CS230

# Time to Default & Time to Prepayment Estimation Using LSTM and Deep Learning Techniques

**Peter G. Drembelas**
Department of Computer Science
Stanford University
pdrembel@stanford.edu

**Vibhor V. Khar**
Department of Computer Science
Stanford University
vvkhar@stanford.edu

## Abstract

The ability to project mortgage prepayment and default is of critical use to financial institutions from a credit and interest rate risk perspective. Prepayment and default are behavioral decisions and, historically, it has been difficult to model these behaviors. Nonetheless, using an LSTM architecture, of which we have found no precedence in the literature, we trained the model on 20 months of mortgage loan data and predicted on 20 months of loan data from a subsequent timeframe that had not been shown to the model and achieved an accuracy of 90%

## 1 Introduction

Default and Prepayment are two of the biggest risks that an investor/originator of a mortgage faces. The team aims to design a model that is capable of accurately predicting mortgage prepayments and defaults using Deep Learning Architecture. Both prepayment and default are highly complex behavioral decisions and developing models to predict human behavior is challenging. Another interesting feature of the problem is that prepayments and defaults are competing risks and traditionally models needed to be considerate of this fact. It is unclear if this complication for traditional models will pose a challenge for a deep learning model.

Consider a mortgage $i$, we are interested in estimating the duration $T_i$ of the interval in between the event of interest (prepayment or default) for $i$ and the time $t_0$ at which we start to measure time for $i$. We allow for right censored data, namely, data for which we do not know when the event occurred, but only that it did not occur before a censoring time $C_i$. The observed time $Y_i$ is defined as $Y_i = min(T_i, C_i)$, and each datapoint corresponds to the pair $(Y_i, \delta_i)$ where $\delta_i = 0$ if the event is censored (in which case $Y_i = C_i$) and $\delta_i = 1$ otherwise. There are many techniques to deal with censored and truncated data [1] but none which use Deep Learning techniques.

This model endeavors to predict the status of a mortgage at a certain time, given it has not faced a hazard event in the past. A Long Short Term Memory (LSTM) model was developed with 45 input characteristics that include various time-insensitive / time-dependent loan level characteristics and macroeconomic variables. The training dataset is comprised of 1,000,000 records with 20 month loan histories for each loan and the model was trained to predict hazard events within the 20 months. The model accuracy is then scored on an out-of-time development / test datasets which is comprised of 20 months of data from 2015.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LaTeXtemplate borrowed from NIPS 2017.)

## 2 Related work

There is a lot of academic research that has gone into this problem. The focus has mainly been on Mortgage Backed Securities[2][3] and depend on traditional mathematical / statistical approaches to the problem. Techniques were of two varieties, the simple classification technique of logistic regression and survival statistics techniques of Meier Kaplan and later Cox Proportional Hazards (CPH) Modeling shown in equation 1 with the likelihood of the event shown in equation 2. CPH is a marked improvement over its predecessors but has well known and documented deficiencies [4].

$$\lambda(t|X_i) = \lambda_0(t)e^{\beta_1 X_{i1}+\ldots+\beta_p X_{ip}}e^{X_i\beta} \tag{1}$$

$$L_i(\beta) = \frac{\lambda(Y_i|X_i)}{\sum_{j:Y_j\geq Y_i}\lambda(Y_i|X_j)} = \frac{\lambda_0(Y_i)\theta_i}{\sum_{j:Y_j\geq Y_i}\lambda_0(Y_i)\theta_j} = \frac{\theta_i}{\sum_{j:Y_j\geq Y_i}\theta_j} \tag{2}$$

where $\theta_j = e^{X_j\beta}$ and the summation is over the set of subjects $j$ where the event has not occurred before time $Y_i$ (including subject $i$ itself). Obviously $0 < L_i(\beta) \leq 1$.

There has been some recent academic research on the use of Machine Learning Techniques to the problem mainly focused on fully connected neural networks[5]. The team noted that there was no recent attempt to incorporate the temporal aspect of the problem and come up with a solution using AI / ML methods.

