Udacity

Machine Learning Engineer Nanodegree

Capstone Project

Catherine Sai
April 12th, 2020

I. Definition

The Project Definition can be found in the document “Proposal_Step_1.pdf”. The Proposal has been updated and thus I will refrain from repeating this information here.

II. Analysis

Data Exploration

A) The initial data of this project looked like the following.

train.csv:

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
0	1	5	2015-07-31	5263	555	1	1	0	1
1	2	5	2015-07-31	6064	625	1	1	0	1
2	3	5	2015-07-31	8314	821	1	1	0	1
3	4	5	2015-07-31	13995	1498	1	1	0	1
4	5	5	2015-07-31	4822	559	1	1	0	1

store.csv:

	Store	StoreType	Assortment	CompetitionDistance	CompetitionOpenSinceMonth	CompetitionOpenSinceYear	Promo2	Promo2SinceWeek	Promo2SinceYear
0	1	c	a	1270.0	9.0	2008.0	0	NaN	NaN
1	2	a	a	570.0	11.0	2007.0	1	13.0	2010.0
2	3	a	a	14130.0	12.0	2006.0	1	14.0	2011.0
3	4	c	c	620.0	9.0	2009.0	0	NaN	NaN
4	5	a	a	29910.0	4.0	2015.0	0	NaN	NaN

B) Through exploration the following insights concerning the data were made in the section ‘Data Understanding’ and addressed in the sections ‘Data Preparation” and “Feature Engineering”:

Insights:

1) there seem to be no missing features in these columns

2) the "Date" column needs to be transformed into a timeseries object

3) the "StateHoliday" column should also be encoded to numerical

4) from this it is first look at the data, it seems that "Customer" column and "Sales" column are highly correlated. But the Number of Customers is not known for a prediction. Thus I would remove this information before starting the prediction. Maybe it can help though to find a correlation between Customers and Promotions.

5) on "DayOfWeek" == 7 the store seems to be closed always (since this is Sundays). It's worth considering to exclude this data from the prediction

6) "DayOfWeek" == 1 (Monday) seems to be a strong sales day

7) otherwise the sales over the week seem to be quite evenly distributed (surprising that Saturday is not a stronger sales day)

8) there seem to be quite a few open Sundays with high sales also. Thus it would be unwise to delete all Sundays but maybe this correlation should be extracted more clearly for the model by creating a binary feature like "Open_sunday"

9) the store_df has quite some missing data

10) the missings in "Promo2SinceWeek", "Promo2SinceYear" and "PromoInterval" are due to the fact that Promo2 is not applicable at these stores. Thus instead of NaN I will fill these fields with zero.

11) "Promo2SinceWeek", "Promo2SinceYear" should be converted into a datetime object. When combining the two df I will create a new column "Promo_2_active" instead of the 3 columns "Promo2SinceWeek", "Promo2SinceYear" and "PromoInterval"

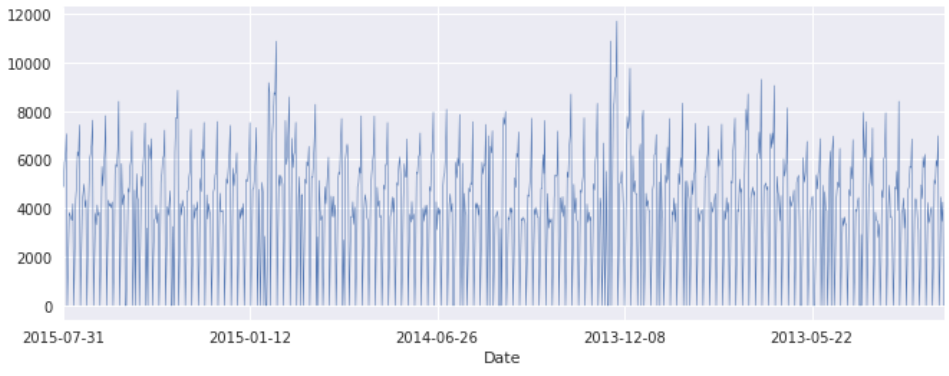
12) the missinings in "CompetitionOpenSinceMonth" and "CompetitionOpenSinceYear" cannot be imputed and it wouldn't make sense to fill these with zeros. I don't see why it should make a difference how long a competitor has been in the area, the distance to the competition seems much more important. Thus I plan to delete these two columns.

