to input sparsity. In our research, we implement an FCN with one-dimensional kernels that differs from a CNN by not using any max-pooling layers and taking advantage of one global average pooling layer that has the advantage of making the model interpretable.

METHODOLOGY

In this section, we describe the methodology we have used to solve the problem of classifying asynchronous TS extracted from the case-study company database, sampled to reduce class imbalance, and modeled with XGBoost and FCN to predict account outcomes within a given time horizon. We use this methodology to predict customer attrition within a 6-month horizon, given the profile and timestamped account activity data stored in the company database.

Data Extraction, Characteristics, and Synchronization

We extract data from a database that contains records of the customer profile, dealer profile, vehicle profile and valuation, vehicle repair orders, account financials, delinquencies, and customer payoff amount requests. All records include two fields ("from" and "to" dates) with timestamps that indicate the time period during which the record is current. This database feature enables the extraction of TS data from a series of database field updates. We categorize data into two major types based on whether or not it changes over time: context data and time-series data.

Context data, such as customer profile, dealer profile, and vehicle profile, does not change over time. Only the latest timestamps are used to extract this type of data. We control for data leakage during exploratory analysis to ensure that this data does not change when the system marks an account as terminated. TS data (such as vehicle valuation, repair orders, account financials, delinquencies, and customer payoff requests) is extracted from the database using a lag function that detects a change in a field value and extracts the changed values along with the change timestamps.

Extracted multivariate daily TS is asynchronous, multi-cardinal, and uneven. Vehicle valuations and account financials are updated frequently and periodically, while repair orders, delinquencies, and customer payoff requests are infrequent and occur sporadically. This asynchronous data form is common in sensors with variable sampling rates. Common integration methods introduce noise by up-sampling or lose information by down-sampling. Both sampling methods adversely affect model performance. In our work, we zero-pad TS for FCN input and reduce dimensionality via calculated TS statistics for input to XGBoost.

Training and Evaluation Sampling

In production, the model does not have access to future data; therefore, providing future data or embedded knowledge about future data during the evaluation phase causes data leakage and low performance on out-of-sample datasets. Given this constraint, we design a sampling process that simulates the feature state available at a specified point in time and computes target values for a specified horizon.

The classification target is represented by termination codes that include active accounts and accounts terminated for various reasons, such as early payoff or scheduled maturity. Training data consists of context data and a set of variable-length TS with termination codes conditioned on a 6-month time horizon. The sampling method for the training dataset is not subject to the same constraints as the evaluation dataset in the production environment if a model trained on such a dataset performs well on evaluation set and in production. We use two different methods for sampling training and evaluation sets. In our case, this allows us to decrease the size of the training dataset, lowering memory complexity, and to mitigate class imbalance, improving classification accuracy.

Our dataset consists of 1.5 million accounts active over five years. Our algorithm splits the dataset into the training and evaluation sets by the split date t_{split}. The algorithm samples the training dataset from the distribution of accounts active between t_0 and t_{split} (Figure 1). The algorithm samples the evaluation dataset from the distribution of accounts active between t_{split} and t_n (Figure 2).

Figure 1. Sample Training Data. Each sample is assigned an $observation\_begin\_date = contract\_open\_date$. *All samples where* $contract\_open\_date > t_{split}$ *are discarded from the training set (sample E). If* $termination\_date < t_{split}$ *then* $observation\_end\_date = termination\_date - \left(horizon - random\_offset\right)$ *and* $target\_class = termination\_reason\_code$ *(samples A and B). Otherwise,* $observation\_end\_date = t_{split} - random\_offset)$ *and* $target\_class = 'active'$ *(samples C and D).*

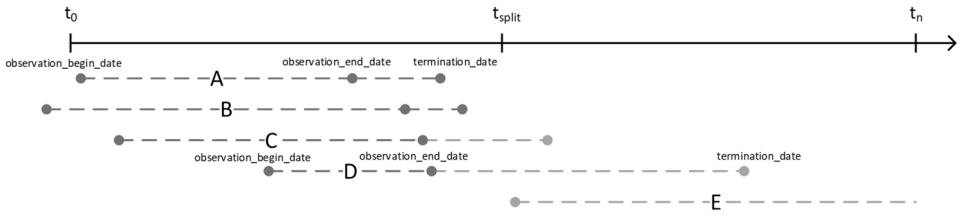
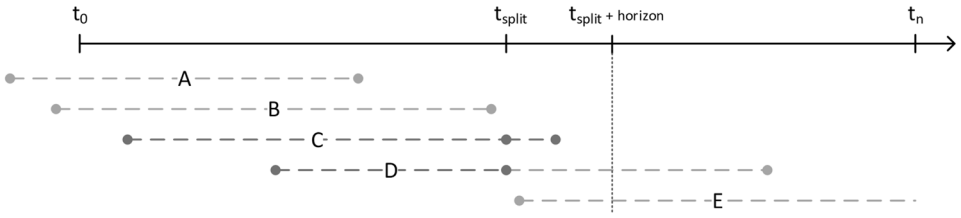


