CHAPTER 1

TECHNICAL CONTRIBUTIONS

Matthew Louis Rosendin
University of California, Berkeley
Department of Industrial Engineering and Operations Research

April 13, 2018

Contents

List of Figures

1 Introduction

Our project, in partnership with Adobe Systems Inc., is an interactive dashboard that forecasts monthly active users (MAU) for any Adobe software product. We've accomplished forecasts with excellent accuracy as far as 5 months into the future. The growth model's forecast can be used for a number of reasons, including as an input to forecasting revenue or as a measure of product health. While forecasting monthly active users is nothing new, our application of artificial neural networks (ANNs) seems to be a novel approach that has yielded promising results.

In this chapter of the paper I analyze and discuss the implications of our model's performance and describe my work in deploying our model. I begin by explaining the concrete goals of this work, then outlining the motivation, before finally diving into a technical introduction to our model. Following the basic conceptual discussion around our model, I start the discussion about how I refined the "raw" model code into a flexible, scalable, and configurable object. The translation of the model into a manageable machine learning software system led to additional improvements to the model, such as hyperparameter optimization. A byproduct of the optimization was an exhaustive experimentation of model performance metrics. The underpinning of the preceding work is the user interface. I discuss my work in front-end and systems engineering that resulted in the final deliverable: a forecasting dashboard powered by machine learning.

1.1 Goal

Our team's foremost goal is to create a practical and interactive product growth model for Adobe senior management. My individual goal was to design and create a user interface to interact with the model. Throughout my contributions, I've added additional value around testing the performance of our model and gathering relevant metrics.

1.2 Motivation

The motivation for creating a dashboard is straightforward. The dashboard enables the user to forecast and analyze a product's future MAU. The motivation for the section on hyperparameter optimization was simply to improve the results of our model. From the resulting tests, we've found that our model's forecasting capability had improved. Part of the discussion in this paper is reserved for analyzing the source of model improvement.

1.3 Time Series Forecasting

In order to explain why we chose our particular model (a long short-term memory network) I will explain the characteristics of the problem we are solving through a brief taxonomy. The problem is best framed as, "how do we forecast monthly active usage multiple weeks into the future?". The most salient characteristic of this problem is that the data is a long sequence (i.e., data is chronologically ordered) of values, known as a time series. What we are trying to achieve is a forecast of time series data, so the solution to the problem is known as time series forecasting.

1.4 Autoregressive Models

The simplest model might be an autoregressive (AR) model in which the forecast values are regressed on prior data:

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \epsilon_t \tag{1}$$

where c is a constant, ϵ_t is an error term, and $\phi_1, ..., \phi_p$ are the parameters to be estimated by linear regression. To forecast a value for the present time t, we specify a parameter p, known as the "order" or "lag". With $p = 3$ the corresponding model would look like this:

$$X_t = c + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \phi_3 X_{t-3} + \epsilon_t \tag{2}$$

We would call this an autoregressive model with a lag of 3 time steps where time steps are defined as the number of sequential data points. The autoregressive model can be notated as $AR(3)$ to indicate its order with $p = 3$. Certain information criteria, such as Akaike's information criterion (AIC) or Bayesian information criteron (BIC), are commonly used to select the best lag length [7].

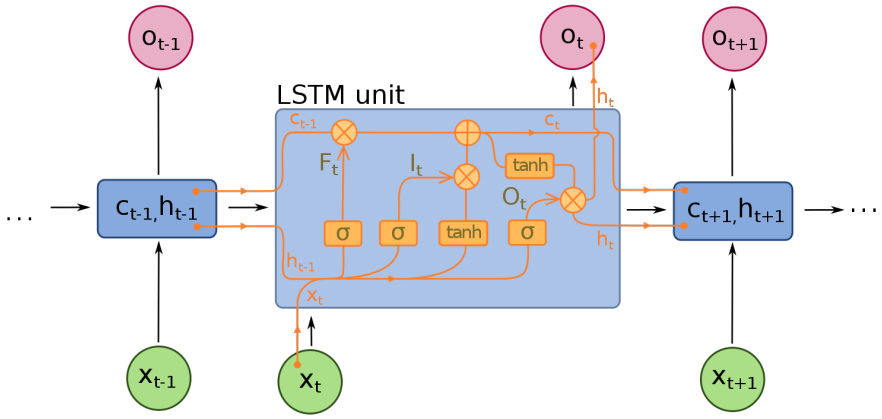
In our work we use the concept of "lag" from the autoregressive model in our supervised learning model. Our dataset is multivariate, meaning there are more than two observations for each time step. Since we would like to forecast multiple weeks into the future, our model is multi-stepped.

