Stockholms universitet

Predicting Insurance Premium for Private Unit-linked Product by Using Long Short-Term Memory (LSTM) Method

Nan Karlsson

Masteruppsats i försäkringsmatematik
Master Thesis in Actuarial Mathematics

Masteruppsats 2024:4
Försäkringsmatematik
Juni 2024

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Stockholm University

Mathematical Statistics
Stockholm University
Master Thesis **2024:4**
http://www.math.su.se

Predicting Insurance Premium for Private Unit-linked Product by Using Long Short-Term Memory (LSTM) Method

Nan Karlsson*

June 2024

Abstract

This thesis explores the efficacy of Long Short-Term Memory (LSTM) networks in forecasting time series data, with a particular focus on their application to real-world datasets in private Unit-linked product in the insurance industry. The study begins by establishing a theoretical foundation for LSTM models, followed by a practical application to simulated data, where they are benchmarked against the traditional time series prediction method, Autoregressive Moving Average (ARMA). The research then transitions to the application of LSTM models to real datasets. In the empirical analysis, we develop both univariate and multivariate LSTM models. To evaluate the predictive accuracy and generalizability of these models, we compare their performance against a baseline Moving Average (MA) model. The findings suggest that ARMA models may excel in certain scenarios, while LSTM networks offer robust alternative for complex time series forecasting. This is evidenced by their ability to capture the interconnectedness within the data, resulting in more accurate predictions compared to the baseline MA model.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: karlsson.nan@gmail.com. Supervisor: Filip Lindskog.

Acknowledgements

I would like to express my deepest gratitude to all those who have contributed to the completion of this thesis. First and foremost, I extend my sincere appreciation to my supervisor, Filip Lindskog, whose expertise, understanding, and encouragement added considerably to my study experience. Your guidance and support from the initial to the final level enabled me to develop a comprehensive understanding of the subject.

My sincere thanks are extended to Swedbank Försäkring AB for providing the data and the unique opportunity to research this subject. The insights into insurance products have been instrumental in facilitating my research. The access to real-world data and industry knowledge has been invaluable in shaping this thesis. I am particularly grateful for the assistance offered by André Wong, Karolis Kaupas, and Mårten Marcus, whose support and expertise have been greatly appreciated.

I am also indebted to my family for their unwavering support and encouragement throughout my study; this accomplishment would not have been possible without them. Thank you to my husband, Peter Karlsson, for believing in me and for pushing me to pursue my goals.

Finally, I would like to thank my employer Ernst & Young. The firm's commitment to fostering a learning culture and providing access to a wealth of learning resources, coupled with the flexibility offered, have been crucial in facilitating my academic endeavors. Additionally, I am immensely grateful to my colleague, Claes Sadenfors. His willingness to share his expertise and provide constructive feedback has been invaluable to the successful completion of this thesis.

Nan Karlsson
June 2024

Contents

1 Introduction

Insurance companies in Sweden offer a range of Unit-linked products designed to provide financial security and growth opportunities to individuals. A "Unit-linked product" refers to a type of investment-linked insurance policy that combines the aspects of protection and investment. The premiums paid by the policyholder are partly used to provide insurance coverage, while the remaining portion is invested in various types of investment funds. These funds are often divided into units, hence the term "Unit-linked," and the value of the policy is directly linked to the performance of these investment units. For example, a Unit-linked retirement plan, which combines life insurance with investments in market-linked funds, offering flexible retirement payouts based on fund performance. The Unit-linked products can contribute significantly to an individual's retirement, supplementing the basic state pension, and can also act as both saving instruments and life insurance policies. The premiums paid into these products constitute the backbone of future liabilities that the companies must meet; hence, forecasting these premiums accurately is a critical actuarial task that ensures the insurer's ability to fulfill its commitments.

Unit-linked products can be funded by employers, as seen in occupational pensions, or by individuals through private Unit-linked plans. Occupational pensions typically have premiums based on a percentage of the pensionable salary, leading to more predictable payment patterns. In contrast, premiums for private Unit-linked plans tend to be more variable and influenced by multiple factors. The complexity of predicting premium volumes in private Unit-linked plans is heightened by individual preferences, policyholders' behavior, economic fluctuations, demographic changes, and evolving regulatory environments. Specifically, the ability to predict these volumes accurately is key to pricing products correctly. Furthermore, such predictions are indispensable for maintaining solvency margins, determining value in force, and ensuring the long-term sustainability and competitive edge of life and pension insurance companies.

Against this backdrop, Long Short-Term Memory (LSTM) networks—an advanced type of recurrent neural network—present themselves as a promising solution for forecasting tasks. LSTMs are adept at capturing intricate patterns in sequential data, potentially offering a breakthrough in premium prediction accuracy. They hold a particular advantage in dealing with the inherent temporal aspects of premium payments and can learn from long-term sequences, which may be laden with complex, influential factors.

The aim of this thesis is to dissect the LSTM method as a tool for predicting future premiums for private Unit-linked insurance product in the Swedish insurance market. The research contemplates the capability of LSTMs and their application to this particular domain, which, if successful, could lead to vast improvements in strategic financial planning for the industry.

The thesis is organized as follows: Section 2 provides a literature review of the current methodologies used for insurance premium prediction, along with an examination of existing applications of LSTM networks in financial forecasting. Section 3 delves into the theoretical aspects of LSTM networks. Section 4 conducts an analytical exploration using simulated data to benchmark the performance of ARMA model against that of LSTM network. This comparative analysis aims to highlight the strengths and limitations of each approach in a controlled environment. Section 5 shifts the focus to real-world data, outlining the research methodology, including data acquisition, preprocessing techniques, and the LSTM modeling process, presents the empirical results, evaluates the predictive accuracy of the LSTM models, and compares them with traditional forecasting method. Section 6 provides a summary of the analyses conducted throughout the study, distilling the conclusions and insights derived from the work. It discusses the practical implications of the findings and identifies avenues for future research.

2 Literature Review

Time series prediction has traditionally relied on classical approaches such as Moving Averages (MA) or Autoregressive Moving Average (ARMA) models, as discussed by Brockwell and Davis (2002) [3]. These methods have provided a foundation for understanding and forecasting temporal data. However, challenges remain in accurately predicting premiums, especially for private Unit-linked insurance products, where premium payments can be influenced by a myriad of factors, including market volatility and policyholder behavior.

An RNN, or Recurrent Neural Network, first introduced by Elman (1990) [6], is a type of neural network designed to handle sequential data, where the output from the previous step is fed back into the model as input for the next step. This contrasts with a standard feedforward neural network, where the flow of information moves in only one direction, from input to output, without looping. However, RNNs often face challenges in learning long-term dependencies, a phenomenon attributed to the vanishing gradient problem, as described by Hochreiter (1991) [11] and Bengio et al. (1994) [2]. The issue emerges during backpropagation, where computed gradients tend to shrink as they are propagated through the network's layers. When these gradients approach zero, they provide little to no adjustment for the network's weights, particularly in the earlier layers, thereby impeding the network's ability to learn from data points that are temporally distant from the relevant output.

Long Short-Term Memory (LSTM) networks, a subclass of Recurrent Neural Networks (RNNs), have gained recognition as a potent tool for time series forecasting. Introduced by Hochreiter and Schmidhuber (1997) [12], LSTMs are specifically designed to address the limitations of traditional RNNs, such as the difficulty in learning long-term dependencies. The distinctive architecture of LSTMs, which includes memory cells and gating mechanisms, enables them to retain information over prolonged periods. This feature is particularly advantageous for modeling financial time series data, which often display long-term trends and cyclical patterns (Gers, Schmidhuber,& Cummins, 2000) [8]. The efficacy of LSTM networks in financial forecasting has been the subject of extensive research. Studies have demonstrated their effectiveness in predicting stock prices (Chen, Kuo, & Wang, 2005) [4], forecasting market indices (Bao, Yue, & Rao, 2017) [1], and analyzing cryptocurrency trends (McNally, Roche, & Caton, 2018) [16]. These studies underscore the LSTM's capability to capture the complex, non-linear relationships that are characteristic of financial time series data.

The consensus in the literature suggests that although traditional models are valued for their transparency and simplicity, LSTM networks and other advanced machine learning techniques significantly enhance predictive performance for complex time series data. Comparative studies have

consistently shown that LSTMs outperform ARIMA models, support vector machines, and other conventional benchmarks in terms of forecasting accuracy (Hansun, 2013 [10]; Fischer & Krauss, 2018 [7]).

Within the insurance domain, the application of LSTM networks has been investigated for various tasks. Lindholm and Palmborg (2021) [15] conducted a study on how to use data efficiently together with a LSTM neural network extension of the Poisson Lee-Carter model. Wüthrich (2022) [19] delved into the efficacy of LSTM networks in mortality prediction, demonstrating their potential to enhance actuarial models with their advanced pattern recognition capabilities. However, research on the use of LSTMs for premium prediction, particularly for private Unit-linked products, remains limited.

The literature indicates that LSTMs have the potential to effectively model the dynamic interplay between various factors and premium payments. Nonetheless, the scarcity of research specifically focused on LSTM applications for insurance premium forecasting highlights a significant opportunity for further investigation and validation of LSTM models in this specialized area of the insurance industry.

3 Theoretical Framework

3.1 Long short-term memory neural networks

The Recurrent Neural Network (RNN) is particularly well-suited for analyzing sequential data due to its intrinsic capacity to process inputs in an ordered sequence. This characteristic allows the outputs generated by a RNN to be influenced by the preceding elements in the sequence. However, traditional RNNs encounter challenges when attempting to capture dependencies over longer sequences—this is known as the long-term dependency problem. Addressing this limitation, Hochreiter and Schmidhuber [12] innovated the Long Short-Term Memory (LSTM) neural network in 1997 as a potent modification designed to retain information over extended intervals. Enhancement of the LSTM architecture continued with Gers et al. [8] in 2000, who introduced the 'forget gate' mechanism. This gate functions as a regulatory component, determining to what extent previous outputs should exert influence on the network's current calculations.

