# Furniture Sales Forecasting

This work uses a dataset consisting of the sales history of a retail store. The aim of this work is to investigate different forecasting models to predict the future sales of furniture. I will be implementing and comparing the performance of a basic Prophet model, a Prophet model incorporating holiday information, a Vanilla LSTM, and a Stacked LSTM. These methods were chosen because they performed best on similar sales forecasting tasks from other research papers.

The performances of the models are compared using multiple different measures- Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE)).

Different architectures and hyperparameters have been investigated for each model, and the results of the best performing architectures can be found below.

## Dataset

## This work uses the Superstore sales dataset. This dataset describes the retail store’s sales from 2014 to the end of 2017, containing 9994 data points with 21 features, and has zero missing values. It consists of sales information from three different categories- furniture, technology, and office supplies. The data is resampled on a monthly frequency, and the end of the month is set as the index.

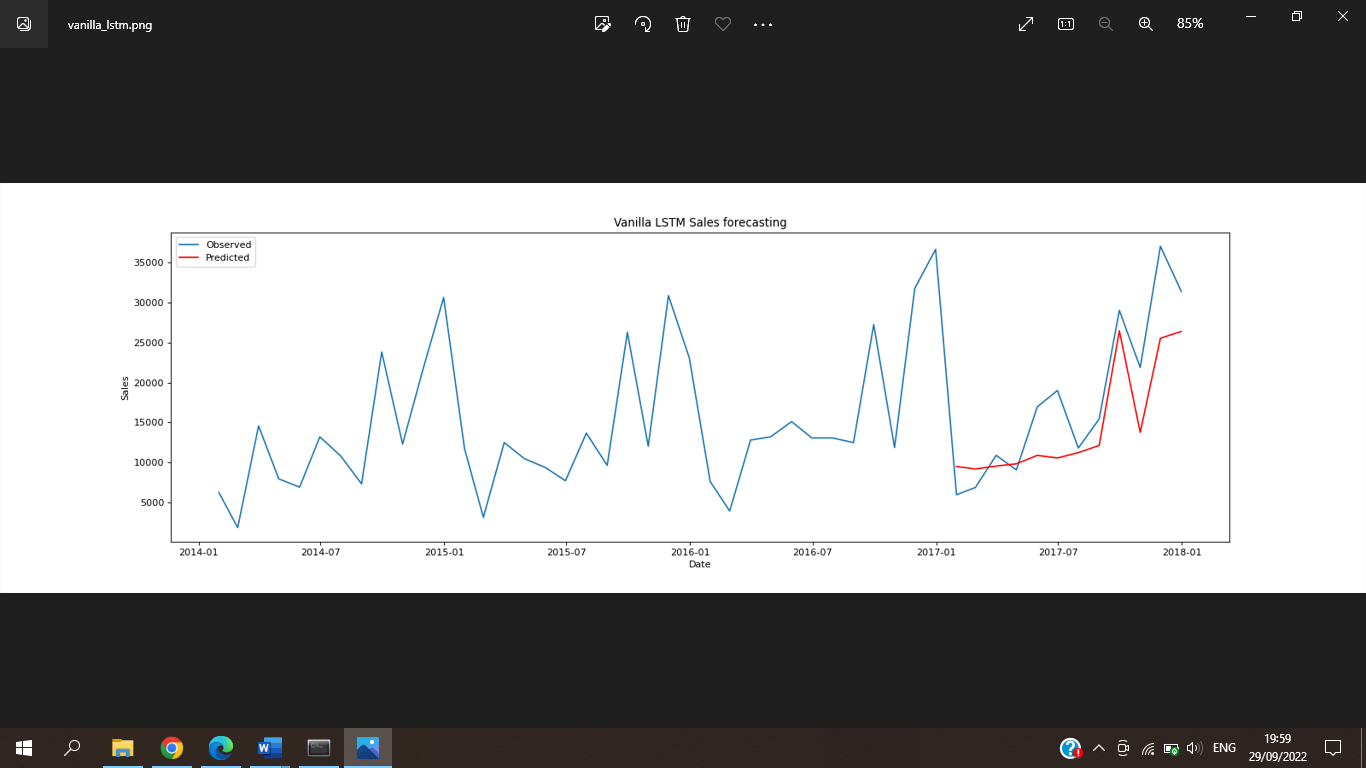
## The only category that displays seasonality in its sales pattern is furniture. Furniture sales tend to peak at the end of each year, and then decline after the holidays. Unlike furniture, there is no strong seasonal characteristic that can be observed in the technology and office supply categories. Their sales data does not experience any specific trend that recurs every year. Therefore, among the three available categories, only the furniture category will be investigated due to its seasonal pattern.