In the world of medicine, however, the team found much wider adoption of Deep Learning methodology for modeling survival. Survival Analysis is similar to modeling a hazard event and thus lends itself to transfer learning from the newest research in medical artifical intelligence [6] [7] [8]. The method of using LSTM on multidimensional data has been demonstrated in survival analysis and the prior research has proved tremendously useful to the work outlined in this paper.

## 3 Dataset and Features

We have three data sources with locations specifed below.

**Fannie Mae Single-Family Loan Performance**

`https://www.fanniemae.com/portal/funding-the-market/data/loan-performance-data.html`

**HPI**

`https://www.fhfa.gov/DataTools/Downloads/Pages/House-Price-Index-Datasets.aspx`

**Unemployment at MSA level**

`(https://fred.stlouisfed.org/search?st=unemployment+by+msa`

Our X (input) dataset contains the various explanatory variables like property type, original loan balance, original loan to value ratio, loan age, state-wise home price index, msa-level unemployment rate, etc. Forty five such explanatory variables were used. Categorical variables were one hot encoded, whereas the continuous variables were demeaned and scaled to have a standard deviation of 0.5 see figure 1 as an example of data normalization. to increase optimization efficiency. Data was pulled from SQL databases using the SQLAlchemy [9] package. We then used numpy [10] and pandas [11] throughout the project to perform data exploration and data cleaning with matplotlib [12] and seaborn [13] libraries used for data visualization. Statistics and simple econometrics were assisted with the SciPy [14] and Scikit-learn [15] packages. The entirety of the code we used can be found on github [16].

Y (output) dataset contained labels for Current (no hazard), default, prepay and removed states. The last state would turn on after a hazard event had occurred. Given that mortgage loans prepay/default

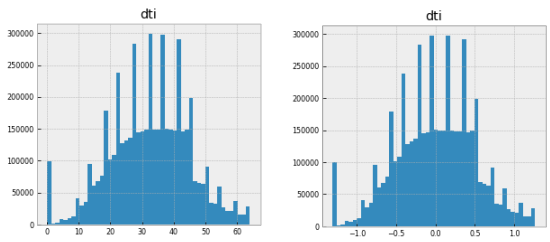


Figure 1: DTI as an Example of Data Normalization

at random times we decided we zero padded our data, i.e., all the X variables' value of the the loan in question are set to 0 after it experiences a hazard event, whereas the removed label in Y would be turned on thereafter. In the time instant when the hazard occurs, the corresponding hazard flag is turned on. In order for the specific trait of zero padding not conveying information to the model regarding the timing of event, we decided to train the model on a reduced dataset in terms of time elapsed (first 20 months of data) even though we have loan history data for 20 years.

Also, we chose a data sample from several years later to create the test and dev datasets to check if the model would predict the loan state for each of those twenty months accurately. We ensured that the test and development datasets, each consisting of records for ~30K loans for twelve months, were of the same distribution. A zero pad was using in these datasets too in case a loan experienced a hazard event. Using a relatively small test/dev combination (~360K records each) from several years apart was done to make the task of prediction challenging for our RNN. Given the paucity of defaults and the abundance of current state loans the dataset was massively unbalanced. We dealt with it using weighted loss function described in the methodology section.

## 4 Methods

Given that the data used in the development/projection is sequential in nature, the team decided to use an LSTM architecture, given its documented use for such purposes. As shown in the figure 2 below, activation of a certain time instant are fed back to the network for use in the next time instant. The complete architecture is shown in figure 3

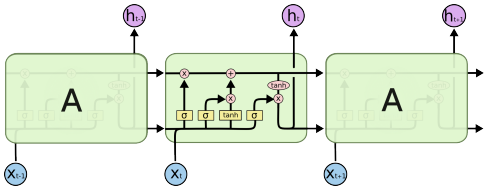


Figure 2: Chain of Three LSTM Cells [17]

Equations 3 below describe how the update and forget gates work and impact the activation of the next time instant when used with the output gate. We did not use any gradient clipping since we did not notice our weights/loss becoming NaN/very large. It is possible that this was due to the fact that we trained the model on just 20 time instances.

