Machine Learning Engineer Nanodegree

Capstone Project

Francesco Deleo

June 19, 2020

I. Definition

Project Overview

How many Ben & Jerry's tubes will a store sell next month? Or Jack Daniel's BBQ sauce bottles in the upcoming summer? Forecasting demand is key for any business selling products and services. Without a solid idea of how sales numbers are going to look like, key business tasks such as managing inventory, predicting cash flows and planning for growth become difficult. In that sense, forecasting demand can help take more informed and intelligent business decisions[1]. Is company X expecting a boom in sales of products and services? Then, X's managers might consider hiring additional staff to successfully meet demand. Are the experts at company Y forecasting a shortfall in sales? Then it would be beneficial to start reducing expenses and reorient Y's marketing efforts.

In the specific case of product demand at physical retail stores, forecasting can be useful in order to better manage inventory, run seasonal offers/discount campaigns, reduce waste, set up appropriate service levels and so on. Accurate forecasts are therefore extremely valuable for companies in retail. Demand forecasting is a well active area of research, with companies often backing its efforts to advance forecasting theory and practice.

Inspiration for this project was found in the *M5 Forecasting – Accuracy*[2] Kaggle competition, where the relevant data was also retrieved. Specifically, the data describes hierarchical sales data from Walmart, the world's largest company by revenue, for 1,913 days. It covers stores in three US States (California, Texas, and Wisconsin) and includes item level, department, product categories, and store details. Although the original dataset is made of 3,049 unique

[1] Kolassa, S. & Siemsen, E. (2014). Demand Forecasting for Managers. New York: Business Expert Press.
[2] Available here: https://www.kaggle.com/c/m5-forecasting-accuracy/data

products across 10 different stores, this work will focus on the 50 most sold products belonging to one of the food departments in one, handpicked, specific store.

Problem Statement

As the Makridakis Open Forecasting Center[3] (MOFC) puts it: "A wrong weather forecast may result in you carrying around an umbrella on a sunny day, while inaccurate business forecasts could result in actual or opportunity losses". With the advancement of machine learning tools for forecasting, one might ask themselves whether these tools perform well in predicting future product demand. Specifically, the following problem will be investigated: "How effective are Recurrent Neural Networks in forecasting product demand compared to a Naive model?".
This work will therefore employ AWS Sagemaker DeepAR forecasting algorithm to assess Recurrent Neural Networks' ability in forecasting scalar time-series.

Metrics

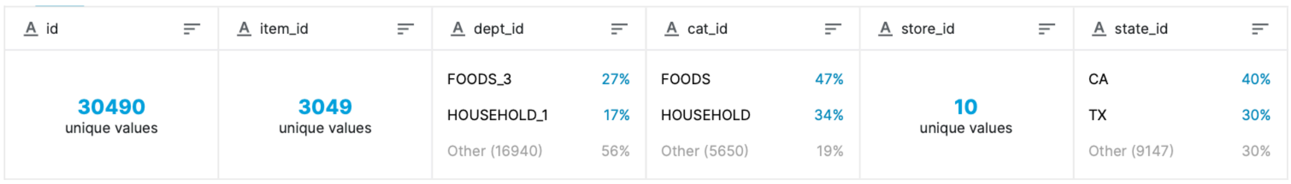
The DeepAR model accuracy is measured through the Mean Absolute Percentage Error (MAPE) and the Root Mean Squared Deviation/Error (RMSD/RMSE). These two metrics are universally accepted forecast error measurements[4]
The DeepAR model's accuracy will then be evaluated against the Benchmark (Naive) model's through the comparison of these metrics.

[3] MOFC (s.d.). M5 Forecasting – Accuracy. Source: https://www.kaggle.com/c/m5-forecasting-accuracy/data
[4] Hyndman, R. J., & Athanasopoulos, G. (2018). Forecasting: principles and practice. OTexts.