## Prophet Model 1

Facebook has released its own open-source time series model called Prophet. It uses historical trend fluctuations to make accurate forecasts, even if given only a small amount of training data. It can also include the effects of seasonality and holidays in its forecasts.

The first implemented Prophet model does not include any seasonality or holiday parameters. Below are the best results obtained using this model.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Architecture** | **MSE** | **RMSE** | **MAPE** |
| **Prophet 1** | Prophet(interval\_width=0.95) | 17780957 | 4217 | 0.2816 |

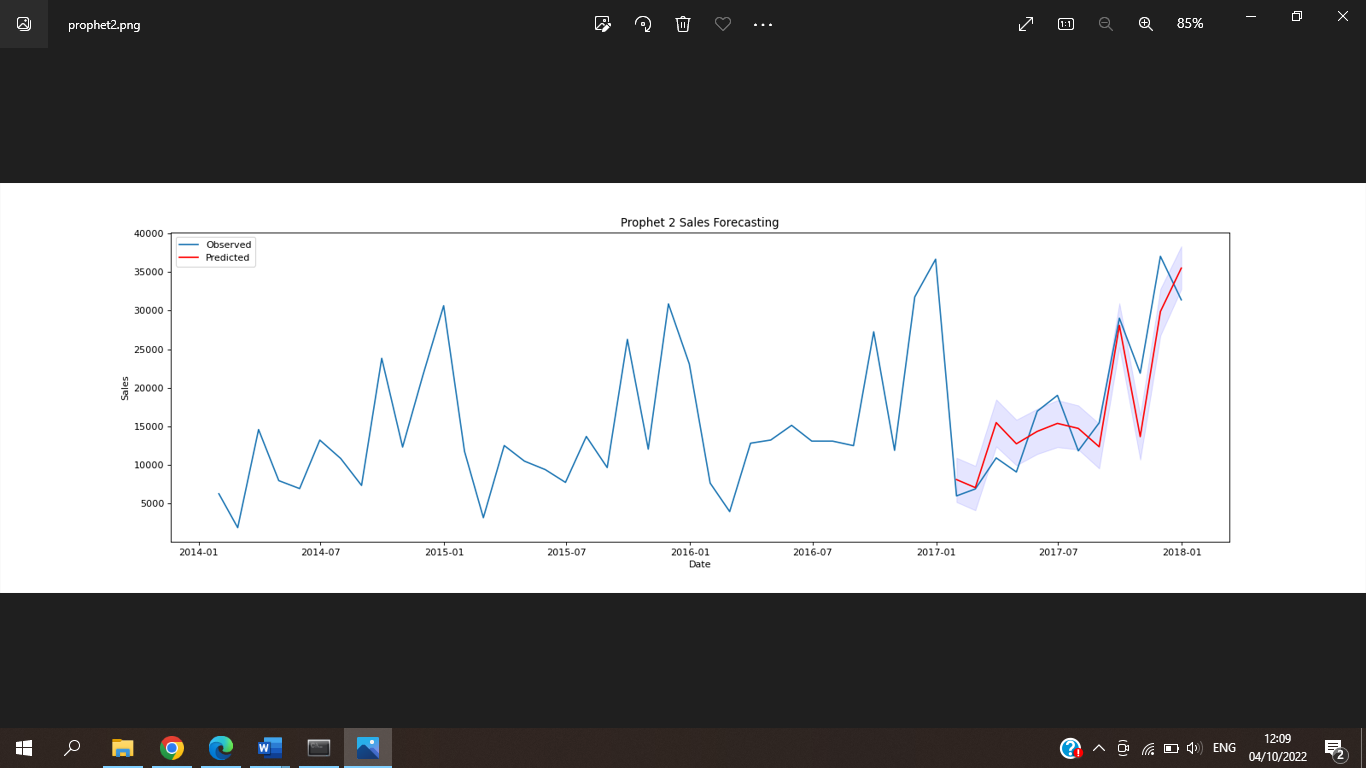


## Prophet Model 2

The second implemented Prophet model includes the effects of holidays and seasonality in its forecasts. The dates of eight American recurring holidays have been included into the model. Seasonalities are estimated using a partial Fourier sum. The number of terms in the partial sum (the order) is a parameter that determines how quickly the seasonality can change. The value of this parameter should be increased when the seasonality needs to fit higher-frequency changes, and generally be less smooth. Increasing the number of Fourier terms allows the seasonality to fit faster changing cycles, but can also lead to overfitting.

Different parameter values for the order of the Fourier sum were investigated (the “yearly\_seasonality” parameter), as well the effect of including or excluding holiday information in the model. Below are the best results obtained, and the parameter values that were used to obtain them.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Architecture** | **MSE** | **RMSE** | **MAPE** |
| **Prophet 2** | Prophet(interval\_width=0.95, yearly\_seasonality=15, holidays=holidays) | 17827865 | 4222 | 0.2274 |