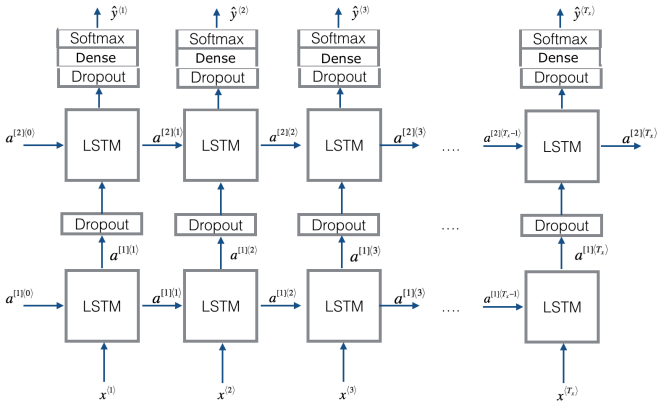


Figure 3: Complete Network Block Diagram

**LSTM**

$$\tilde{c}^{<t>} = tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \tag{3a}$$
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \tag{3b}$$
$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \tag{3c}$$
$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \tag{3d}$$
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>} \tag{3e}$$
$$a^{<t>} = \Gamma_o * c^{<t>} \tag{3f}$$

The output state prediction is defined by by the softmax function:

$$\hat{y} = \frac{e^{x_i}}{\sum_{i=1}^{m} e^{x_i} + \epsilon} \tag{4}$$

For our loss function, we chose a weighted categorical cross entropy function because the data is very imbalanced where defaults represent a very small proportion of the total dataset and in order to not overtrain on prepayments of no-events it's important to emphasize the importance of the default, explicitly, through the loss function:

$$\mathcal{L} = -\frac{1}{T_x}\frac{1}{m}\sum_{t=1}^{T_x}\sum_{j=1}^{N_{states}}\sum_{i=1}^{m} w_j y_{i,t} \log \hat{y}_{i,t} \tag{5}$$

**Final Weights**

The final weights were chosen on the basis of the inverse proportion of the rate of occurrence in the dataset.

| Current | Prepay | Default | Removed |
|---|---|---|---|
| 0.005 | 0.24 | 0.75 | 0.005 |

Table 1: Loss Weights Used

## 5 Experiments/Results/Discussion

We considered the same variables in the CPH method as well as the LSTM, and for sufficiently large number of neurons the RNN was easily outperforming the baseline. Therefore the task at hand was to

ensure that the model was not over fitting on the train and dev datasets. This was carefully checked when hyperparameter tuning was undertaken.

The various hyperparameters that were tuned and the various values for them are shown in Table 2 below.

| Batch Size | Hidden Units | Layers | Optimization | Dropout |
|---|---|---|---|---|
| 32, 64, 128 | 15, 20, 30, 40 | 1, 2 | RMSProp, Adam | 0.0, 0.08, 0.15, 0.2 |

Table 2: Hyperparameters tested

Due to paucity of computing power (the team could not get an AWS GPU instance faster than the local machine for unknown reasons), the team did not attempt to run several randomly spaced values of hyper-parameters. Instead, for the number of hidden units and dropout rate the team did not use evenly spaced intervals (to mimic randomness) since optimum values could be found at random intervals too[18].

The team used TensorFlow[19] and Keras[20] to implement the model shown in figure 3 with hyperparameter settings seen in Table 3

| Batch Size | Hidden Units | Layers | Optimization | Dropout | Epochs |
|---|---|---|---|---|---|
| 64 | 30 | 2 | RMSProp | 0.08 | 10 |

Table 3: Final Hyperparameters

This decision was made on the basis of obtaining a reasonably high accuracy in the training set over the baseline, and ensuring that the model does not overfit over train and dev samples. We noted that our model not only predicts the occurrence of a hazard event with 94% accuracy in the train dataset/ 90% accuracy in dev / test datasets, it is spot on when it comes to predicting the timing of the event.

When using sufficiently large number of neurons (>20), the team noted that the RNN was easily outperforming the baseline Cox-Proportional Hazard Model (accuracy of 73%). Therefore, the task at hand was to ensure that the model was not overfitting on the train and dev datasets. This was carefully checked when hyperparameter tuning was undertaken and played a critical role in the selection of the final model.