Within the hidden layer of a Long Short-Term Memory (LSTM) network, the fundamental component is known as the memory block. Each memory block is composed of one or several memory cells, which are the core units responsible for preserving information over time. Integral to the functionality of these memory cells are three distinct gates: the forget gate, the input gate, and the output gate. The architecture of a memory cell is shown in Figure 3.1 below,

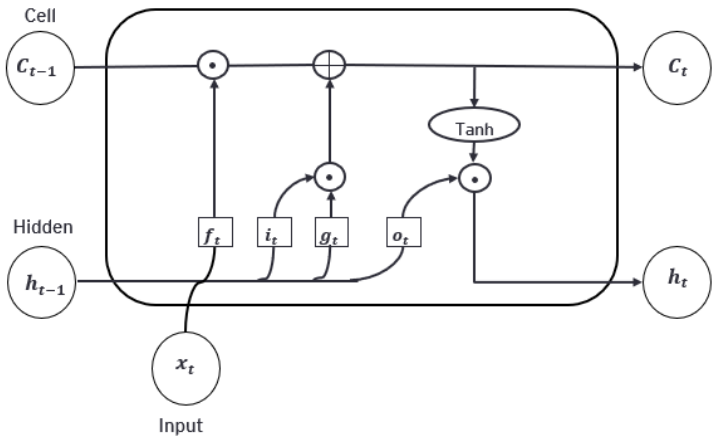

Figure 3.1: Memory cell architecture in LSTM

Let x_t be denoted as the features vector at time t, h_t is the hidden layer vector at time t, and c_t is the cell state. Thus, a standard LSTM neural

network with one hidden layer can be described by the following equations:

$$
\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
g_t &= \tanh(W_g x_t + U_g h_{t-1}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

Where $\odot$ denotes the Hadamard product, g_t is the gate, f_t is the forget gate, i_t is the input gate, o_t is the output gate, σ is the sigmoid function, tanh is the hyperbolic tangent function, $\mathbf{W}_f, \mathbf{W}_i, \mathbf{W}_o, \mathbf{W}_g, \mathbf{U}_f, \mathbf{U}_i, \mathbf{U}_o, \mathbf{U}_g$ are weights vectors, b_f, b_i, b_o and b are denoted as the biases.

By setting the forget gate f_t to 0, the input gate i_t and the output gate o_t to 1, and replacing one of the activations tanh with the identity function, a standard recurrent neural network layer would be realized.

The cell state c_t embodies the long-term memory component within the LSTM network, encapsulating the retained information that the network carries through successive time steps. The evolution of the cell state from one time step to the next is governed by a regulated process through multiplication with the forget date f_t and by adding a term simultaneously as a function of the current input features x_t and the previous hidden layer vector h_{t-1}, multiplied by the activation of the input gate i_t. When the forget gate f_t is close to one, then the previous cell state c_{t-1} will exert a significant influence on the next cell state c_t, meaning the network preserves long-term memory from time step $t-1$ to t. Conversely, if the forget gate f_t is near zero, the network essentially discards long-term memory, meaning the current cell state c_t is influenced predominantly by the current input features x_t and the previous hidden layer output h_{t-1}. This functionality is particularly valuable when processing lengthy sequences, as it allows the network to recognize the start and finish of subsequence patterns within the sequence. Similarly, the input gate regulates the extent to which features x_t and the previous hidden layer output h_{t-1} modify the current cell state c_t, safeguarding the stored information in the cell state c_t from being perturbed by irrelevant new inputs. Lastly, the output gate o_t dictates how much of the current cell state c_t is transferred to the current hidden layer output h_t, acting as a filter that prevents the hidden layer output from being influenced by nonessential information stored in the cell state at time step t. In a word, the 3 gates collectively orchestrate the flow of information into and out of the memory cells, thereby modulating the cell's memory. They are critical in the LSTM's ability to model sequential data, enabling the network to learn and retain relevant information across long intervals while discarding non-essential data.

3.2 Back-propagation and stochastic gradient descent

A loss function quantifies the performance of a neural network based on its training data and expected outcomes. The gradient of the loss function, essential for optimizing the network, is commonly computed by using the backpropagation algorithm. Introduced in the 1970's, the full potential of backpropagation was not recognized until the seminal 1986 paper by Rumelhart et al [17]. This study highlighted several neural network models where backpropagation significantly outperformed previous learning methods, thus enabling the practical application of neural networks to previously intractable problems. Presently, backpropagation is the cornerstone algorithm underpinning the learning process in neural networks.

Let's start with assuming a mean squared error loss function for the LSTM network as below:

$$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,$$

where n is the number of data points, y_i is the actual value of the i-th data point, and $\hat{y}_i$ is the predicted value of the i-th data point by the model.

The goal of the backpropagation algorithm is to compute the partial derivative $\frac{\partial L}{\partial W}$, $\frac{\partial L}{\partial U}$, $\frac{\partial L}{\partial b}$ of the loss function L with respect to any weight W, U or bias b within the network. We know that for the time step t, where $1 \leq t \leq T$:

$$\frac{\partial L}{\partial W_i} = \sum_{t=1}^{T} \frac{\partial L}{\partial i_t} i_t(1 - i_t) x_t$$

$$\frac{\partial L}{\partial W_f} = \sum_{t=1}^{T} \frac{\partial L}{\partial f_t} f_t(1 - f_t) x_t$$

$$\frac{\partial L}{\partial W_o} = \sum_{t=1}^{T} \frac{\partial L}{\partial o_t} o_t(1 - o_t) x_t$$

$$\frac{\partial L}{\partial W_g} = \sum_{t=1}^{T} \frac{\partial L}{\partial g_t} (1 - g_t^2) x_t$$

Similarly, we have:

$$\frac{\partial L}{\partial U_i} = \sum_{t=1}^{T} \frac{\partial L}{\partial i_t} i_t(1 - i_t) h_{t-1}$$
$$\frac{\partial L}{\partial U_f} = \sum_{t=1}^{T} \frac{\partial L}{\partial f_t} f_t(1 - f_t) h_{t-1}$$
$$\frac{\partial L}{\partial U_o} = \sum_{t=1}^{T} \frac{\partial L}{\partial o_t} o_t(1 - o_t) h_{t-1}$$
$$\frac{\partial L}{\partial U_g} = \sum_{t=1}^{T} \frac{\partial L}{\partial g_t} (1 - g_t^2) h_{t-1}$$

and

$$\frac{\partial L}{\partial b_i} = \sum_{t=1}^{T} \frac{\partial L}{\partial i_t} i_t(1 - i_t)$$
$$\frac{\partial L}{\partial b_f} = \sum_{t=1}^{T} \frac{\partial L}{\partial f_t} f_t(1 - f_t)$$
$$\frac{\partial L}{\partial b_o} = \sum_{t=1}^{T} \frac{\partial L}{\partial o_t} o_t(1 - o_t)$$
$$\frac{\partial L}{\partial b_g} = \sum_{t=1}^{T} \frac{\partial L}{\partial g_t} (1 - g_t^2)$$

By applying back propagation through time, we can calculate all the derivatives of the cell and hidden states with following equations iteratively:

$$\frac{\partial L}{\partial i_t} = \frac{\partial L}{\partial c_t} g_t$$
$$\frac{\partial L}{\partial f_t} = \frac{\partial L}{\partial c_t} c_{t-1}$$
$$\frac{\partial L}{\partial o_t} = \frac{\partial L}{\partial h_t} o_t \tanh(c_t)$$
$$\frac{\partial L}{\partial W_y} = \frac{\partial L}{\partial \hat{y}} h_T$$

and

$$\frac{\partial L}{\partial h_{t-1}} = \frac{1}{4} \left[\frac{\partial L}{\partial g_t} (1 - g_t^2) U_g + \frac{\partial L}{\partial i_t} i_t(1 - i_t) U_i + \frac{\partial L}{\partial f_t} (1 - f_t) U_f + \frac{\partial L}{\partial o_t} (1 - o_t) U_o \right]$$

When we have all the derivatives of the loss functions, we can then use stochastic gradient decent (SGD) to find the optimal weights. SGD is an iterative method employed to minimize the loss function L. It uses the gradient of the loss to make small adjustments to the weights, which, over time, lead to a reduction in the loss and an improvement in the model's predictions. The iteration can be expressed in formula as follows:

$$W := W - \eta \frac{1}{N} \sum_{n=1}^{N} \frac{\partial L}{\partial W}$$

Where W are the weights and η is the learning rate. We also provide the pseudocode of iteration process:

Initialize weights W randomly
Choose a learning rate η
Set the number of iterations or a convergence threshold
for each iteration or until convergence **do**
  Randomly shuffle the dataset
  for each batch in the dataset **do**
    Calculate the gradient $\frac{\partial L}{\partial W}$ for the current batch
    Update the weights: $W := W - \eta \frac{\partial L}{\partial W}$
  end for
end for
Return the optimized weights W.

During SGD implementation, the process begins with the random initialization of the network's weights. A subset of the training data is randomly selected. Then the selected data is fed through the network to obtain predictions. The loss function is computed using the predictions and the true values from the training data. Subsequently, the gradient of the loss function with respect to each weight is calculated through backpropagation. The weights are updated in the opposite direction of the gradient by a small step, proportional to a learning rate. This step is intended to reduce the loss. Steps above are repeated with new batches of data until the loss function converges to a minimum or until a predefined number of iterations or epochs is reached, indicating that the model has been sufficiently trained.

In this thesis, we implement the Adam Optimizer for weight training, aligning with the technique of weight updates similar to SGD. The primary distinction lies in the adaptive learning rate used in the Adam Optimizer, which varies throughout the training process. For an in-depth understanding of the iterative weight adjustments facilitated by the Adam Optimizer, can be found at the work by Kingma and Ba (2014) [14].

4 Analysis on Simulated data

4.1 Simulated data

To investigate the performance of Long Short-Term Memory (LSTM) networks in time series forecasting, we initiated our study by simulating a stationary time series dataset. A stationary time series is one whose statistical properties such as mean, variance, and autocorrelation are constant over time, making it a suitable candidate for modeling and forecasting. We employed the `tsa.arma_generate_sample` function from the statsmodels library in Python to simulate a weakly stationary time series with 1000 observations, which is from an $ARMA(3,3)$ model. With weak stationarity it means that the mean and autocovariance of the time series do not change over time, although individual values in the series can fluctuate. The code snipped shows as below:

```
# Set the random seed for reproducibility
np.random.seed(42)

# Define the AR and MA coefficients
ar_coefficients = np.array([1, -0.5, 0.25, -0.1])
ma_coefficients = np.array([1, 0.4, -0.3, 0.2])

# Number of observations
n_samples = 1000

# Generate the time series data
time_series_data = sm.tsa.arma_generate_sample(ar=ar_coefficients, ma=ma_coefficients, nsample=n_samples, scale=0.1)
```

As we can see from the arrays for `ar_coefficients` and `ma_coefficients`, the first element is typically 1 and correspond to $AR(0)$ and $MA(0)$ respectively, which are not part of the AR and MA order. The other three elements in each of the `ar_coefficients` and `ma_coefficients` arrays represent the coefficients for the lagged terms in the autoregressive AR and moving average MA parts of the ARMA model, respectively. Each coefficient corresponds to the influence of a past value or past shock (error term) on the current value of the time series.