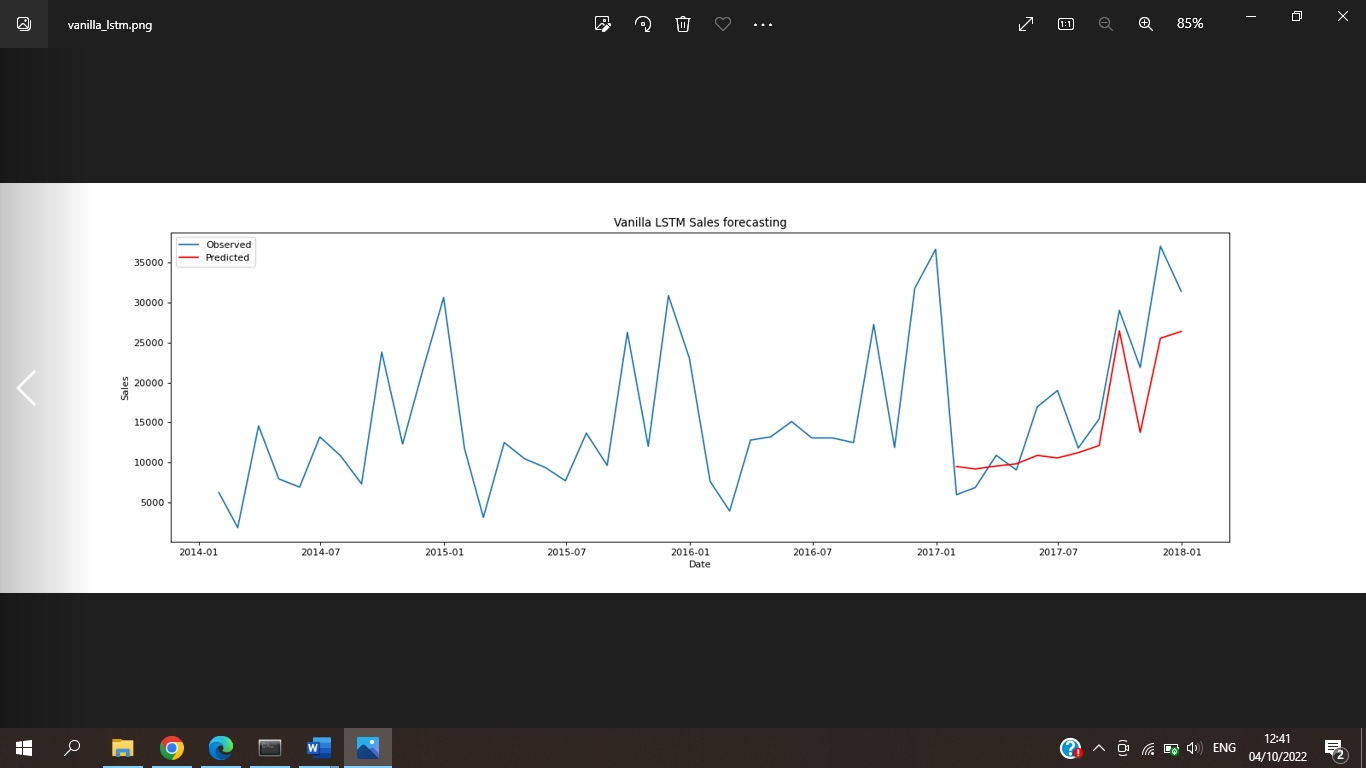


## Vanilla LSTM

Before building the LSTMs, some data pre-processing steps were needed. The data was scaled to between 0 and 1 using the MinMaxScaler from the sklearn library. Then the LSTMs were built using the Tensorflow Keras library. To define LSTM models with the Keras library, the time-series should be transformed to a supervised learning problem. This can be achieved using the previous observations (in this case, the previous 12 observations representing the past year) as the input, and the current observation as the output. This transformation was performed using the TimeseriesGenerator library.

The first implemented LSTM was a Vanilla LSTM, consisting of a single hidden LSTM layer and an output layer. The activation function used was Relu, and the optimizer was an Adam optimizer. Different parameter values for the number of nodes in the LSTM layer (the “units” parameter) were investigated, as well the effect of using dropout. Different parameter values for dropout were also investigated when it was used. Below are the best results obtained, and the parameter values that were used to obtain them.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Architecture** | **MSE** | **RMSE** | **MAPE** |
| **Vanilla LSTM** | (LSTM(units=32, activation='relu', dropout=0.1))  optimizer='adam' | 30808886 | 5551 | 0.2610 |