II. Analysis

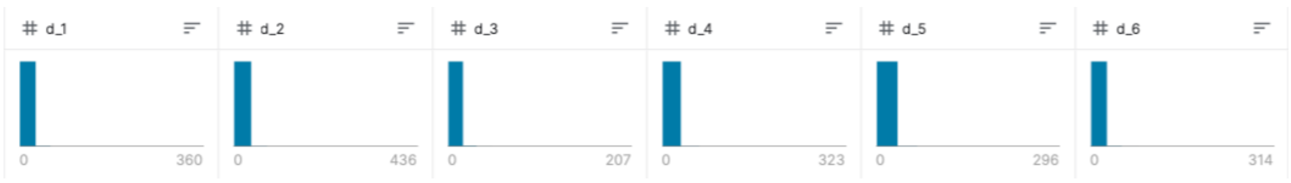
Data Exploration

As previously stated, the data is taken from the “M5 Forecasting – Accuracy” Kaggle competition and describes hierarchical sales data from Walmart, the world’s largest company by revenue, for 1,941 days (~ 5.3 years)



id	item_id	dept_id		cat_id		store_id	state_id	
30490 unique values	3049 unique values	FOODS_3	27%	FOODS	47%	10 unique values	CA	40%
		HOUSEHOLD_1	17%	HOUSEHOLD	34%		TX	30%
		Other (16940)	56%	Other (5650)	19%		Other (9147)	30%

Descriptive variables



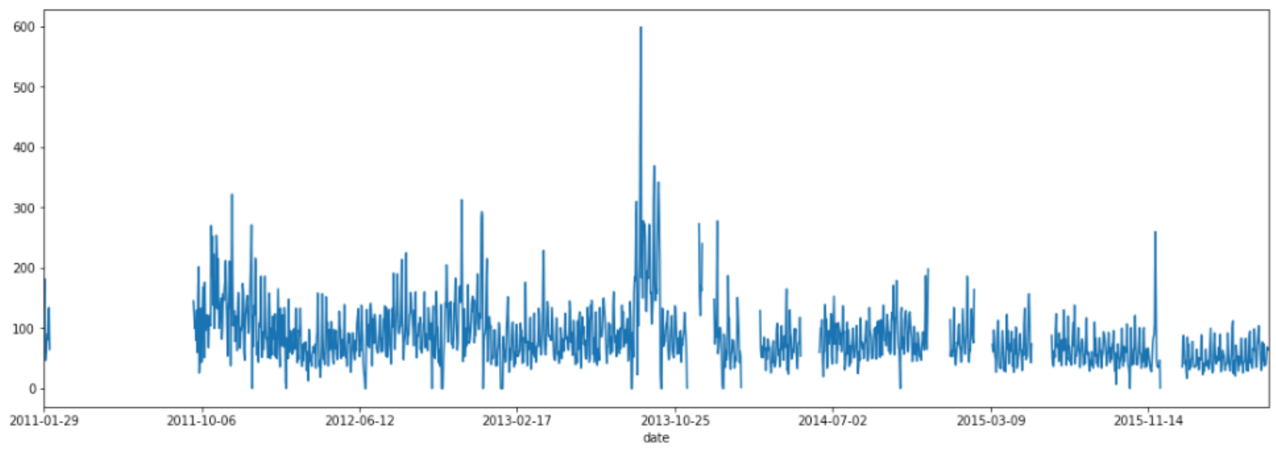
First 6 days (of 1941) – time-series columns.

The data covers stores in three US States (California, Texas, and Wisconsin) and includes item level, department, product categories, and store details. Although the original dataset is made of 3,049 unique products across 10 different stores, this work will focus on the products belonging to one of the food departments in one, handpicked, specific store.

It is important to note that the time-series data columns do not present any missing values. On the other hand, some products are characterized by several null values. The processing of these null values will be discussed in the following sections.

Exploratory Visualization

An example of a time-series for a specific product is shown below.



Example of time-series extracted from the dataset

Although the original dataset – as previously stated – does not present missing values, the time-series shown above presents some gaps. This is due to data preprocessing, whose steps will be explained in the relative section.