Figure 2. Sample Evaluation Data. Each sample is assigned an $observation\_begin\_date = contract\_open\_date$. *All samples where* $contract\_open\_date > t_{split}$ *are discarded from the evaluation set (sample E).* $observation\_end\_date = t_{split}$, *and* $target\_class = termination\_reason\_code$ *if* $termination\_date < \left(t_{split} - horizon\right)$ *(sample C). Otherwise,* $target\_class = \text{'}active\text{'}$ *(sample D).*



The training set sampling method yields a more class-balanced dataset because it selects all the terminated accounts from the history, while it samples the active accounts only from accounts active at t_{split} timestep. Our experiments show that balanced class distribution has a robust positive impact on model performance. For an account to be classified as terminated, its termination date must occur within the horizon window. We add a random offset to the calculation of observation end-date in order to simulate random sampling and to counter the effects of the timewise-anchored sampling at the known termination date.

The evaluation set sampling method simulates data that the model encounters in production. The observation end-date is the date when the model predicts using all currently active account data available at that time. In our evaluation sampling case, the observation end-date is equal to t_{split} and the classification is made based on whether the termination date occurs within the predetermined time horizon. We use this evaluation data sampling method to extract multiple out-of-sample evaluation splits by altering t_{split}, which allows us to evaluate future model accuracy decay and generate multiple cross-validation splits.

The class imbalance between active and terminated accounts is 10:1 in the evaluation set. In the training set, the imbalance is 1:1.8. When the evaluation set sampling method is applied to sample from the training set, there are two options: sample at fixed intervals and sample at random intervals. The problem of class imbalance remains. The sampling process may miss possibly valuable samples of an underrepresented class, causing the under-sampling of terminated class. A possible downside of the developed sampling methods is data leakage of the active class from the training set where the model learns identities of the class and classifies all previously active accounts as active in all future evaluation sets.

Modeling

XGBoost and TSFresh

We use XGBoost to create a non-deep baseline for the evaluation of our deep modeling approach. XGBoost (Chen & Guestrin, 2016) is an algorithm based on gradient tree boosting (Friedman, 2001). XGBoost algorithm trains an ensemble of decision trees in an additive manner. In most cases, it is

computationally infeasible to enumerate all tree structures and their combinations. Trees that result in most improvements in the model are added greedily to an ensemble. XGBoost builds a structure of each decision tree using a greedy algorithm that starts from a single leaf and iteratively adds split branches. Most tree-boosting algorithms use a greedy algorithm that enumerates all possible splits on all features, which makes it computationally demanding when continuous features are present in the data. XGBoost uses an approximate algorithm to reduce computational load. In a nutshell, the approximate algorithm proposes candidate splitting points according to feature distribution, maps continuous features into buckets based on the splitting points, aggregates the statistics, and finds the best solution based on the aggregated statistics (Chen & Guestrin, 2016).

Input to XGBoost is a two-dimensional tensor X with shape $\left(n, m\right)$, where n is a total number of samples, and m is a feature dimension.

$$X = \left(x_{0,0},\ldots,x_{i,j},x_{i,j+1},\ldots x_{i+1,j},x_{i+1,j+1},x_{n,m}\right),$$

where the element $x_{i,j}$ is a feature j that belongs to sample i and $X \subset \mathbb{R}$. TS has a three-dimensional shape $\left(n, m, t\right)$, where t is a time dimension. Therefore, XGBoost does not support TS as input. We perform feature engineering with TSFresh (Christ et al., 2018) to reduce the dimensionality of TS data to XGBoost input dimensions. We concatenate engineered TS and context data to produce a two-dimensional representation of the feature space. We train the XGBoost model on approximately 14000 features, of which 260 features belong to context data, and the remaining features are generated from TS data using TSFresh. (TSFresh is an open-source software library that provides a time-series statistics framework.) Training time is 2.6 hours on a 32-core instance.