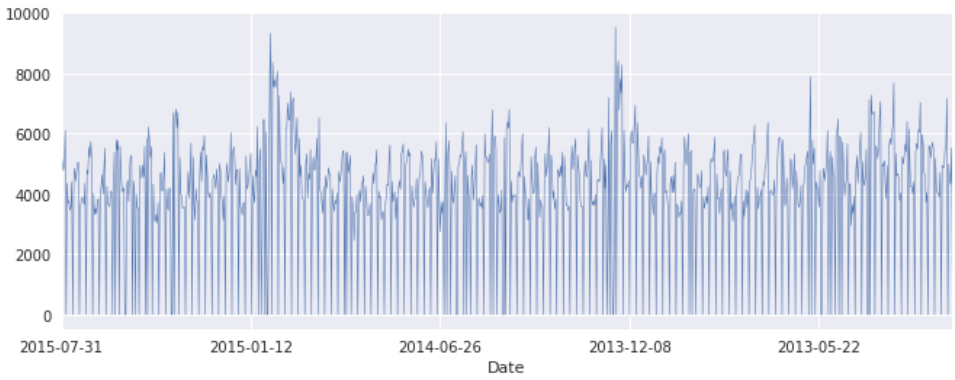
13) "StoreType", "Assortment" and "PromoInterval" need to be converted to numerical features

14) For the 3 rows with a missing "CompetitionDistance" I assume that these stores have no competition in close approximation, thus I plan to impute the values as 99999 (which is representing a competition far away/ nonexistent)

Exploratory Visualization

A) Example time series visualizations for store no. 1 and store no.5 (randomly chosen):



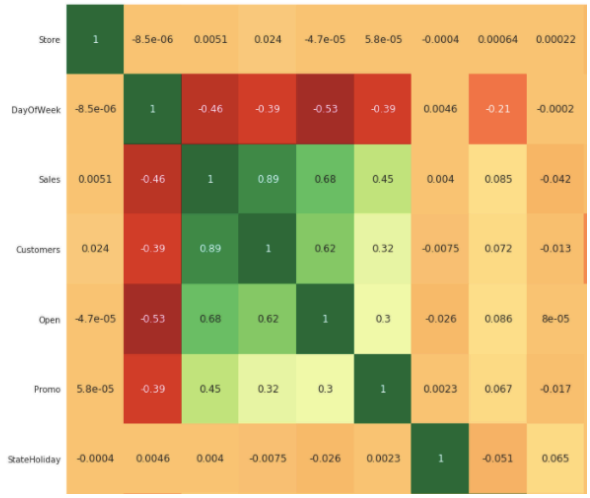


Insight:

There seems to be no upward or downward trend for these stores and no clear seasonality either. The only thing that strikes are the peaks around Christmas time.

B) Extract of a correlation matrix (between the target (sales) and all features of the final prepared df):

```
Sales        1.000000
Customers    0.894711
Open         0.678472
Name: Sales, dtype: float64
```



Insights:

‘Customer’ and ‘Open’ have the highest correlation with Sales. The Customer column doesn't seem to show insight which the sales column doesn't show directly too. Since the number of customers is not available for the forecast, this column was deleted from the data frame.

Algorithms and Techniques

For this Project the following two Algorithms were used:

A) Simple Exponential Smoothing

The most basic way to predict a time series for e.g. the next year is probably to just take the last years data and copy it to the new year. The second most basic I would say is to build the mean of last year’s data and predict this for the coming year. The second approach clearly averages the prediction compared to the first one. Now people could argue that historical data of e.g. 11 month ago is not as relevant as historical data of 1 month ago for the prediction of the coming month. This is an issue Simple Exponential Smoothing deals with, which to me is probably the third most basic time series prediction approach.