1.5 Recurrent Neural Networks

There are many good reasons to justify using an artificial neural network. Much of the recent literature on time series forecasting is focusing on the advantage of ANNs versus other methods,

including autoregressive interactive moving average (ARIMA) models. For instance, Ahmed et al. and Zhang show that ANNs are shown to be superior for generalized time series problems [2]. Although complex models such as neural networks can hard to interpret, our project team values performance over interpretability. Furthermore, we have designed features similar to those in financial time series models. For financial stock data, Adebiye et al. show that ANNs are superior to ARIMA models [1]. Lastly, ANNs are capable of modeling non-linearities which can improve the forecast.

Figure 1: Diagram of a one-unit Long Short-Term Memory (LSTM) network [4]



Recurrent neural networks (RNNs) are a class of artificial neural network that use reasoning about previous events in the data to inform predictions [9]. In other words, RNNs retain state at one time to the next, using the previous state's output for the current estimate. This characteristic enables multi-stepped time series forecasting. A particular architecture of RNNs known as Long Short-Term Memory (LSTM) allows the model to recognize and retain short-term patterns for long periods of time [5]. This architecture is used best where data from an earlier state needs to be recalled at a later state. Examples of LSTM networks in industry include Google Voice [3] and state-of-the-art performance on Google Translate [6].

2 Object-Oriented Abstraction

The working LSTM model was created in a Jupyter Notebook. While serving as a minimum proof of concept, the model should not require that the end-user engage directly with code. Furthermore, the following criteria were set: 1) the lifecycle of the model from training, testing, to

final predictions needed to be automated and callable with a simple applicaition programming interface (API); 2) the Jupyter Notebook code (which executes similarly to a script) needed to be organized into an object-oriented structure to store data within the object instance; 3) the model needed to be capable of loading datasets of arbitrary products. Criterion (2) is more a matter of preference and style to the author, as other programming paradigms could be considered. The rest of this section briefly discusses the process involved in restructuring the model code to meet the criteria.

2.1 Decomposition

The goal of decomposition is to break a complex system into parts that are easier to understand, program, and maintain. The process of decomposing the Jupyter Notebook code involved defining separate modules for loading datasets (`data.py`), feature engineering (`features.py`), training/testing/predicting (`core.py`), and accessing utilities (`utils.py`). Furthermore, the code was organized into functions, and eventually into methods (discussed below).

2.2 Encapsulation

After breaking the code into manageable chunks, I created a Python class wrapper to store internal state and to allow the end-user to specify runtime options. The entire model can be run by using the wrapper's simple API.

3 Hyperparameter Optimization

Hyperparameter optimization deals with the learning parameters of a model rather than weights for features. While I made a significant effort in this area, the optimization was only run on one machine. Increasing compute resources to perform a more extensive search may yield further interesting results.

3.1 Cross-validation

In time series problems, cross-validation is performed chronologically with splits at fixed time intervals. In my trials, I ran cross-validation with 2, 3, and 10 splits. I wanted to understand the change in our key metric, MAPE (mean absolute percent error), as the number of splits S increases. According to the `scikit-learn` documenation, the training set Tr for the ith split has a size

$$Tr_i = \left\lfloor \frac{i \times N}{(S+1)} \right\rfloor + N \bmod (S+1) \tag{3}$$

where N is the number of samples, S is the number of splits, and $\lfloor expression \rfloor$ denotes the floor function. Meanwhile, the test set Te has the size shown in equation 4.

$$Te_i = \left\lfloor \frac{N}{(S+1)} \right\rfloor \tag{4}$$

For time series splits with $S > 1$, the sum of the first train/test set is strictly less than N and the last train/test set is equal to N:

$$Tr_i + Te_i < N \quad \forall i \in \{1, \ldots, S-1\} \tag{5}$$

$$Tr_S + Te_S = N \tag{6}$$

These equations describe the train/test split pairs. It is important to note that the chronological indexing of the data is preserved for both input data and additional "held-out" data. This splitting strategy helps validate the results of the exhaustive hyperparameter search that is described in the next section.