The simulated data were not constrained to a specific range; thus, to prepare the data for the LSTM model, which is sensitive to the scale of the input data, we performed a min-max scaling to transform the time series values to fall within the $(0, 1)$ interval. The formula for Min-Max normalization of a value x is:

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

This normalization process is crucial for the efficient training of neural networks and helps in accelerating the convergence of the optimization algorithms used during the learning process.

We adopted a common approach to split our time series data, allocating 70% of the observations to the training set and the remaining 30% to

the test set. This split was chosen to ensure that the models have a sufficiently large amount of historical data to learn from, while still retaining a substantial portion of the data for an unbiased evaluation of the models' predictive capabilities. The training set consists of the first 70% of the time series data, which is used to fit the models. The LSTM network learns to recognize patterns and dependencies in the data during this phase, adjusting its weights through backpropagation and gradient descent optimization techniques. Similarly, the ARMA model parameters are estimated using the same subset of data to capture the underlying process generating the observations. The test set comprises the latter 30% of the time series data and serves as a new dataset for the models to forecast. This set is not used during the training phase and is strictly reserved for evaluating the models' performance. By comparing the models' forecasts against the actual values in the test set, we can assess their out-of-sample predictive accuracy.

4.2 ARMA model

The ARMA model is a cornerstone in time series analysis, and its ability to capture the dynamics of stationary time series makes it an ideal reference model for our study. As a benchmark for our LSTM model, we utilized an ARMA model fitted to the simulated data.

The ARMA model combines two fundamental components: autoregressive AR and moving average MA. The ARMA model is denoted as $ARMA(p, q)$, where p represents the order of the autoregressive part and q the order of the moving average part. The autoregressive component AR of the ARMA model captures the relationship between a time series observation and a specified number of lagged observations. The moving average component MA models the relationship between the time series observation and a linear combination of the error terms from previous time steps. Formally, $ARMA(p, q)$ process can be written as:

$$X_t = \varphi_1 X_{t-1} + \cdots + \varphi_p X_{t-p} + Z_t + \theta_1 Z_{t-1} + \cdots + \theta_q Z_{t-q}$$

Where X_t is the time series at time t, $\varphi_1, \ldots, \varphi_p$ are the coefficients of the AR part, and $\theta_1, \ldots, \theta_q$ are the coefficients for the MA part. Z_t represents the white noise error terms and $\{Z_t\} \sim \mathrm{WN}(0, \sigma^2)$.

Ensuring stationarity is important in ARMA modelling because the presence of a unit root can lead to unreliable and spurious results in time series analysis. To confirm the stationarity of the simulated time series, we conducted the Augmented Dickey-Fuller (ADF) test [5]. The ADF test is a formal statistical test for stationarity based on estimating the following regression model:

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \delta_1 \Delta y_{t-1} + \cdots + \delta_p \Delta y_{t-p} + \varepsilon_t$$

Where Δ is the difference operator, y_t is the time series at time t, α is a constant term, βt represents a deterministic time trend, γ is the coefficient on the lagged level of the time series, $\delta_1, \ldots, \delta_p$ are coefficients on the lagged differences of the time series, ε_t is the error term. The ADF test examines the null hypothesis that a unit root is present in the time series sample. A rejection of the null hypothesis implies that the time series is stationary. The results of the ADF test confirmed that our simulated time series and train dataset and test dataset are stationary.

A critical step in the application of the ARMA model to time series forecasting is the identification of the optimal order of the autoregressive AR and moving average MA components, denoted by p and q, respectively. The selection of these parameters is pivotal as it determines the model's ability to capture the underlying dynamics of the time series data. We employed an iterative approach, systematically varying p and q within the range of 1 to 6. This range was chosen based on the assumption that the underlying process would not require a higher order to capture the essential dynamics, while also keeping the model complexity manageable. For each combination of p and q, we fitted an ARMA(p, q) model to the training data and evaluated its performance using the Akaike Information Criterion (AIC). The AIC is a widely used metric for model selection that balances the model's goodness of fit with its complexity. A lower AIC value suggests a better model, as it indicates a more parsimonious fit to the data. Through this iterative process, we identified that the ARMA model with $p = 3$ and $q = 1$ yielded the lowest AIC value among the considered range of model orders. This result suggests that an ARMA(3,1) model provides the best balance between model complexity and fit to the historical data within the specified search space. The ARMA(3,1) model implies that the current value of the time series is influenced by the three immediately preceding values AR component and the most recent prediction error MA component. The selection of this model is consistent with the observed autocorrelation and partial autocorrelation functions of the time series, which indicated a significant correlation at lag 3 and a significant spike at lag 1, respectively.

In practice, we use the `tsa.ARIMA` function from `statsmodels` in Python to fit our ARMA model on the training dataset in order to estimate the coefficients of the AR and MA parts. The coefficients of the fitted model shows in tabel below:

	coef
const	0.4837
ar.L1	-0.0722
ar.L2	-0.0616
ar.L3	0.0827
ma.L1	0.9569
sigma2	0.0152

As we can see, σ^2, the variance of the optimal prediction error $Z(t)$ is 0.0152, which is smaller than the variance of the stationary time series $X(t)$ 0.02754. This is a good indicator that the ARMA model effectively captured the underlying structure of the time series. The fitting process is about to find a set of parameters including coefficients of the AR and MA parts and the variance of the errors that can maximize the log-likelihood function. When fitting an ARMA model to time series data, the goal is to capture the underlying autocorrelation structure of the time series so that the residuals are as close to white noise (i.e., they are independently and identically distributed with a mean of zero and constant variance) as possible. In order to check the residuals of the fitted model to ensure that they resemble white noise, we performed model diagnostics by checking the Q-Q plot for residuals, calculating the autocorrelation function (ACF), checking the Ljung-Box test result, and the Jarque-Bera test result. After this, we use our fitted model to predict one-step ahead on the test dataset.

The ARMA model serves as a reference point, allowing us to compare the forecasting performance of the LSTM model against a well-established traditional method in time series analysis.

4.3 LSTM model

Next, we proceed to develop a LSTM model. In the context of time series forecasting using LSTM networks, we can conceptualize the model within a stochastic framework. A one time-step LSTM model can be described by the equation:

$$X_t = LSTM(input_{t-1}) + \epsilon_t$$

where X_t represents the actual value at time t, and $input_{t-1}$ is the input to the LSTM at time $t-1$, which could be the observed value X_{t-1} at time $t-1$ for a univariate model, or a vector that includes X_{t-1} and other relevant features at time $t-1$ for a multivariate model. The dimension of X_t is determined by the nature of the time series data and the problem we are solving. In this study for the simulated data, X_t is a real number since we want to predict the value in the time series at time t. The term ϵ_t represents the error or noise at time t, which is assumed to be independently and identically distributed with a mean of zero and some variance σ^2. This error term captures the random fluctuations that the LSTM model does not account for. Unlike the linear relationship in an ARMA model, the LSTM's function is non-linear and has the ability to capture complex dependencies in the data. The LSTM achieves this through its internal mechanisms, such as gates and cell states, which allow it to learn from long sequences of data and remember important information while forgetting the irrelevant.

For this time series, we proceed to develop a 3 timesteps LSTM model. With a length of 3 timesteps, meaning it considers the past 3 payments to

forecast the next one. In this study, we employ the Keras API provided by TensorFlow to construct and train our LSTM models. The Python code snippet is shown as below:

```python
# Define LSTM model
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))  # Output layer with one neuron for regression

# Compile the model
model.compile(optimizer='Adam', loss='mse')

# Train the model
history = model.fit(train_X, train_y, epochs=10, batch_size=32, validation_data=(test_X, test_y))
```

The architecture of our LSTM network includes a layer with 50 neurons. Neurons, or units, are the fundamental processing elements of neural networks. In a LSTM layer, each neuron is capable of learning from the temporal dependencies of the input sequence, making it particularly suitable for time-series data. The choice of 50 neurons is an initial choice with consideration of balance between model complexity and computational efficiency, providing the network with sufficient capacity to capture the underlying patterns in the data without becoming overly complex. Following the LSTM layer, we have a dense layer with a single neuron. A dense layer is a fully connected neural network layer where each input is connected to each output by a learned weight. In our case, the dense layer serves to consolidate the information learned by the LSTM layer and produce the final output. The LSTM model architecture is illustrated in Figure 4.1.

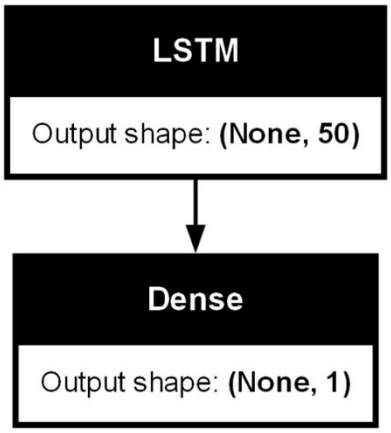

Figure 4.1: LSTM model architecture

The model utilizes the mean squared error (MSE) as its loss function. MSE computes the average of the squares of the differences between the predicted values and the actual values, emphasizing larger errors more significantly than smaller ones. This characteristic of MSE is particularly useful

when large errors are deemed more detrimental than smaller ones, as it penalizes the model more heavily for larger deviations from the actual values. By minimizing the MSE during training, we aim to enhance the model's predictive accuracy and ensure that it is not disproportionately influenced by outliers or large errors.