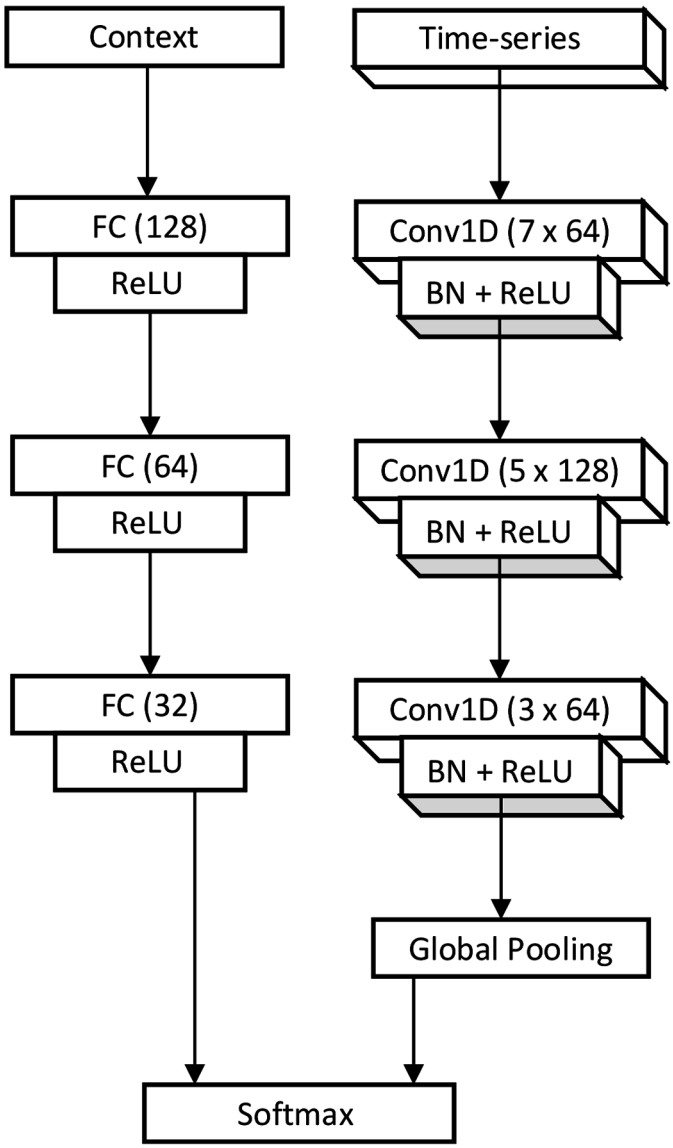
We use initial training to eliminate most features based on feature-importance scores provided by XGBoost, with the importance threshold of 100 and the correlation of greater than 0.95. This process returns 367 features. We perform final feature selection using backward feature elimination based on aggregated SHAP values (Shapley Additive Explanations) (Lundberg & Lee, 2017) that assign each feature an importance value for a class prediction of the sample. Our algorithm eliminates one feature with the lowest aggregated SHAP value, and the model is retrained on remaining features at each elimination step until the model accuracy starts to decline significantly. The reason we do not use SHAP on the initial set of 14000 features is the computational and time complexity of calculating the SHAP values and effectively training 14000 models. The feature selection process leaves us with the ten most important features according to the aggregated SHAP values from all classes. We perform parameter tuning on the remaining features via random search sampling from parameter space of the maximum tree depth, learning rate, and number of trees. We set the multi-class logarithmic loss as our evaluation metric for the early stopping based on the evaluation dataset and use softmax (7) as the classification objective function.

Fully Convolutional Network with Context

Fully Convolutional Neural Networks (Wang et al., 2017) have been successfully validated for TS classification on 44 datasets from the University of California, Riverside/University of East Anglia archive. Unlike CNN, FCN does not contain any local pooling layers and uses the Global Average Pooling (GAP)

layer instead of the traditional Fully Connected (FC) layer. An FCN consists of three convolutional layers, where each convolutional layer is followed by a batch normalization (BN) layer and a rectified linear unit (ReLU) activation. This structure outputs to a GAP layer, followed by a softmax activation for supervised classification.