3.2 Grid Search

This search algorithm is simple. It exhaustively generates candidate learning parameters and selects the best candidate based on the scoring function (MAPE). The number of candidates C is equal to all possible combinations of user-specified parameter values. For n parameters, the number of candidates is equal to

$$C = \prod_{i=1}^{n} x_i \tag{7}$$

where x_i is the number of values for the ith parameter. Each candidate is fit using the time series split strategy discussed in Section 3.1 to help validate the candidate's score. As a result, the number of fits is equal to the number of candidates times the number of splits.

$$\mathrm{F} = C \times S \tag{8}$$

For further discussion about alternative parameter seearch algorithms, see Section 6.3.

3.3 Results

Many trials with cross-validation were run, producing standardized output containing the learning parameters, test loss, and error metrics. Example output of this process is shown in Figure 2. Even though they are refitting multiple times, cross-validated trials completed very fast. The system is not so robust, failing 1/3rd of the time; typically 300% of the host machine's CPU usage was exceeded during failures. Although few learning parameters were tested, the cross-validation results are very consistent. We are reasonably sure that our model will perform well with unseen data. As a note, choosing the hyperparameters with higher cross-validation errors will help prevent overfitting when forecasting [8].

Table 1: Cross-validation Trials

Trial	Product	Length	MAPE	Runtime	Candidates	Splits	Fits	Failure
1	1	16 w	5.13%	4:26	8	3	24	-
2	1	16 w	5.06%	8:24	8	3	24	-
3	1	16 w	-	-	8	3	24	✓
4	2	16 w	4.13%	1:59	8	3	24	-
5	2	16 w	4.28%	2:17	8	3	24	-
6	2	16 w	-	-	8	3	24	✓
7	3	16 w	5.08%	2:04	8	3	24	-
8	3	16 w	-	-	8	3	24	✓
9	3	16 w	4.82%	2:10	8	3	24	-

4 Systems Engineering

The goal at the onset was to develop a minimal machine learning system to avoid the problem of technical debt [10]. I worked to productionize our model by building a back-end that is capable of extracting and preparing data, training and validating the model, and producing forecasts. The back-end performs the business logic computation without end-user interaction. The back-end was written entirely in Python and its component parts are discussed in this section.

4.1 Server Infrastructure

All software written for this project was developed and tested on local machines. System specifications of the host machine producing the results in this report are shown in Figure 3. The system

Figure 2: Example of grid search output

```
Parameters: {
  'batch_size': 150,
  'epochs': 200,
  'optimizer': 'adam',
  'units': 100
}
Test loss: [128.97596047141334, 128.97596047141334]
Testing set RMSE: 24318.87
Testing set MAPE: 5.79%
Testing set sMAPE: 0.06%
```

runs Python 3.6.4 and we've tested on local machines (macOS) and a Docker virtual host container running Ubuntu 16.04. A simple web server framework, Flask (`http://flask.pocoo.org/`), is employed to interact with the front-end (see in Section 5). A basic API has been developed to enable asynchronous communication with the front-end client device.

4.2 Distributed Task Queue

Since the machine learning processes are long-running processes, an open-source distributed task queue called Celery (`http://www.celeryproject.org/`) is used to manage scheduling, state changes, and output. When the user makes a new forecast, an API endpoint queues the task. In our case, there was only one client utilizing the queue at a time. However, the additional benefit of using a task queue is to coordinate background jobs. When the end-user begins a new forecast task, they receive a response from the server immediately. The code on the client polls another API endpoint every second to check the status of the forecast task. When the status appears successful, the server responds with the necessary data to render the visualizations on the client device, as well as summary statistics.