We use the Adam optimizer to update the model's weights during training. Adam is an optimization algorithm that adjusts the learning rate dynamically for each weight in the model, combining the advantages of two other extensions of stochastic gradient descent (SGD): AdaGrad and RMSProp. This adaptive learning rate helps the model to converge more quickly and efficiently to the optimal solution. The model is trained over 10 epochs, where an epoch represents one complete pass through the entire training dataset. Training for multiple epochs allows the model to iteratively learn and refine its predictions. A batch size of 32 is used, meaning that the model weights are updated after every 32 samples are processed. The choice of batch size can affect the speed and stability of the learning process. At this juncture, we have established our preliminary hyperparameter settings drawing upon empirical insights. These initial choices are informed by prior experimentation and serve as a starting point for the model's configuration. In other words, we have not engaged in fine-tuning the hyperparameters to enhance the performance of this model.

4.4 Results

Upon completing the training phase for both the ARMA and LSTM models using the designated training dataset, we proceeded to evaluate their forecasting performance on the test dataset. This evaluation phase is crucial as it provides insights into how well each model generalizes to unseen data, which is a key indicator of their practical utility in time series forecasting.

To facilitate a direct comparison between the predictive capabilities of the ARMA and LSTM models, we generate forecasts for the test dataset using both models. The predicted values obtained from each model are then plotted alongside the actual observed values from the test dataset. The resulting figure 4.2 offers a visual representation of the models' accuracy, allowing us to assess the alignment between the predicted and actual values.

Based on the visual evidence presented in the graph above, it is apparent that both the ARMA and LSTM models have successfully identified the underlying pattern of the time series data. However, closer inspection reveals that the ARMA model more closely mirrors the trajectory of the actual values, suggesting a higher degree of precision in its forecasts. The graph indicates that the ARMA model's predictions are more consistently aligned with the actual data points throughout the test period, which may point to its superior ability to model the linear aspects of the time series. This

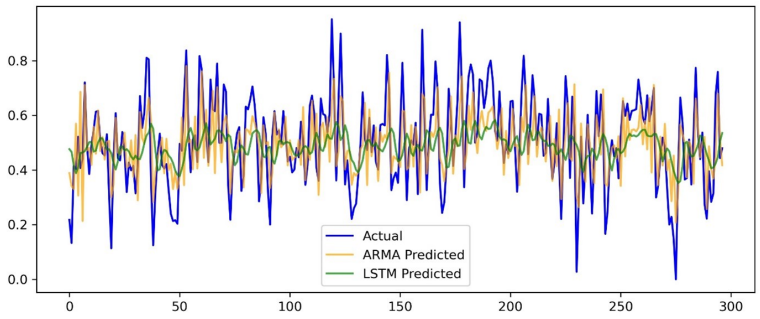

Figure 4.2: Actual vs Predicted ARMA and LSTM models for Simulated data

observation is particularly noteworthy given the simplicity of the ARMA model compared to the more complex LSTM network. While the LSTM model also demonstrates a commendable performance in tracking the general movements of the time series, the ARMA model's tighter adherence to the actual curve implies that it may be more adept at capturing the specific nuances of this dataset. The insights gained from the comparative analysis of the ARMA and LSTM models are further substantiated by examining the residual plots played in the subsequent Figure 4.3:

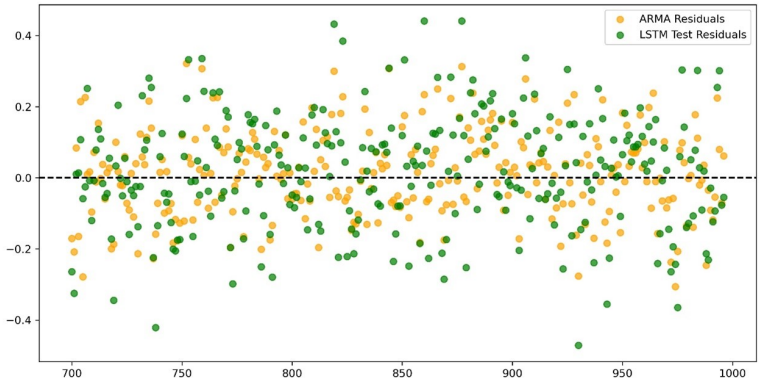

Figure 4.3: Residuals of ARMA and LSTM models for Simulated data

The graphical analysis provides a clear initial indication that the ARMA model has delivered a superior level of accuracy in its predictions for this particular time series. It is important to complement this qualitative assessment with quantitative metrics to confirm the visual interpretation. Metrics such as mean absolute error (MAE) or the root mean squared error (RMSE)

can provide a more definitive evaluation of each model's accuracy.

$$\begin{aligned}
\text{MAE for ARMA} &= 0.09766 \\
\text{MAE for LSTM} &= 0.12773 \\
\text{RMSE for ARMA} &= 0.12209 \\
\text{RMSE for LSTM} &= 0.15998 \\
\text{Standard deviation for X(t)} &= 0.16484 \\
\text{Standard deviation for Z(t)} &= 0.12304
\end{aligned}$$

By looking at MAE and RMSE, we observe that the ARMA model exhibits lower values for both MAE and RMSE. The lower MAE value for the ARMA model suggests that, on average, its predictions are closer to the actual observations. Furthermore, the lower RMSE indicates that the ARMA model is less prone to large errors, which are more heavily penalized in the RMSE calculation. Given that the standard deviation of the stationary time series $X(t)$ is 0.16484 and the standard deviation of the optimal prediction error $Z(t)$ is 0.12304, the superior performance of the ARMA model on both MAE and RMSE metrics implies that it is more adept at capturing the underlying patterns in the time series data. Consequently, for this specific dataset, the ARMA model is the more suitable choice for forecasting, providing a closer fit to the observed data compared to the LSTM model.

It is worth to point out that the standard deviation of the optimal prediction error $Z(t)$ represents the theoretical minimum level of prediction error (RMSE) that the model could achieve if the true parameters were known and the model was correctly specified. If the model were perfect and the parameters were exactly estimated, the RMSE would be equal to the standard deviation of $Z(t)$, because there would be no systematic errors in the predictions, only random disturbances. However, here we observe that the standard deviation of $Z(t)$ is slightly higher than the RMSE value for ARMA. This is because , in practice, we rarely know the true parameters of a model. Instead, we estimate the parameters from a finite sample of data, which introduces estimation uncertainty. The size of the sample and other aspects of how the model is estimated (such as the choice of starting values and optimization method) can also affect the quality of the estimation and, consequently, the RMSE. Therefore, it is possible to obtain an estimated RMSE that is smaller than the standard deviation of $Z(t)$ in a specific sample. In our case, we have 300 observations in our test set, which is a relatively small sample, and it's possible that the particular sample we have just happens to fit our estimated model unusually well, giving us a RMSE that's better than expected.

While the ARMA model has demonstrated superior performance on our

simulated dataset, as evidenced by its lower MAE and RMSE values, real-world time series data often exhibit more complexity, including non-linear relationships among variables and features. These complexities can challenge traditional linear models, potentially limiting their effectiveness in capturing the full spectrum of dynamics present in the data. In contrast, LSTM (Long Short-Term Memory) networks, with their sophisticated architecture designed to learn from sequences, are well-suited to model such non-linear dependencies and long-range temporal interactions. This capability positions LSTM models as a promising alternative for handling the intricacies of real-world time series data. In our next chapter we will delve into the application of LSTM networks to more complex datasets in the real world, and we will explore if LSTM models can provide reliable predictions when faced with the multifaceted nature of actual time series data.

5 Analysis on Real world data

5.1 Exploratory data analysis

The real data we use in this study is from Swedbank Försäkring AB. It is a Unit-linked Child endowment insurance product. Unit-linked insurance as a financial product combines insurance coverage and investment exposure in equities or bonds. The purpose of this child endowment product is for parents or grandparents to save for the future of their children or grandchildren. Policyholders pay an initial lump-sum payment when they start the insurance, followed by monthly premium payments.

The raw dataset received for this research comprises approximately 8,5 millions premium payment transactions associated with around 120,000 policies. The transactions span from 201201 to 202404. Upon analysis of the raw data, it was observed that over half of the policies exhibited a stationary premium payment pattern, characterized by consistent premium amounts throughout the policy duration. For the objectives of this study, the focus was narrowed to policies with variable premium payments. Consequently, the analysis was conducted on approximately 43,000 policies, accounting for about 3,6 millions transactions, to explore the dynamics of fluctuating premium payment behaviors. The table in Figure 5.1 illustrates the distribution of premium payment transactions across different years, revealing a relatively even spread with a discernible upward trend. This indicates an increasing volume of data in the later years. The dataset includes all policies with premium payments active as of the beginning of 2012, as well as any new policies issued throughout the duration of the study period.

Counts for Product Code H:

	Transaction year	counts
0	2012	215066
1	2013	237650
2	2014	258892
3	2015	282642
4	2016	299648
5	2017	316870
6	2018	325855
7	2019	334099
8	2020	335891
9	2021	337459
10	2022	326410
11	2023	314267
12	2024	78209

Total count for Product Code H: 3662958

Figure 5.1: The distribution of premium payments per year

The study incorporates data from both internal and external sources. The internal data comprises anonymized snapshots of individual policy trans-

actions, capturing a range of covariates that include premium payment transactions, product information, policy characteristics, and policyholder demographics. To ensure the company's confidential information and proprietary secrets are protected, the amounts for certain covariates have been rescaled. External data is derived from macroeconomic indicators, collected in monthly snapshots. The dataset comprises a mix of categorical and numerical variables. Due to the sensitive nature of the data, specific details regarding the dataset are confidential and cannot be disclosed publicly.

To prepare the dataset for modeling and elevate the data's quality, thorough data cleaning and feature engineering are undertaken. The data cleaning process encompassed tasks like handling missing data, removing duplicates, converting data types, correcting data entry errors, among others. For instance, to address the issue of missing data, various techniques are employed, such as imputing the missing values by assigning them a reasonable estimate based on other available data or treating the missing values as a separate category. The choice of technique depends on the nature of the data and the specific circumstances surrounding the missing values.