Figure 3. Fully Convolutional Network with Context. A basic convolutional block consists of a convolutional layer (i.e., Conv1D (8 x 64) with 64 one-dimensional kernels of size 8), followed by a batch normalization (BN) layer and ReLU activation function. Context data is input into the multilayer perceptron that consists of fully connected layers (i.e., FC (128) with 128 neurons), followed by the ReLU activation function. Outputs of FCN and MLP are inputs to the softmax activation function.



A basic convolution block consists of a convolutional layer, BN, and ReLU, where W and b are the weight and bias vectors:

$$y = W \cdot x + b$$

$$s = BN\left(y\right)$$

$$h = ReLU\left(s\right) \tag{1}$$

Our model (Figure 3) has three convolutional blocks (1) with the {64,128,64} numbers of one-dimensional kernels of sizes {8,5,3} without striding. The kernels are used to extract TS features.

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} y_i \tag{2}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}\left(y_i - \mu_{\mathcal{B}}\right)^2 \tag{3}$$

$$\hat{y}_i \leftarrow \frac{y_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{4}$$

$$s_i \leftarrow \gamma\hat{y}_i + \beta \equiv BN_{\gamma\beta}\left(y_i\right) \tag{5}$$

BN (2,3,4,5) is applied to speed up convergence and improve generalization (Wang et al., 2017), where m is the size of mini-batch $\mathcal{B}$, $\mu_{\mathcal{B}}$ is the mini-batch mean, $\sigma_{\mathcal{B}}^2$ is the mini-batch variance, and $\hat{x}_i$ is the normalized mini-batch; γ and β are parameters to be learned via backpropagation; y_i is the scaled, shifted, and normalized output of the convolutional layer over the mini-batch.

ReLU (Hahnloser et al., 2000) is an activation function that has shown to improve the performance of deep neural networks. It prevents saturation of gradients in deep networks by thresholding values at 0 (6).

$$ReLU\left(s\right)_i = \max\left(0, s_i\right) \tag{6}$$

The input of a convolutional layer has a shape of $\left(b, c, l\right)$, where b is the sample batch size, c is the number of channels, and l is the channel length. TS data has an identical shape $\left(n, m, t\right)$, where the

samples are equivalent to the batch size, the features to the channels, and the time to the length of each feature. For our XGBoost model, we reduce TS data with TSFresh into the two-dimensional space and concatenate the context data, because the data dimensionalities match. The context and TS data cannot be concatenated for an FCN model because of the dimensionality mismatch. One way to integrate context data into an FCN model is to duplicate context data across the l dimension of a convolution. This method results in added memory complexity because it duplicates over 260 context features in addition to 53 TS features across all time steps of the sample. We integrate context by concatenation of a multilayer perceptron (MLP) that consists of three FC layers with the {128,64,32} neurons and ReLU activation, each followed by a dropout layer with the rate of 0.2. The output of concatenation of FCN and MLP is connected to the final softmax (7) output layer that normalizes its inputs into a probability distribution over the number of classes K, where h_{n-1} is the output of the last convolutional block.

$$softmax\left(h_{n-1}\right)_i = \frac{e^{h_{n-1_i}}}{\sum_{j=1}^{K} e^{h_{n-1_j}}} \tag{7}$$

The FCN model uses cross-entropy loss function (8) to compute loss and backpropagate gradients throughout the network, where $softmax\left(h_{n-1}\right)_i$ is the output of the softmax function (7) for class i, C is the total number of classes, and t_i is the ground-truth class label.

$$CE = \sum_{i}^{C} t_i \log\left(softmax\left(h_{n-1}\right)_i\right) \tag{8}$$

We implement the FCN model with context using Keras and Tensorflow 2.0. We train this model on a 32-core compute instance with 256 gigabytes of memory. The training set consists of 963,712 records with 260 context features and 53 time-series features, where each feature is limited to the length of 365, corresponding to days in one year. The evaluation set is significantly imbalanced and consists of 456,128 records. We use a data generator to iterate over all instances to conserve disk space by generating zero-filled dense input from the sparse input saved in a file. Tensorflow Dataset API is used to optimize data pipeline by prefetching, shuffling, batching, and caching training and evaluation data at runtime. The caching stores all data generated during the first pass over the dataset into memory. Once we cache the data during the first epoch of training, it occupies 100 gigabytes of memory, including the original sparse dataset. The first epoch training time is 1203 seconds during the first pass over the dataset. The average training time per epoch is 800 seconds with the cached dataset. We train the model with early stopping conditioned on validation loss metric to stop training before the model begins to overfit.