5 User Interface

The user interface (UI) ties together my team's work on building an advanced machine-learning model and providing real value to Adobe senior management. Creating the UI was a careful process because it had to meet the needs of our industry sponsor (Adobe) and it had to integrate

Figure 3: Host machine system information

with our back-end. My general approach involved gathering requirements, sketching the UI, presenting the sketch for feedback, developing visual elements, and iterating.

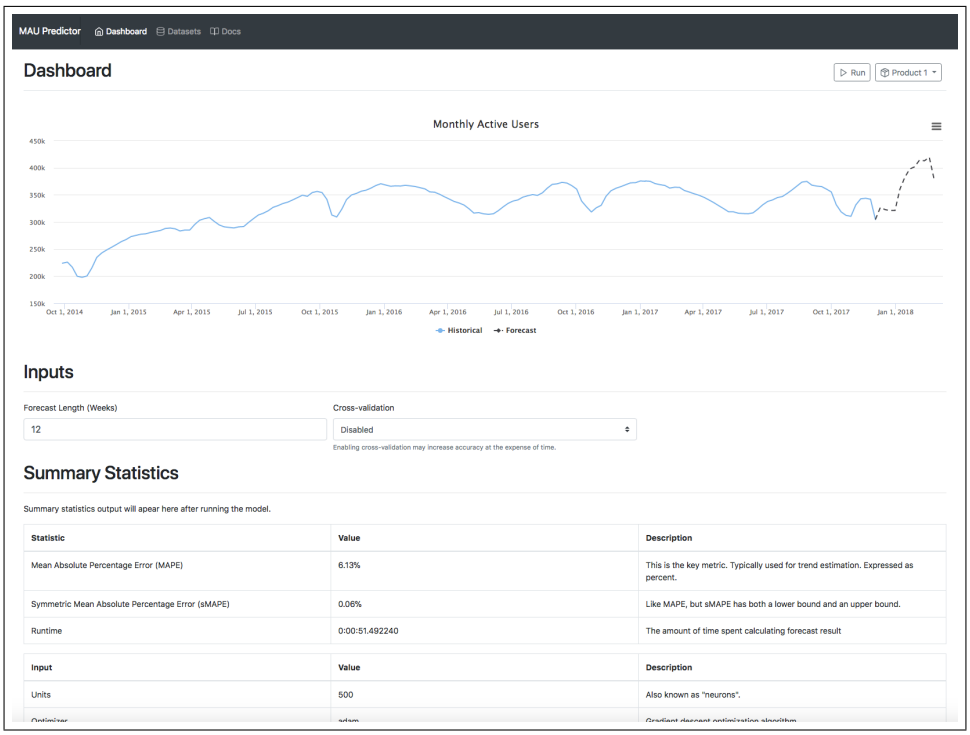
5.1 Forecast Visualization

The forecast app shows a line chart of historical data and forecasted monthly active users. The end-user is able to isolate the forecast trendline and export the chart to various image formats. Charts related to forecasting are shown in the Appendix.

5.2 Documentation

I also wrote documentation for the forecasting app with three sections tailored to different stakeholder segments. The first section provides basic usage instructions for the non-technical end-user. The second section, tailored towards data scientists, goes more in-depth about the organization of the model code and how the cross-validation works. Lastly, the third section provides all technical documentation about the application itself, tailored towards developers.

Figure 4: Dashboard user interface



6 Next Steps

6.1 Websockets

Currently, the forecast app polls the task status endpoint every second. The issue is that the task's status may not have changed, resulting in network overhead. Websockets provide a better alternative to HTTP. A websocket channel will enable the client device to subscribe to "events" from the back-end. For instance, when the task completes, the back-end can publish a new event that notifies the client device in real-time. However, websocket support with Flask may be limited and this improvement would not yield benefits unless the forecast app being used by multiple parties.