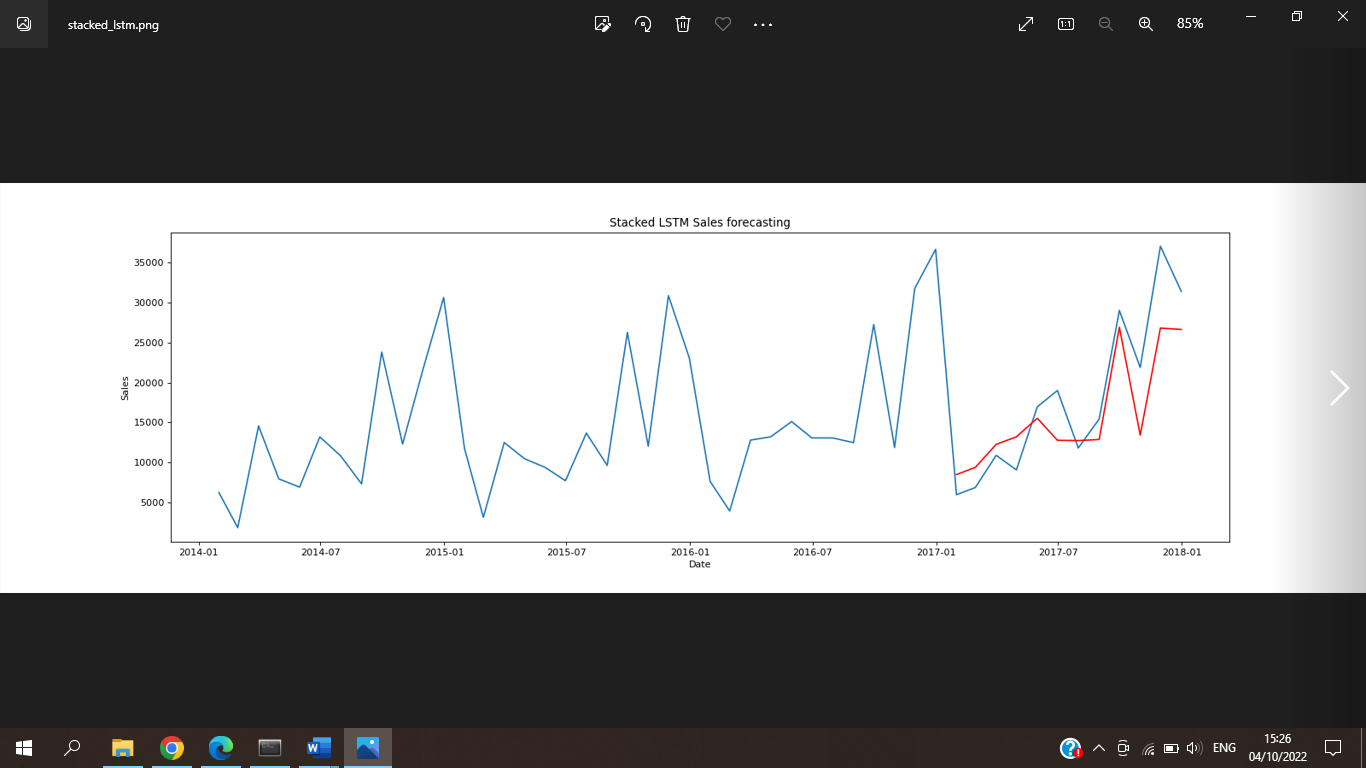


## Stacked LSTM

The second implemented LSTM was a Stacked LSTM, consisting of multiple hidden LSTM layers and an output layer. The activation function used was Relu, and the optimizer was an Adam optimizer.

The effect of changing the number of hidden layers was investigated, as well as different parameter values for the number of nodes in the LSTM layers (the “units” parameter). The effect of using dropout was also investigated, as well as different parameter values for dropout when it was used. Below are the best results obtained, and the parameter values that were used to obtain them.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Architecture** | **MSE** | **RMSE** | **MAPE** |
| **Stacked LSTM** | (LSTM(units=32, activation='relu'))  (LSTM(units=32, activation='relu'))  optimizer='adam' | 23642784 | 4862 | 0.2431 |



## Discussion

Below are the overall best results obtained from each model investigated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Architecture** | **MSE** | **RMSE** | **MAPE** |
| **Prophet 1** | Prophet(interval\_width=0.95) | 17780957 | 4217 | 0.2816 |
| **Prophet 2** | Prophet(interval\_width=0.95, yearly\_seasonality=15, holidays=holidays) | 17827865 | 4222 | 0.2274 |
| **Vanilla LSTM** | (LSTM(units=32, activation='relu', dropout=0.1))  optimizer='adam' | 30808886 | 5551 | 0.2610 |