Throughout the feature engineering phase, we have crafted new features from the existing data, such as deriving age by calculating the difference between the birth date and transaction date. Given that our chosen model, Long Short-Term Memory (LSTM), necessitates numerical input, we have applied encoding techniques to transform categorical variables into a numerical format. One method we have utilized is one-hot encoding, which constructs a binary vector for each category, marked by a '1' for the active category and '0' for all others. This approach is particularly effective for nominal data as it avoids imposing an artificial order among categories. However, its suitability diminishes with variables that have a large number of categories, as it can result in a significant increase in the dimensionality of the dataset. For ordinal categorical variables, where the order of the categories carries meaning, we have opted for label encoding, assigning a unique integer to each category in accordance with its order. For further reading on one-hot encoding and label encoding, we refer to the books by James G. et al. [8] and Raschka, S. et al. [9]. In addition to encoding, we have standardized the features by using normalization to ensure a consistent scale across all inputs, a crucial step for models sensitive to input magnitude. Normalization adjusts the scale of the data without distorting differences in the ranges of values or losing information. It brings all the numeric features into a common scale, allowing the model to converge more quickly during training and reducing the risk of getting stuck in local optima. In this study, we use min-max scaling, which linearly transforms the features so that they fall within a given range, typically [0,1].

By applying normalization, we ensure that each feature contributes approximately proportionately to the final prediction. This is particularly important when combining features that are on different scales and have

different units of measurement.

The primary goal of this study is to forecast premium payments. Therefore, our initial step involves examining the characteristics and distribution of premium payment data within the dataset. The graph in Figure 5.2 illustrates that within the dataset, an overwhelming majority of policies (99%) register more than four payment instances. On average, policies paid approximately 90 times. Notably, the most common number of payment occurrences stands at 147, corresponding to roughly 3,800 policies.

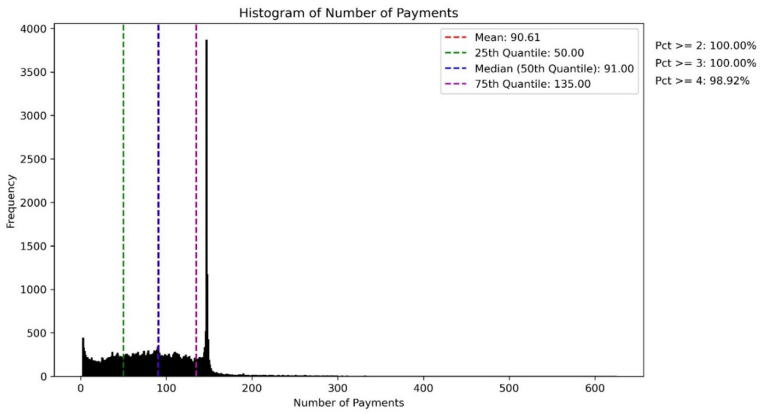

Figure 5.2: Histogram of Number of Payments

Additionally, upon examining the contracts with a payment count of 147, we observe sporadic fluctuations over time, with no discernible trend emerging from the data (refer to Figure 5.3).

To further explore and gain insights into the relationship between covariates and premium payments, we have conducted an analysis with the help of domain knowledge. Below are some examples of our findings.

The Figure 5.4 reveals the distribution of policyholders' ages at the time of insurance purchase, which spans from 0 to 95 years. This wide range is attributable to the fact that policyholders can include parents, grandparents, or even the children themselves. It is readily apparent that the majority of policyholders are parents. There is a noticeable trend where the average premium paid increases within the age brackets of 20 to 40 and 55 to 90, while it decreases between the ages of 40 to 55. This pattern may suggest that as the time approaches for children to utilize the savings, policyholders are inclined to pay higher premiums. Additionally, the data indicates that grandparents, who are typically of retirement age, tend to pay higher average premiums compared to parents, as evidenced by the elevated average premium levels for individuals aged 65 and above.

An additional noteworthy finding is the correlation between the duration of the policy and the amount of the premium payments, refer to Figure 5.5. There appears to be a positive association, where longer-standing policies are

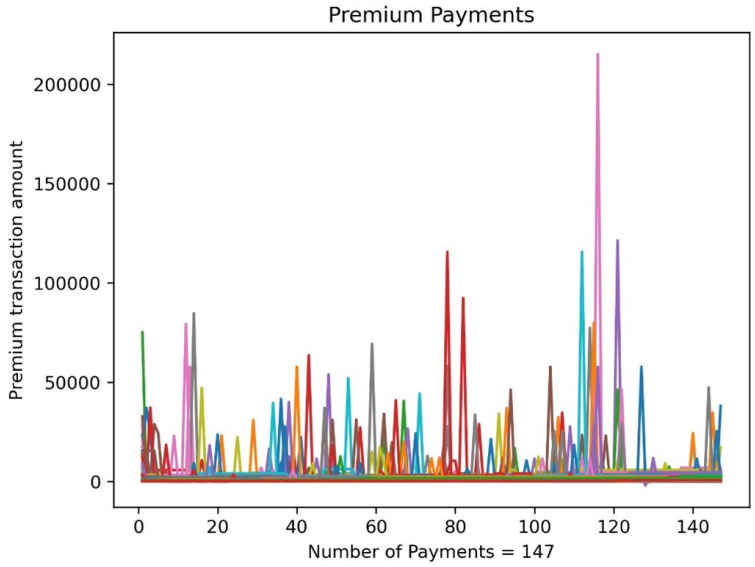

Figure 5.3: Premium Payments for Number of Payments=147

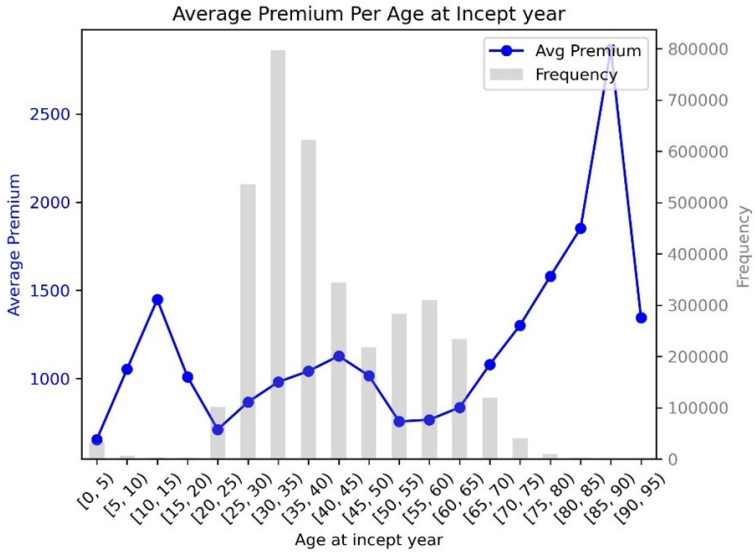

Figure 5.4: Average Premium Per Age at Incept year

linked with higher premium payments. A particularly high average premium payment is observed for policies in their first year. This could be rationalized by the tendency of customers to make substantial lump-sum payments at the inception of their contracts.

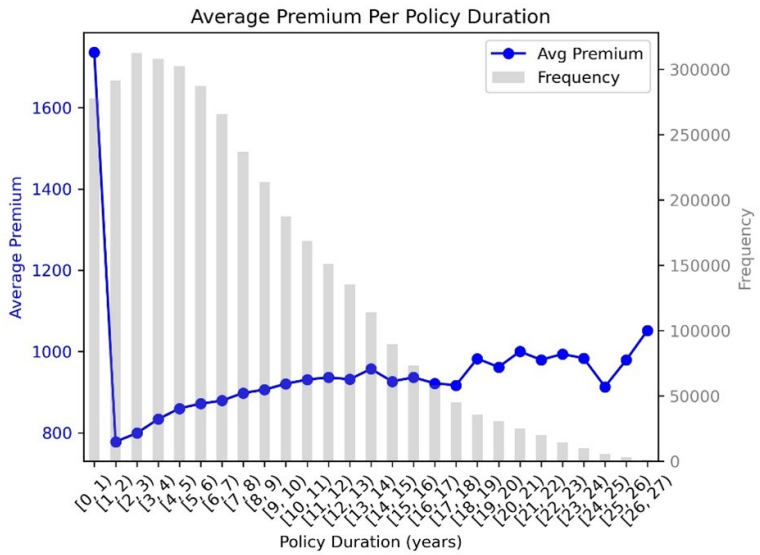

Figure 5.5: Average Premium Per Policy Duration

Finally, an intriguing trend emerges from the data: while female customers are more likely to purchase this insurance product, their male counterparts tend to pay higher premiums, refer to Figure 5.6.

5.2 LSTM model development and validation

To prepare the dataset for LSTM modeling, it is necessary to transform the data into a format suitable for sequential learning. Given the variable length of observation sequences for each policyholder, we must standardize the input sequence length for the LSTM model. To achieve this, we define a fixed timestep, which represents the number of past observations the model should consider when making predictions. For instance, if we set the timestep to 3, the model will utilize the past three payment instances to inform its predictions. Consider a policyholder with a payment sequence represented as {t1, t2, t3, t4, t5}. Under our framework, the LSTM model will interpret {t1, t2, t3}, {t2, t3, t4}, and {t3, t4, t5} as distinct input sequences. This approach enables the model to make predictions based on a series of events over time, rather than relying on a single payment instance. By incorporating temporal dependencies between events, the LSTM can capture patterns and trends that are essential for accurate forecasting in the context of policyholder behavior.