6.2 Data Pipeline Automation (Apache Hive/Hadoop)

Currently, new datasets must be moved to the project data folder manually. A future improvement may involve a direct connection to Adobe's Hive and Hadoop data services in order to automate

the pull of new data. For instance, a SQL query editor may enable the end-user to create or modify new or previous queries to forecast on various datasets. Perhaps an intermediate step would involve an interface for managing CSV file datasets from within the host file system.

6.3 Improving Hyperparameter Optimization Efficiency

The brute-force search algorithm used to optimize hyperparameters can probably be replaced with another. The `GridSearchCV` class from `scikit-learn` implements the algorithm that was used for our project, but other hyperparameter optimizers are available in the `model_selection` module of `scikit-learn`. The other options include a randomized search where a subset of the parameter grid is sampled. This requires passing in distributions of the hyperparameters themselves, which could be an interesting opportunity to avoid an exhaustive search.

7 Conclusion

The forecast app creates 16 week forecasts in under 1 minute with a 5% error rate and the results have been cross-validated. The model was productionized into a dashboard interface that can be deployed anywhere, for any product. There are still further steps that would improve the user experience and model accuracy. There is also further research to be conducted around verifying the efficacy of our model and for architectural improvements.

References

Adebiyi, A. A., Adewumi, A. O., & Ayo, C. K. (2014). Comparison of arima and artificial neural network models for stock price prediction. *Journal of Applied Mathematics*.

Ahmed, N. K., Atiya, A. F., Gayar, N. E., & El-Shishiny, H. (2010). An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews, 29*, 5-6.

Beaufays, F. (2015). *The neural networks behind google voice transcription*. `https://research.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html`.

Deloche, F. (2017). *Diagram of a one-unit long short-term memory (lstm) network*. `https://commons.wikimedia.org/w/index.php?curid=60149410`.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*, 1735-1780.

Le, Q. V., & Schuster, M. (2016). *A neural network for machine translation, at production scale*. `https://research.googleblog.com/2016/09/a-neural-network-for-machine.html`.

Liew, V. K. (2004). Which lag length selection criteria should we employ? *Economics Bulletin*, 1-9.

Ng, A. (1997). Preventing "overfitting" of cross-validation data. *ICML '97 Proceedings of the Fourteenth International Conference on Machine Learning*, 245-253.

Olah, C. (2015). *Understanding lstm networks*. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, ., D., & Dennison, D. (2015). Hidden technical debt in machine learning systems. *NIPS '15 Proceedings of the 28th International Conference on Neural Information Processing Systems, 2*, 2503-2511.