Algorithm and Techniques

This work employs AWS Sagemaker DeepAR forecasting algorithm to assess Recurrent Neural Networks' ability in forecasting scalar time-series. Specifically, the algorithm will be employed to forecast daily sales for the 28 days subsequent to the model training's time-window. The model hyperparameters will be tuned to find an optimal predictor among the hyperparameter ranges given to the tuner. The optimal predictor will then be deployed to extract predictions for the 28-days test period.

Benchmark

The DeepAR model will be evaluated against a Naive Forecasting benchmark model, which will try to forecast units sold for the 28 days in question based on the previous 28-days average.

As previously discussed, the comparison between the DeepAR model and the Naïve model will be based on the Mean Average Percentage Error (MAPE) and Root Mean Squared Error (RMSE) metrics.

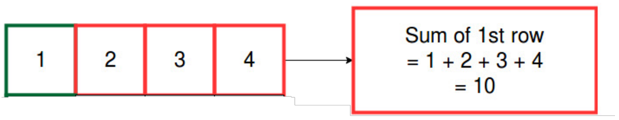
III. Methodology

Data Preprocessing

1st step: Dataset trimming

The data was first trimmed by selecting the products sold in a food department of a specific store. In our case, the FOOD_3 department and the CA_1 store were picked.

The resulting dataset was then manipulated in order to extract the 50 most sold products. To do this, a column-wise sum of the time-series values for each row was computed (as in the example below).



Example of column-wise sum

The resulting manipulated dataframe was then sorted to extract the first 50 products' indexes, which were used to subset the original dataframe.

2nd step: Creation of time-series objects and handling of null values

The DeepAR algorithm needs to be fed JSON-formatted data. To make this transformation process easier, series objects were extracted from the dataframe (1 row = 1 time-series) through the use of a helper function.

The helper function:

- receives the dataframe as input,
- selects the columns relevant for the time-series creation (the "d_#" columns),
- selects each row in a loop
- transposes each row and concatenates the timestamps (taken from the calendar.csv data)

Additionally, the helper function also deals with two issues:

- the need of having an accurate start date for each time-series, since the original dataset does not account for it. Some products, in fact, started selling way after the first day in the dataset (meaning that they were out of stock or simply not yet introduced in the store), therefore presenting several trailing null values at the beginning of their relative time-series, This issue needed to be dealt with in order to avoid *towards-zero bias* in the model. The data was processed accordingly by "cutting" those series that presented more than 14 trailing null values in their initial period. The reasoning behind selecting 14 as a threshold value is explained below.
- the need of differentiating between out-of-stock periods and actual lack of sales, also required in order to avoid *towards-zero bias.* The data was adjusted accordingly, based on the assumption that if no sales were registered for a product across 14 consecutive days, the product was to be considered either out of stock or not being sold throughout a certain period. The data was therefore processed in this fashion:

- If more than 14 trailing null values were found in a series, the "block" of 0s was transformed to *NaN* values.
- Otherwise, the null values were kept.

The transformation to *NaN* values was considered due to the DeepAR algorithm being able to handle them directly within the model in order to make more accurate predictions.

Last step: converting to JSON format and uploading to S3

The so-preprocessed time-series were then converted to the JSON format accepted by the DeepAR model (requiring a "start" key and a "target" key):

{"start": *earliest timestamp in the time-series*,
"target": *time-series values*}.

The JSON lines were then grouped together to form the training set (1,913 days) and the test set (1,941 days, the whole time-window, as required by DeepAR[5]). These were finally uploaded to S3 to make them accessible to the DeepAR estimator's container.

Implementation

A DeepAR estimator was instantiated and its hyperparameters tuned through a hyperparameter tuner. The tuner ranked the different models (based on different hyperparameters) through the optimization (minimization) of the Root Mean Squared Error. In fact, by providing a "test" data channel, DeepAR is able to calculate accuracy metrics (again, RMSE in our case) for the trained model by comparing it to the test data.