| **Stacked LSTM** | (LSTM(units=32, activation='relu'))  (LSTM(units=32, activation='relu'))  optimizer='adam' | 23642784 | 4862 | 0.2431 |

Prophet 1 and Prophet 2 have almost the same MSE and RMSE, however Prophet 2 has a significantly lower MAPE compared to Prophet 1 (0.2274 and 0.2816 respectively). Therefore we can conclude that Prophet 2 outperformed Prophet 1, and that for the task of forecasting Furniture sales, including holiday information and setting the seasonality parameter to 15 improves the performance of the forecast.

The Stacked LSTM has a lower MSE, RMSE, and MAPE when compared to the Vanilla LSTM (MAPE vales of 0.2431 and 0.2610 respectively). Therefore we can conclude that a Stacked LSTM outperforms a Vanilla LSTM, and that for the task of forecasting Furniture sales, including more than one hidden layer in the LSTM improves the performance of the forecast. Both LSTMs performed best when they had 32 nodes in their hidden layers, so this may be the optimum number of units for this specific task. However, the vanilla LSTM performed best when a dropout rate of 0.1 was used, but the Stacked LSTM performed best when no dropout was used.

The overall best performing model for this task is Prophet 2. It outperformed the Prophet 1 model, the Vanilla LSTM, and the Stacked LSTM.