Appendices

Figure 5: Product 1 trend: 6.24% MAPE

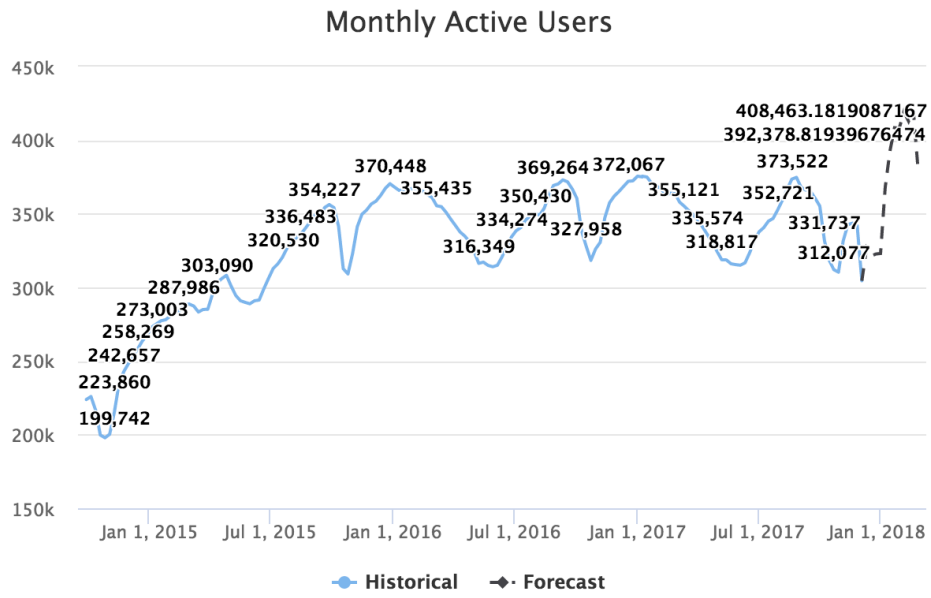


Figure 6: Product 1 forecast: 6.24% MAPE

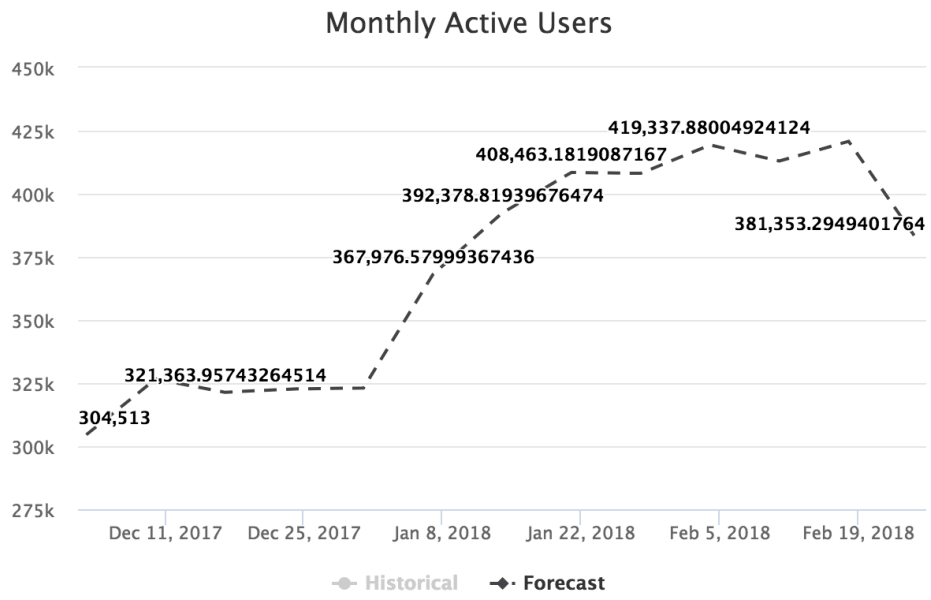


Figure 7: Product 2 trend: 6.05% MAPE