Note the CPH model was run on an AWS instance m5dn.8xlarge with 32, 128 GB of RAM, and a 256 GB SSD hard drive. The NN was implemented on an i7 processor based computer with a Radeon GPU, 32 GB of RAM, and a 1 TB hard drive.

## 6 Conclusion/Future Work

The use of LSTM for modeling mortgage prepays and defaults was the novelty of this project. The team learning that adding more layers / neurons adds to train data accuracy, however, it comes at the cost of overfitting on the train and dev datasets. Therefore, we included dropout and only selected the model specification where test and dev dataset performance was to each other and to the train dataset accuracy. Given that the team selected a small sample of the total data, there is tremendous potential to increase train data accuracy by training on larger samples given for higher computing power allowances. Also, there is potential to mimic the architecture used in RNN Surv [6] and then tailor to the mortgage domain.

## 7 Contributions

Each member of the team contributed immensely in all areas including research and literature review; data sourcing, cleaning, and transforming; feature selection; algorithm development and testing; and report writing.

## References


[1] John P Klein and Melvin L Moeschberger. *Survival analysis: techniques for censored and truncated data*. Springer Science & Business Media, 2006.

[2] Eduardo S. Schwartz and Walter N. Torous. Prepayment and the valuation of mortgage-backed securities. *The Journal of Finance*, 44(2):375–392, 1989.

[3] Mikhail Chernov, Brett R Dunn, and Francis A Longstaff. Macroeconomic-driven prepayment risk and the valuation of mortgage-backed securities. Working Paper 22096, National Bureau of Economic Research, March 2016.

[4] Paul Allison. For causal analysis of competing risks, don't use fine & gray's subdistribution method, March 2018.

[5] Justin Sirignano, Apaar Sadhwani, and Kay Giesecke. Deep learning for mortgage risk, 2016.

[6] Michael F. Gensheimer and Balasubramanian Narasimhan. A scalable discrete-time survival model for neural networks, 2018.

[7] Eleonora Giunchiglia, Anton Nemchenko, and Mihaela Schaar. Rnn-surv: a deep recurrent model for survival analysis. In *International Conference on Artificial Neural Networks*, pages 23–32. Springer, 10 2018.

[8] Jared L. Katzman, Uri Shaham, Alexander Cloninger, Jonathan Bates, Tingting Jiang, and Yuval Kluger. Deepsurv: personalized treatment recommender system using a cox proportional hazards deep neural network. *BMC Medical Research Methodology*, 18(1), Feb 2018.

[9] Michael Bayer. Sqlalchemy. In Amy Brown and Greg Wilson, editors, *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks*. aosabook.org, 2012.

[10] Stéfan van der Walt, S. Chris Colbert, and Gaël Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[11] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[12] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science Engineering*, 9(3):90–95, May 2007.

[13] Michael Waskom, Olga Botvinnik, Drew O'Kane, Paul Hobson, Saulius Lukauskas, David C Gemperline, Tom Augspurger, Yaroslav Halchenko, John B. Cole, Jordi Warmenhoven, Julian de Ruiter, Cameron Pye, Stephan Hoyer, Jake Vanderplas, Santi Villalba, Gero Kunter, Eric Quintero, Pete Bachant, Marcel Martin, Kyle Meyer, Alistair Miles, Yoav Ram, Tal Yarkoni, Mike Lee Williams, Constantine Evans, Clark Fitzgerald, Brian, Chris Fonnesbeck, Antony Lee, and Adel Qalieh. mwaskom/seaborn: v0.8.1 (september 2017), September 2017.

[14] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. *arXiv e-prints*, page arXiv:1907.10121, Jul 2019.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[16] Peter G. Drembelas and Vibhor V. Khar. Time to default & prepayment estimation deep learning. https://github.com/pdrembelas/mortgage, December 2019.

[17] Christopher Olah. Understanding lstm networks, August 2015.

[18] Karsten Eckhardt. Choosing the right hyperparameters for a simple lstm using keras, November 2018.

[19] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[20] François Chollet et al. Keras. `https://keras.io`, 2015.