[5] Hudgeon, Doug, and Richard Nichol (2020). "Machine Learning for Business: Using Amazon SageMaker and Jupyter." Amazon. docs.aws.amazon.com/sagemaker/latest/dg/deepar.html.

Specifically, this is done by predicting the last 28 points of each time series in the test set and comparing this to the actual value of the time series.

The best model was then extracted from the hyperparameter-tuning job and deployed in order to extract predictions for the 28-day test period. The extractions were conducted through a JSON query as explained in the DeepAR documentation[6]. The prediction requests were then decoded in order to extract the prediction values and quantile data.

The results were first visualized through graphs, showing quantiles, the median prediction and how they compare with the actual data points. A result example will be shown in the relative section.

Following, the model was compared to the Benchmark Naïve Model through visualization and tables reporting the MAPE and RMSE metrics.

Refinement

The refinement process was conducted both in the implementation (which also required additional data preprocessing) and in the comparison with the benchmark model.
At first, I was running the DeepAR model on all the products from the department and store picked (around 800 products), but quickly realized there would've been more value in focusing in the most sold 50 products. I then adjusted for the *towards-zero bias* and the inaccuracy in the start date by addressing the null values blocks (both at the beginning and throughout the window period) in the data preprocessing. Finally, in the last iteration I tuned the model through a hyperparameter tuning job.

[6] Hudgeon, Doug, and Richard Nichol (2020). "Machine Learning for Business: Using Amazon SageMaker and Jupyter." Amazon. docs.aws.amazon.com/sagemaker/latest/dg/deepar.html.

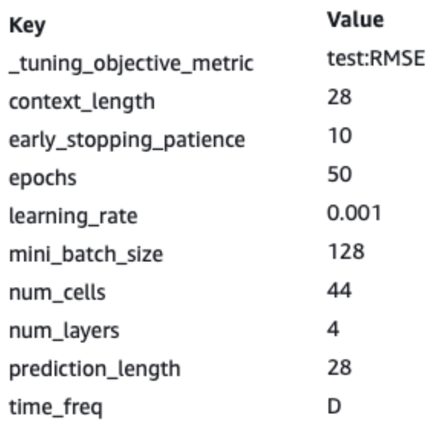
For the metric part, I was first counting on the only use of MAPE but realized that this metric would not be effective for those series reporting null values among the target values. MAPE requires in fact to have the target value at the denominator, making the metric not computable for those series reporting no product sold throughout one or more of the 28-days test period. Therefore, I included the RMSE in the analysis to make the comparison between the DeepAR model and the Benchmark Naive Model more reliable.

IV. Results

Model Evaluation and Validation

This section will report results for 1 product out of 50 as an example. All the results are available in the iPython Notebook in the GitHub repo.

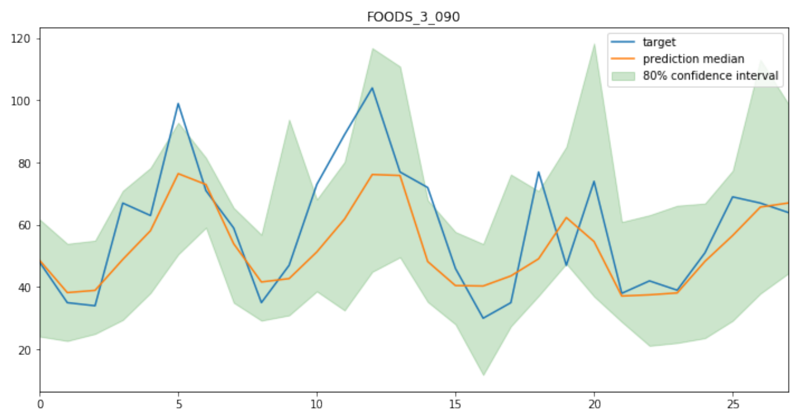
Through the hyperparameter tuning job, the model with the following parameters was selected.