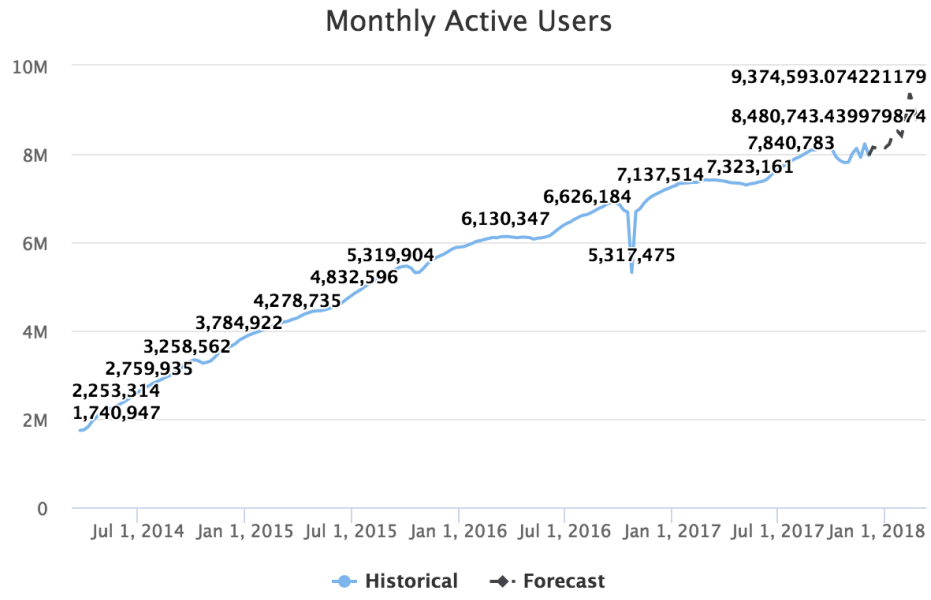


Figure 8: Product 2 forecast: 6.05% MAPE

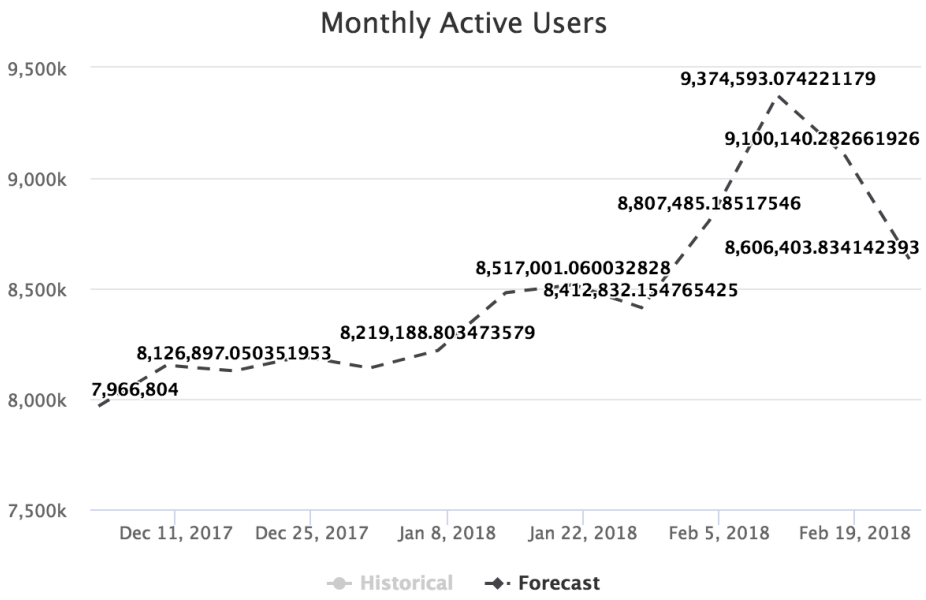


Figure 9: Product 3 trend: 4.82% MAPE

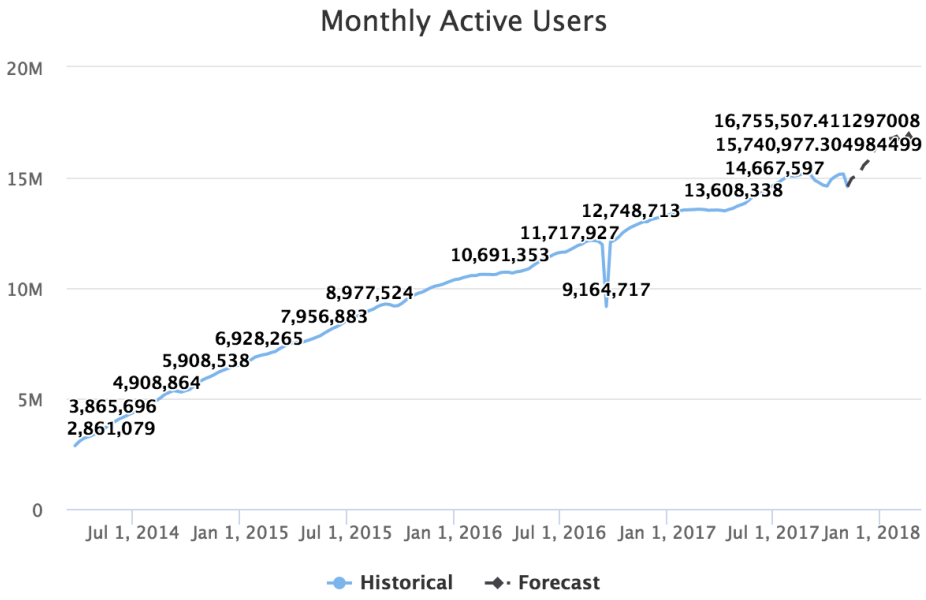


Figure 10: Product 3 forecast: 4.82% MAPE