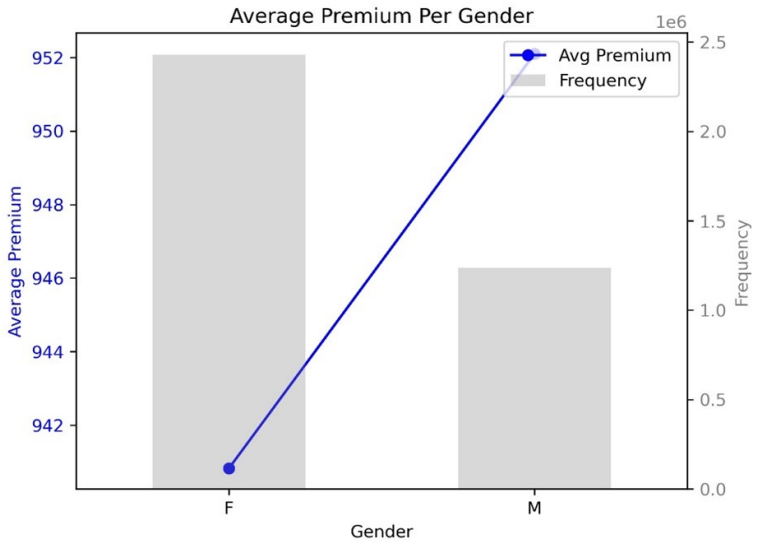

Figure 5.6: Average Premium Per Gender

To ensure the robustness and generalizability of our LSTM model, we partitioned our dataset into three distinct subsets: training, validation, and testing. These subsets represent 60%, 20%, and 20% of the policyholder data, respectively. The training dataset is the primary resource for the model's learning process. It is used to adjust the model's weights and biases to minimize the prediction error. The model iteratively learns from this subset, capturing the underlying patterns and temporal dependencies present in the policyholder payment sequences. The validation dataset serves a dual purpose. Firstly, it acts as a checkpoint during the training phase, allowing for the monitoring of the model's performance on data that it has not been trained on. This helps in detecting issues such as overfitting, where the model performs well on the training data but fails to generalize to new data. Secondly, the validation set is instrumental in fine-tuning the model's hyperparameters. By evaluating various hyperparameter configurations against the validation set, we can select the combination that yields the best performance, thereby optimizing the model's predictive capabilities. The test dataset is the final arbiter of the model's performance. It is used exclusively to assess the model's predictive accuracy after the training and validation phases are complete. The test set provides an unbiased evaluation of the model, as it consists of policyholder data that has remained unseen by the model throughout the training and validation processes. This ensures that the performance metrics obtained from the test set reflect the model's true generalization ability to new, real-world data. It is imperative that each policyholder's data is assigned to only one of these subsets to maintain the integrity of the model's evaluation. This exclusivity prevents data leakage

and ensures that the model's performance metrics are not artificially inflated by having overlapping data between the training, validation, and test datasets. The careful segregation of data into these partitions is a cornerstone of our methodology, enabling us to develop a reliable and effective LSTM model for predicting policyholder behavior.

We start with the construction of a univariate LSTM model, which means we focus on a single feature, the "premium payment amount," to predict future values based on historical data. The model is designed to process input sequences with a length of 24 timesteps, meaning it considers the past 24 payments to forecast the next one. In the construction of our LSTM univariate model, we employed the same foundational techniques as those used in the simulation phase. However, when applying these models to real-world data, we adapted our approach by varying the selection of hyperparameters. This adjustment was crucial to account for the complexities and unique characteristics of the real dataset, which differed from the controlled conditions of the simulated environment.

As default, we choose no shuffling. Shuffling is a technique used during training to randomize the order of the training data before each epoch. It helps prevent the model from learning spurious patterns that may arise from the order of the data rather than the underlying data distribution. However, for time-series data where the sequence order is significant, we opt not to shuffle the data to preserve the temporal relationships within the sequences.

The figure presented Figure 5.7 illustrates the progression of training loss versus validation loss over the course of the model's training epochs. Observing the training loss curve, we notice a trend where the model exhibits continuous learning up to the 20th epoch, as indicated by a consistent decrease in loss. Beyond this point, the curve begins to plateau, suggesting that the model is reaching a point of convergence where additional epochs do not yield substantial improvements in learning on the training dataset. In contrast, the validation loss curve displays fluctuations around the training loss curve. This behavior is indicative of the model's performance on the validation dataset, which consists of data not seen during the training phase. The oscillations in validation loss suggest that the model's ability to generalize to new data is not improving consistently with each epoch and may be subject to variability. The flattening of the training loss curve combined with fluctuations in the validation loss could be a sign of overfitting. This occurs when the model learns the training data too well, including noise and idiosyncrasies, to the detriment of its performance on unseen data.

Building upon the foundation laid by the univariate LSTM model, our subsequent endeavor involves the development of a multivariate LSTM model. This advanced model integrates a total of 39 distinct features, encapsulating a richer and more complex representation of the underlying data. The model is structured to process sequences with a timestep of 3, thereby considering a trio of consecutive data points to predict subsequent outcomes.

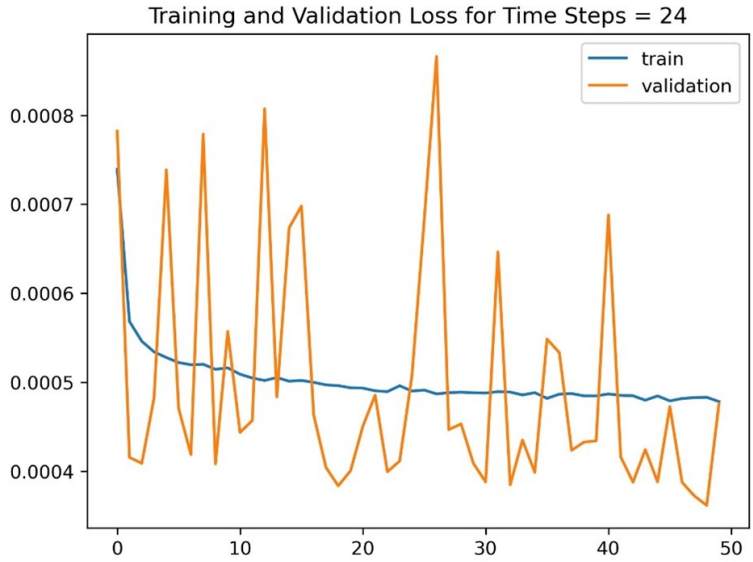

Figure 5.7: Univariate LSTM Model Train and Validation loss curves

In the construction of the multivariate LSTM model, we have elected to retain the hyperparameters that were previously established in the univariate model. This decision allows for a direct comparison between the two models, isolating the impact of incorporating multiple features on the model's performance.

In contrast to the univariate model, the loss curve of the multivariate LSTM model in Figure 5.8 exhibits markedly less fluctuation, suggesting a more stable learning process. Notably, the validation loss consistently remains below the training loss for the majority of the training epochs. This pattern is indicative of a model that generalizes well to unseen data. The reduced fluctuation in the loss curve of the multivariate model may be attributed to the richer feature set, which provides the model with a more comprehensive understanding of the underlying data patterns. With a greater number of features, the model has the potential to capture more complex relationships and dependencies, which can lead to improved performance and stability during training. The consistent positioning of the validation loss beneath the training loss is a positive sign of the model's generalization ability. It suggests that the model is not merely memorizing the training data but is effectively learning the salient characteristics that are applicable to the validation data as well. This behavior is desirable in a predictive model, as it implies that the model is likely to perform well on real-world data that it has not encountered during the training phase. Overall, the performance of the multivariate LSTM model, as reflected by the loss curves, demonstrates the advantages of incorporating multiple features

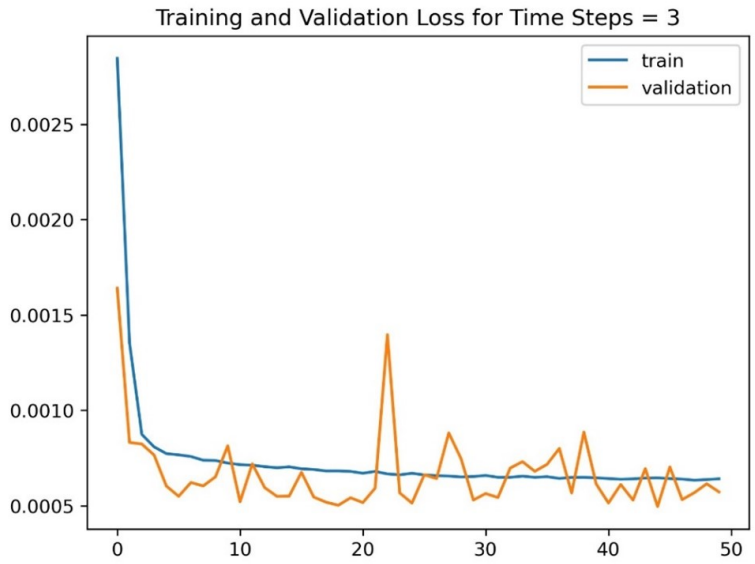

Figure 5.8: Multivariate LSTM Model Train and Validation loss curves

into the model. It underscores the potential for achieving a robust model that not only learns effectively from the training data but also possesses strong predictive capabilities when applied to new datasets.

To fit the multivariate LSTM model and prevent overfitting, early stopping as a regularization technique is adopted in training process. By early stopping, it means monitoring the model's performance on a validation dataset during the training process and if the model's performance on the validation set ceases to improve or starts to degrade for a specified number of epochs, then the training is halted. By implementing early stopping, we can ensure that the LSTM model retains its generalization ability and does not learn the noise or random fluctuations present in the training data. This technique not only helps in obtaining a more robust model but also can save computational resources by reducing unnecessary training time.

To gauge the efficacy of the LSTM model relative to conventional approaches, we established a baseline model employing a simple moving average (SMA) with a sliding window of size 3. This baseline serves as a benchmark for comparison, providing a straightforward method to generate predictions based on the average of the most recent three observations in the test dataset. We then computed the Root Mean Square Error (RMSE) for the baseline model, which quantifies the average magnitude of the prediction errors. The RMSE is a widely-used measure of accuracy that penalizes larger errors more severely, making it a robust indicator of model performance. By comparing the RMSE obtained from the baseline model against the RMSE derived from the multivariate LSTM model on the same test dataset, we

can assess the degree to which the LSTM model surpasses the traditional method. A lower RMSE for the LSTM model would indicate superior predictive accuracy, highlighting the benefits of leveraging more sophisticated machine learning techniques for time-series forecasting. The RMSE for the multivariate LSTM model registers at 2444.36198, which is notably lower than the RMSE of 2761.263 observed for the baseline SMA model. This differential in RMSE values is indicative of the multivariate LSTM model's enhanced performance in forecasting. The reduction in RMSE suggests that the LSTM model is more adept at capturing the intricacies and temporal dependencies present within the dataset, leading to more precise predictions when compared to the baseline model. The baseline model, while useful as a point of reference, relies on a simpler heuristic that does not account for the potential complexities and non-linear relationships between features. In contrast, the LSTM model's ability to process multiple input features and learn from sequences of data allows it to provide a more nuanced understanding of the underlying patterns, resulting in a more accurate forecast.

To effectively illustrate the comparative performance of the LSTM model and the baseline MA model, we can create a visualization of the residuals, which are the differences between the actual values and the predicted values from each model. Residual analysis is a powerful diagnostic tool that can reveal patterns in the model's predictions and highlight areas where the model may be systematically underperforming or overperforming.