RESULTS

The evaluation dataset is significantly imbalanced because of the small number of contract terminations compared to active contracts at any given time horizon. Additionally, terminations are divided into four subtypes, most of which represent less than 10% of the evaluation dataset. Accuracy is not a reliable

model performance measure with imbalanced datasets. If a model classifies all samples as the majority class and misclassifies all minority classes, the accuracy would be misleadingly high.

Table 1a. FCN results

Class	Precision	stdev	Recall	stdev	F1	Stdev
0	0.8741	0.004	0.9934	0.002	0.9299	0.001
1	0.4041	0.011	0.0750	0.012	0.1261	0.017
2	0.6869	0.062	0.1095	0.028	0.1882	0.044
3	0.7005	0.106	0.0068	0.004	0.0133	0.008
4	0.3848	0.007	0.0266	0.003	0.0497	0.005
Mean	0.6101	0.038	0.2423	0.010	0.3468	0.015

Table 1b. XGBoost results

Class	Precision	stdev	Recall	stdev	F1	stdev
0	0.8950	0.002	0.6387	0.006	0.7454	0.003
1	0.1232	0.002	0.4450	0.009	0.1929	0.001
2	0.3923	0.013	0.3563	0.007	0.3734	0.010
3	0.1878	0.021	0.1544	0.015	0.1686	0.011
4	0.3574	0.013	0.0751	0.024	0.1231	0.032
Mean	0.3911	0.010	0.3339	0.012	0.3207	0.011

We evaluate model performance on five out-of-sample datasets that are sampled by shifting the split date one month forward with the time horizon of six months. At each split date, we use the time horizon to determine the contract classification. This procedure simulates real-world conditions, where the model has a task of classifying accounts given the data available up to a point in time. We report results based on the average and standard deviation of model performance across all test splits.

FCN achieves a higher unweighted mean F1-score (Table 1a) than our XGBoost model (Table 1b). F1-score is a harmonic mean of precision and recall. FCN performs much better in terms of precision by classifying the true positives of each target class while capturing less false negatives. FCN recall scores are lower because it captures fewer examples per class than XGBoost. XGBoost captures more true positives than FCN and more false negatives. The standard deviation of the XGBoost model is lower, which suggests that it is more stable over time; however, it is not significant enough to make a definitive judgment.

Precision is more important than recall in our problem domain because of the financial implications of actions taken based on model predictions. If the model predicts an active account to be terminated within the next six months, the company may decide to take action to retain the account, which may include financial incentives or refinancing. These actions have financial consequences. Profits are negatively impacted when a customer terminates their account because of financial incentives based on

an incorrect model prediction. When a model has low precision, it is more likely to include more such accounts in a target class.

DISCUSSION

In this paper, we have presented an end-to-end approach for modeling asynchronous time-series data using a deep Fully Convolutional Network with context. We have compared its performance to XGBoost. We have extracted data from a company database as time-series using timestamps to detect field value changes over time. To balance class representation in the training dataset, we have sampled all terminated accounts from three years of account histories using an offset from known target termination date for randomization. This algorithm has captured the accounts that had not been terminated before the train/evaluation split date.

Our training sampling method effectively discards many samples of the active class by only considering terminated accounts at the points of termination. A possible future direction of this research is to experiment with sampling terminated accounts at the termination, as well as immediately before the termination as active accounts. This possible future approach generates samples around the boundary between active and terminated account states, which may be significant for the model to learn to distinguish the two states with a higher degree of accuracy.

We have reduced the time dimension of our data by computing time-series statistics using TSFresh and performed backward stepwise elimination using SHAP values with XGBoost to select ten most important features. Feature engineering and backward stepwise elimination required for XGBoost was computationally expensive. We have implemented FCN with no computationally-complex feature engineering and obtained similar results with higher mean precision scores than XGBoost.

Our final feature composition and modeling results suggest that most extracted and engineered features have low predictive power. Our goal was to predict which accounts terminate within the next 6-month period, which is a market timing problem. Market timing is an especially tricky problem when information about market participants is sparse. Our dataset contained relatively few data points on the underlying market participants and their states over time, making it a more challenging task. In future work, we plan to explore the effects of additional data sources, time-series data smoothing, data sampling with better representation of active/terminated accounts' boundary, and variants of asynchronous data representation.