Key	Value
_tuning_objective_metric	test:RMSE
context_length	28
early_stopping_patience	10
epochs	50
learning_rate	0.001
mini_batch_size	128
num_cells	44
num_layers	4
prediction_length	28
time_freq	D

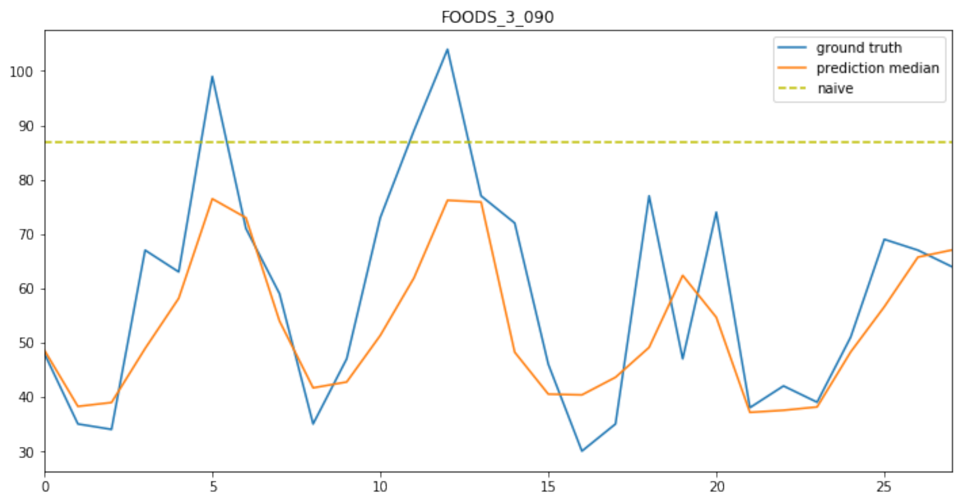
Optimal hyperparameters found by the tuning job

Results – Visualization: DeepAR with quantile data VS Actual (target)



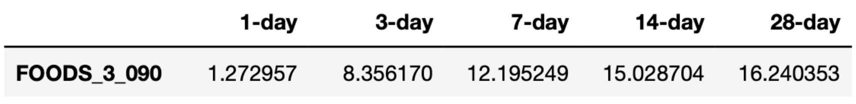
DeepAR vs Actual

Results – Visualization: DeepAR vs Naive vs Actual

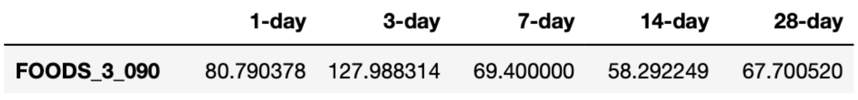


DeepAR vs Naive vs Actual

Results – MAPE: DeepAR vs Naive (product-wise)

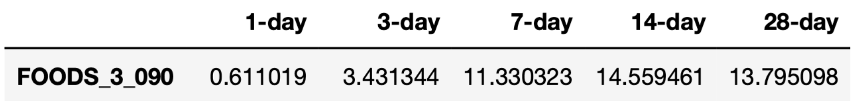


	1-day	3-day	7-day	14-day	28-day
FOODS_3_090	1.272957	8.356170	12.195249	15.028704	16.240353

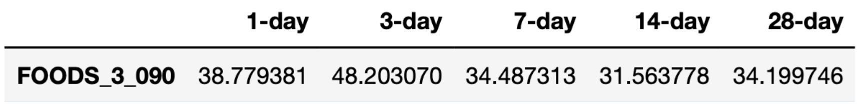


	1-day	3-day	7-day	14-day	28-day
FOODS_3_090	80.790378	127.988314	69.400000	58.292249	67.700520

Results – MRSE: DeepAR vs Naive (product-wise)