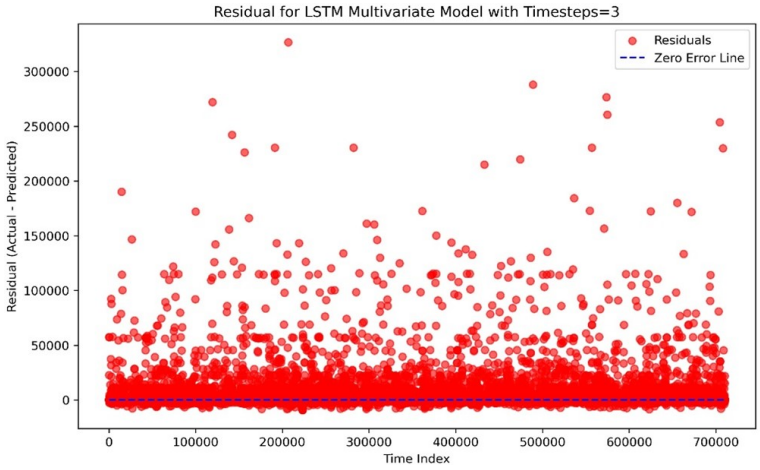

Figure 5.9: Residual for LSTM Multivariate Model with 39 features

The graphical representations in Figure 5.9, 5.10 and 5.11 provided above reveal that the MA model appears to overestimate the target variable when the actual values are on the lower end of the spectrum. This pattern of overestimation is evidenced by a clustering of negative residuals for smaller actual values. This can be explained by an example sequence below, where

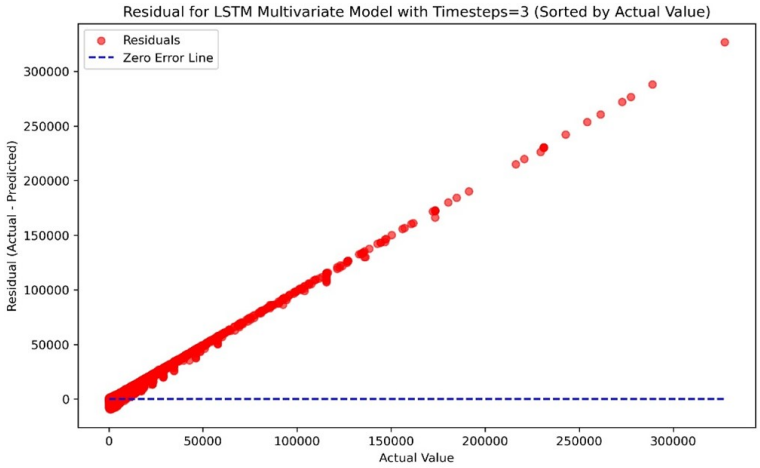

Figure 5.10: Residual for LSTM Multivariate Model with 39 features (sorted by Actual value)

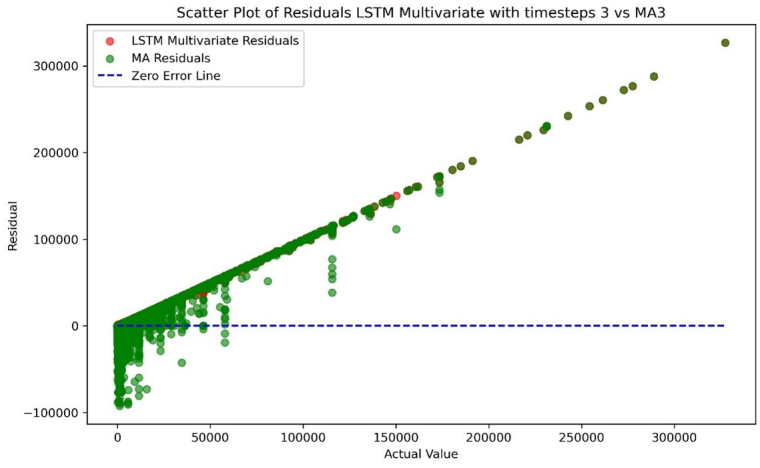

Figure 5.11: Residuals for LSTM Multivariate Model with 39 features vs MA model)

the MA model yields a prediction of 77,356. This inflated forecast can be attributed to the influence of an anomalously high initial payment amount of 231,145, which skews the moving average upwards. In stark contrast, the LSTM model offers a prediction of 445, aligning more closely with what would be considered a reasonable estimate given the context of the data.

```
    var1(t-3)  var1(t-2)  var1(t-1)    var1(t)  Predicted_LSTM3  \
231145.469958  462.29094  462.29094  462.29094       445.861729

Predicted_MA3
 77356.683946
```

However, the residual plot for the LSTM model indicates a trend where the model exhibits diminished predictive accuracy when tasked with forecasting higher values of premium payment amounts. A clear correlation emerges, indicating that as the actual premium payment amounts increase, the residuals also tend to rise. This observation suggests that the LSTM model's performance is not uniform across the range of payment amounts. While the model may predict lower values with relative precision, its ability to accurately forecast higher payment amounts is less reliable. The increasing residuals associated with higher actual values point to a potential systematic bias or limitation within the model when dealing with larger magnitudes. It highlights the need for further investigation into the model's structure and training process to identify the factors contributing to this predictive shortfall.

5.3 Feature importance

In the context of time series forecasting, understanding the relative importance of different features can provide valuable insights into the underlying factors that drive the predictions. To this end, we employ a regression tree as a post-hoc analysis tool to calculate and interpret the feature importance derived from the predictions of a more complex model, our LSTM networks. After training the multivariate LSTM model with 39 features on the training data and obtaining its predictions, we use these predictions as the target variable for a regression tree. The input features to the regression tree are the same as those used by the LSTM multivariate model with 39 features. We use DecisionTreeRegressor from SKLearn.tree in Python for fit the regression tree.

The structure of a regression tree can be illustrated in Figure 5.12. The tree is constructed from a root node and grows by splitting the data into branches based on the input features. As shown in Figure 5.12, the entire dataset starts at the root node where we have the whole train dataset with 2,119,835 observations. The tree then splits the data first based on the value of "Feature 1". The split points are chosen by examining each feature and determining the best split that minimizes the variance within the resulting

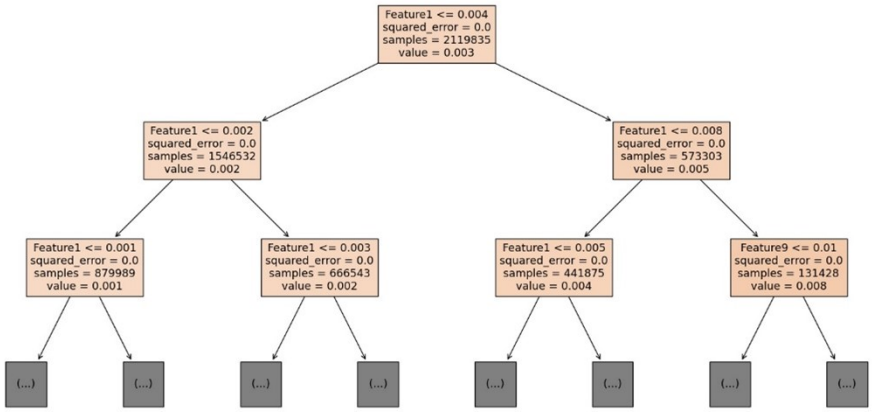

Figure 5.12: Regression tree structure illustration

child nodes. The algorithm considers all possible split points for each feature and calculates a metric, such as the mean squared error (MSE), for each potential split. The split that results in the largest reduction in MSE is selected. This process is repeated recursively for each child node until a stopping criterion is met, such as reaching a maximum tree depth. Once the tree has been grown and no further splits are made, each leaf represents a partition of the data with similar target values. The value given to a leaf is typically the mean of the target variable for all the samples within that leaf. In our case, the value in the left leaf at the second depth is 0.002. When a new sample is fed into the tree for prediction, it traverses the tree based on its feature values until it reaches a leaf. The predicted value for that sample is the mean value of the leaf.

In order to find the optimal maximum depth for our regression tree model, we use grid search with cross-validation. The grid search with cross-validation is a systematic approach to hyperparameter tuning that seeks to find the best combination of parameters for a given model. It does this by training and evaluating a model for each combination of hyperparameters specified in a predefined "grid". Cross-validation involves partitioning the data into subsets, training the model on some subsets while validating on others, and then averaging the results to estimate the model's predictive performance. For further details regarding grid search and cross-validation, we refer to James et al. [13]. In our case, we have defined the grid parameter to a range of (1,20) with 3-fold cross-validation. After evaluating all the

combinations, the best max-depth turns out to be 11, which is applied later in the regression tree model.

After training the regression tree model, we can determine the importance score for each feature based on how much they contribute to the model's predictions. In regression tree, feature importance is calculated based on the improvement in the splitting criterion, such as MSE that each feature provides when it is used to split the data. The importance of a feature can be expressed as the sum of the reduction in error brought by that feature across all the nodes where it is used to create a split. The reduction in error for a single tree node split can be calculated as follows:

Let $\text{MSE}_{\text{parent}}$ be the MSE of the parent node before the split, and MSE_{left} and $\text{MSE}_{\text{right}}$ be the MSE of the left and right child nodes after the split, respectively. Let N_{parent} be the number of samples in the parent node, and N_{left} and N_{right} be the number of samples in the left and right child nodes, respectively. The reduction in error due to the split, denoted as ΔMSE, can be calculated as:

$$\Delta\text{MSE} = \text{MSE}_{\text{parent}} - \left(\frac{N_{\text{left}}}{N_{\text{parent}}} \times \text{MSE}_{\text{left}} + \frac{N_{\text{right}}}{N_{\text{parent}}} \times \text{MSE}_{\text{right}} \right)$$

The feature importance for a given feature is then the sum of the ΔMSE values for all splits where that feature is used, normalized by the sum of ΔMSE values for all splits in the tree. This normalization ensures that the feature importances sum to 1 (or 100% when expressed as a percentage), making it easier to compare the relative importance of features.

$$\text{Importance(feature)} = \frac{\sum \Delta\text{MSE(feature)}}{\sum \Delta\text{MSE(all features)}}$$

The computed importance scores are as follows in Figure 5.13: notably, 'feature1', which represents the premium payment amount, exhibits a dominant importance with a score of 92%. This observation aligns with our expectations, as the premium payment amount is our predictor. Beyond the premium payment amount, additional features have also been identified as making certain contributions to the model's predictive capabilities. Notably, 'feature9', 'feature11', and 'feature8', among others, have been recognized as relevant factors, albeit with less dominance than 'feature1'. These features, while secondary in their individual contributions, collectively enhance the model's accuracy and underscore the multifaceted nature of the prediction task.