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of the National Science Foundation [grant numbers MRI CNS-1532061 and MRI CNS-1920182] and of J.M. Family Enterprises, Inc., and its experts Alfredo Cateriano, Ashley Taylor, Ben Fowler, Arthur Engelman, and Eddie Rivera.

REFERENCES

Binkowski, M., Marti, G., & Donnat, P. (2017). Autoregressive convolutional neural networks for asynchronous time series. *ICML 2017 Time Series Workshop*.

Che, Z., Purushotham, S., Cho, K., Sontag, D., & Liu, Y. (2018). Recurrent neural networks for multivariate time series with missing values. *Scientific Reports*, *8*(1), 6085. doi:10.103841598-018-24271-9 PMID:29666385

Chen, T., & Guestrin, C. (2016). XGBoost: a scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785–794). Association for Computing Machinery. 10.1145/2939672.2939785

Cho, K., Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 1724–1734. 10.3115/v1/D14-1179

Christ, M., Braun, N., Neuffer, J., & Kempa-Liehr, A. W. (2018). Time series feature extraction on the basis of scalable hypothesis tests (TSfresh - a Python package). *Neurocomputing*, *307*, 72–77. doi:10.1016/j.neucom.2018.03.067

Ephraim, Y., & Merhav, N. (2002). Hidden Markov processes. *IEEE Transactions on Information Theory*, *48*(6), 1518–1569. doi:10.1109/TIT.2002.1003838

Friedman, J. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, *29*(5), 1189–1232. doi:10.1214/aos/1013203451

Hahnloser, R., Sarpeshkar, R., Mahowald, M., Douglas, R. J., & Seung, H. S. (2000). Digital selection and analog amplification coexist in a cortex-inspired silicon circuit. *Nature*, *405*(6789), 947–951. doi:10.1038/35016072 PMID:10879535

Heckerman, D., Geiger, D., & Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, *20*(3), 197–243. doi:10.1007/BF00994016

Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, *30*, 4765–4774.

Neil, D., Pfeiffer, M., & Liu, S. (2016). Phased LSTM: accelerating recurrent network training for long or event-based sequences. In *Proceedings of the 30th International Conference on Neural Information Processing Systems* (pp. 3889–3897). Curran Associates Inc.

Nielsen, T. D., & Jensen, F. V. (2009). *Bayesian networks and decision graphs*. Springer Science & Business Media.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–286. doi:10.1109/5.18626

Wang, Z., Yan, W., & Oates, T. (2017). Time series classification from scratch with deep neural networks: a strong baseline. *2017 International Joint Conference on Neural Networks*, 1578-1585. 10.1109/IJCNN.2017.7966039

Wu, S., Liu, S., Sohn, S., Moon, S., Wi, C., Juhn, Y., & Liu, H. (2018). Modeling asynchronous event sequences with RNNs. *Journal of Biomedical Informatics*, *83*, 167–177. doi:10.1016/j.jbi.2018.05.016 PMID:29883623

KEY TERMS AND DEFINITIONS

Convolutional Neural Network: A type of neural network with an architecture that consists of kernels that learn to perform the matrix convolution operation on inputs to find patterns that have spatial proximity such as images or time-series.

Deep Learning: A subfield of machine learning that specializes in neural network based algorithms that have more than one hidden layer.

Fully Convolutional Network: A type of convolutional network that does not contain any dense layers in its architecture. This type of network only contains layers that are specific to convolutional neural networks, such as convolution, pooling, and batch normalization.

Gated Recurrent Unit: A modification of the long-short term memory unit with fewer parameters and no output gate.

Gradient Tree Boosting: A supervised machine learning algorithm that consists of an ensemble of decision trees.

Long-Short Term Memory Network: A type of recurrent neural network where regular neurons are replaced by long-short term memory units designed to allow deep networks to "remember" inputs over larger number of steps and mitigate the exploding/vanishing gradient problem.

Neural Network: A supervised machine learning algorithm that searches for a function to fit existing data via an iterative training process. Neural Networks are characterized by multiple hidden layers that consist of neurons with activation functions that adjust weights using the backpropagation algorithm and a loss function.

Recurrent Neural Network: A neural network with an added dimension to represent the sequence or time component of sequential or temporal data.

Time-Series Data: Are data points spread across the time dimension. Time-series data is sequential and ordered based on timestamps. Examples include series of event occurrences and temperature measurements over time.