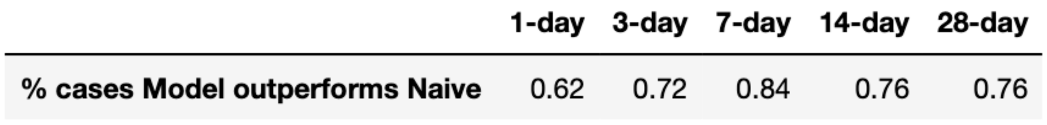


	1-day	3-day	7-day	14-day	28-day
FOODS_3_090	0.611019	3.431344	11.330323	14.559461	13.795098



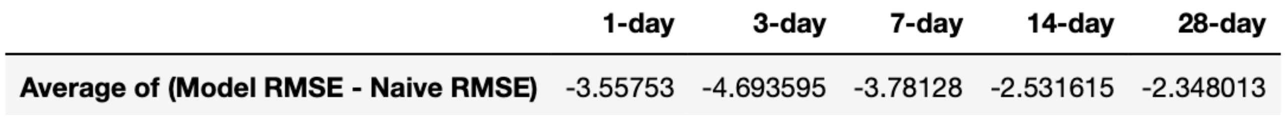
	1-day	3-day	7-day	14-day	28-day
FOODS_3_090	38.779381	48.203070	34.487313	31.563778	34.199746

Results – % cases in which the DeepAR outperforms the Naive Model, in terms of RMSE



	1-day	3-day	7-day	14-day	28-day
% cases Model outperforms Naive	0.62	0.72	0.84	0.76	0.76

Results – Magnitude of outperformance on average: DeepAR vs Naive Model, in terms of RMSE (Model RMSE – Naive RMSE)



	1-day	3-day	7-day	14-day	28-day
Average of (Model RMSE - Naive RMSE)	-3.55753	-4.693595	-3.78128	-2.531615	-2.348013

Through the analysis of the results in their entirety, the DeepAR model based on RNN networks seems to perform reasonably well in making accurate predictions for the test time-window, therefore generalizing well to unseen data. The parameters of the model seem appropriate, although the number of hidden layers (4) may be seen as too high. It is important to note that the hyperparameter tuning is based on 10 jobs only. A bigger number of jobs could bring to more optimal parameters. Nonetheless, the found model seems to be trustworthy.

Justification

The DeepAR model seems to generally outperform the Naive model, both in the short- and long-term.
If we look at the magnitude of the difference between the DeepAR RMSE and the Naive RMSE, it seems that the DeepAR algorithm performs comparatively better in the long run than in the short run.
Thanks to these comparisons, it can be stated that DeepAR is definitely more effective in forecasting product demand compared to a Naive model. "How effective are Recurrent Neural Networks in forecasting product demand compared to a Naive model?"

V. Conclusion

Reflection

This project was taken on in order to investigate the effectiveness of Recurrent Neural Networks in predicting product demand. Data came from the Kaggle Competition M5 Forecast – Accuracy. The dataset was first trimmed and the resulting data preprocessed in order to account both for inaccuracies in the starting date of each product's time series and towards-zero bias. The data was then transformed into JSON format and uploaded to AWS S3 service in order to be accessed from the Sagemaker estimator. To this generic estimator was passed the DeepAR model through a specific container. Further steps included hyperparameter tuning through a Sagemaker tuner. Both the training channel and the test channel were accessible to the model in order for the tuning job to compute metrics for the test time-period based on RMSE. The resulting best model was then deployed to extract the specific predictions for the products analyzed. Through both visualization and the analysis of the metrics discussed above, the DeepAR algorithm proved more effective in forecasting product demand compared to a Naive model.

Improvement

This project can be further improved on different areas. First of all, it would be beneficial to a run a more extensive hyperparameter tuning job in order to find more optimal hyperparameters. Secondly, the model could be trained on a bigger dataset, focusing not only on a number of products from a specific store's department, but on more stores and more department. This would be valuable to see if the DeepAR proves to be effective also when trained on many time-series that are somewhat related to each other, but not entirely. Finally, the possibility of including categorical features (e.g. type of product) and dynamic features (e.g. promotion days) could be explored in order to try and improve further the accuracy of the model.