The calculated importance scores serve as a valuable tool for understanding the relative impact of each feature within the model's decision-making framework. They shed light on the complex interplay of variables that the model relies upon to forecast outcomes. The prominence of 'feature1' reaffirms its central role in the prediction process, while the significance of the

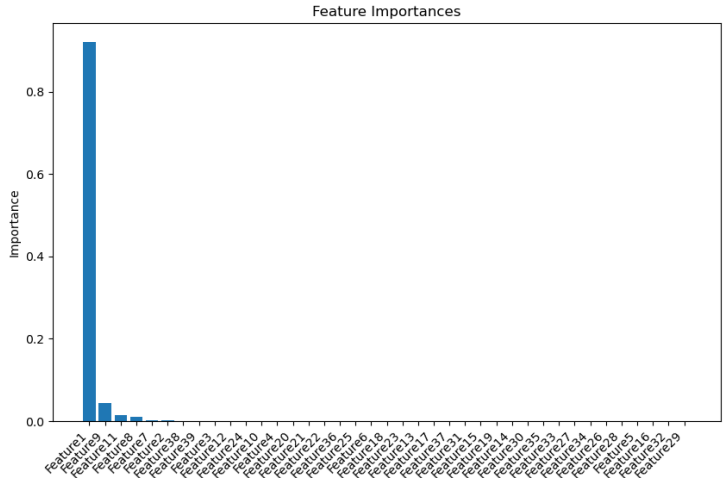

Figure 5.13: Feature Importance scores for LSTM Multivariate Model with 39 features

other features underscores the necessity of a holistic approach to modeling that encompasses a diverse array of predictors.

Leveraging the insights gained from the feature importance analysis, combined with domain expertise and an examination of the correlations among covariates, we proceeded to construct a refined model. This new model incorporates a carefully curated subset of 16 features, selected for their relevance and contribution to the predictive task at hand. The selection process was guided by the importance scores derived from the regression tree, which highlighted the most influential features. Domain knowledge played a crucial role in interpreting these results, ensuring that the selected features are not only statistically significant but also meaningful within the context of insurance premium forecasting. Additionally, we considered the interrelationships between covariates, aiming to minimize multicollinearity and retain features that provide unique and valuable information. The streamlined model, now operating with a reduced feature set, demonstrates a performance that is on par with the original model, which utilized all 39 features. The RMSE of the new model stands at 2427.71654, a marginal difference compared to the RMSE of 2444.36198 achieved by the full-featured model. This negligible change in RMSE suggests that the reduced model maintains its predictive accuracy while benefiting from increased simplicity and interpretability. The nearly equivalent RMSE indicates that the 16 selected features capture the essential patterns and relationships necessary for accurate predictions, without the additional complexity introduced by the full set of 39 features. This outcome underscores the effectiveness of feature selection in enhancing model efficiency and underscores the potential for

a more parsimonious model to deliver robust predictions in the domain of insurance premium forecasting.

5.4 Illustration with one example

To provide a more tangible comparison between the LSTM model and the MA model, we selected a single insurance policy as a case study. This particular policy features 89 payment instances over several years. The majority of these payments are consistent, hovering around the 1000 kr, with the exception of two significant outliers exceeding 15000 kr. The graphical representation of the models' predictions versus the actual data below offers a clear visual distinction between their performances. It is evident from the graph that the LSTM model more accurately captures the shifts in payment patterns, particularly in its ability to predict changes in payment behavior. However, it is worth noting that while the LSTM model demonstrates superior pattern recognition, it tends to underestimate the magnitude of the two anomalous high-value payments when compared to the MA model. This observation aligns with previous findings from residual analysis, which indicated that the LSTM model tends to underestimate higher values. This pattern of underestimation by the LSTM model, particularly for outlier payments, suggests an area for model refinement to improve its accuracy in forecasting payments of varying magnitudes. Despite this, the LSTM's overall predictive accuracy and its nuanced understanding of complex patterns underscore its potential as a valuable tool for forecasting insurance payment behaviors.

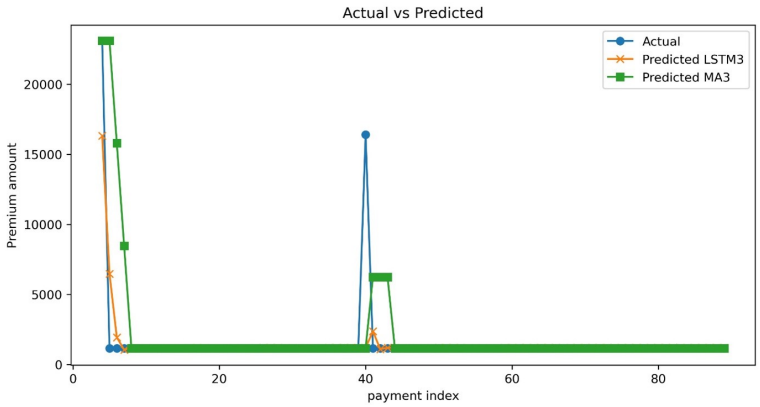

Figure 5.14: Actual vs Predicted LSTM with 16 features vs MA models for one example policy

6 Results and Future work

This study embarked on an exploration of time series forecasting by comparing the performance of LSTM (Long Short-Term Memory) neural networks against the traditional ARMA (AutoRegressive Moving Average) model. Initially, we constructed simulations of time series data to serve as a testing ground for these predictive models. The findings indicated that both ARMA and LSTM models were able to identify the underlying patterns within the data. However, the ARMA model demonstrated superior performance with respect to metrics such as the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE).

Advancing beyond simulations, the study applied LSTM models to real-world datasets, specifically targeting variable premium payments associated with unit-linked children's savings product. A meticulous exploratory data analysis was undertaken to gain a deep understanding of the dataset's characteristics. This phase was followed by extensive feature engineering, which included techniques like encoding to refine the data for LSTM modeling.

The study experimented with three distinct LSTM models:

1. A univariate model that solely considered a single variable, i.e. premium payment amount.

2. A multivariate model that incorporated a comprehensive set of 39 features.

3. A refined multivariate model that utilized a curated subset of 16 features, selected based on their feature importance.

The results were encouraging, as all LSTM models surpassed the baseline Moving Average (MA) model, achieving lower RMSE scores. Nevertheless, the univariate LSTM model displayed potential overfitting issues, which were apparent from the erratic behavior of the validation loss curve in proximity to the training loss curve. This suggested that the model might be too closely fitted to the training data, potentially compromising its performance on new data. On the other hand, the multivariate LSTM models showcased strong predictive capabilities, outshining the MA model in the benchmarks. The behavior of the validation loss curves for these models was particularly telling; they consistently displayed lower values in majority of epochs compared to the training loss curves. This pattern is indicative of a model's effective generalization, as it implies that the models were not merely memorizing the training data but were actually learning the underlying patterns, thus performing well on data they had not previously encountered.

An intriguing aspect of the study was the utilization of feature importance derived from a regression tree. This analysis informed the refinement of the multivariate LSTM model, leading to a version with a simplified structure yet comparable predictive prowess.

In conclusion, the study presents a compelling case for the adoption of LSTM models in the realm of time series forecasting, particularly in the

context of insurance unit-linked product. The LSTM models not only outperformed traditional benchmark but also demonstrate its ability to handling complexity of real-world data. Future studies could capitalize on the knowledge obtained from the LSTM models to enrich our understanding of customer premium payment patterns, which could lead to improvements in product design. Moreover, the forecasting models tailored for premium prediction could be scaled up and modified to suit an array of endowment insurance products, enhancing their relevance and practicality within the industry. Furthermore, LSTM models could be crafted to project additional pivotal metrics, such as lapse rates. By broadening the scope of these models, a more holistic evaluation of the determinants affecting the insurance sector could be achieved, yielding insightful forecasts that could guide risk management strategies and the development of insurance policies.

References

[1] Bao, W., Yue, J., & Rao, Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PLOS ONE*, 12(7), e0180944.

[2] Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2), 157-166. DOI: 10.1109/72.279181.

[3] Brockwell, P. J., & Davis, R. A. (2002). Introduction to Time Series and Forecasting. Springer.

[4] Chen, K. Y., Kuo, L., & Wang, C. H. (2005). Modeling the volatility of futures return in rubber and electronics for hedging strategy: An empirical study. *Applied Financial Economics*, 15(10), 731-743.

[5] Dickey, D. A., & Fuller, W. A. (1979). Distribution of the Estimators for Autoregressive Time Series with a Unit Root. *Journal of the American Statistical Association*, 74(366), 427-431.

[6] Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2), 179-211.

[7] Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2), 654-669.

[8] Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10), 2451-2471.

[9] Hamilton, J. D. (1994). Time Series Analysis (Chapter 17). Princeton University Press.

[10] Hansun, S. (2013). A new approach of moving average method in time series analysis. *Proceedings of Conference on New Media Studies*.

[11] Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Technische Universität, Munich.

[12] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8), 1735-1780.

[13] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning: with Applications in R. (Chapter 3). Springer.

[14] Kingma, Diederik P., and Jimmy Ba. (2014). Adam: A Method for Stochastic Optimization. *Proceedings of the International Conference on Learning Representations*.

[15] Lindholm, M. & Palmborg, L. (2021). Efficient Use of Data for LSTM Mortality Forecasting. Available at SSRN 3805843.

[16] McNally, S., Roche, J., & Caton, S. (2018). Predicting the price of Bitcoin using machine learning. *Proceedings of 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing*.

[17] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by backpropagating errors. *Nature*, 323/6088, 533–536.

[18] Raschka, S., & Mirjalili, V. (2017). Python Machine Learning (Chapter 4). Packt Publishing.

[19] Wüthrich M.V. & Merz M. (2022). Statistical foundations of Actuarial learning and its applications. Springer.