SES, Simple Exponential Smoothing (synonym: Single Exponential Smoothing) differentiates to the average only by one parameter – the alpha setting, which stands for the smoothing of the historical data. This parameter can be set between 0 and 1 and influences the weight of past data on the prediction. An alpha of 0 is equal to the average approach as it will weigh all historical data the same way and thus smooth the prediction. The bigger the alpha the more the weight transfers to the recent historical data, still taking all data into account but the older the data the less influence it has on the prediction.

SES is not able to recognize trends or seasonality.

Ressource: https://machinelearningmastery.com/exponential-smoothing-for-time-series-forecasting-in-python/

B) AWS Sagemaker DeepAR

As I find the AWS documentation on the basic functionality of the algorithm quite clear, I will copy the most important parts of that documentation in here.

“The Amazon SageMaker DeepAR forecasting algorithm is a supervised learning algorithm for forecasting scalar (one-dimensional) time series using recurrent neural networks (RNN). Classical forecasting methods, such as autoregressive integrated moving average (ARIMA) or exponential smoothing (ETS), fit a single model to each individual time series. They then use that model to extrapolate the time series into the future.

In many applications, however, you have many similar time series across a set of cross-sectional units. For example, you might have time series groupings for demand for different products, server loads, and requests for webpages. For this type of application, you can benefit from training a single model jointly over all of the time series. DeepAR takes this approach. When your dataset contains hundreds of related time series, DeepAR outperforms the standard ARIMA and ETS methods. You can also use the trained model to generate forecasts for new time series that are similar to the ones it has been trained on.”

Ressource: https://docs.aws.amazon.com/sagemaker/latest/dg/deepar.html

“During training, DeepAR accepts a training dataset and an optional test dataset. It uses the test dataset to evaluate the trained model. In general, the datasets don't have to contain the same set of time series. You can use a model trained on a given training set to generate forecasts for the future of the time series in the training set, and for other time series. Both the training and the test datasets consist of one or, preferably, more target time series. Each target time series can optionally be associated with a vector of feature time series and a vector of categorical features.“

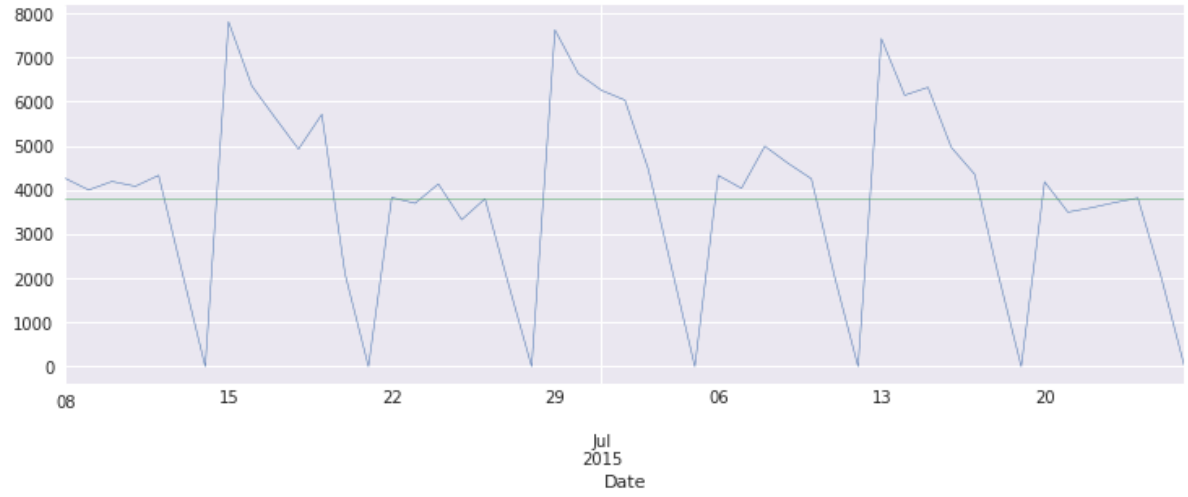
Ressource: https://docs.aws.amazon.com/sagemaker/latest/dg/deepar_how-it-works.html

For further (more detailed information) please see the Resource links.

Benchmark

As a benchmark for this project I used a simple exponential smoothing model which did not take features into consideration but only had past sales per store as input. The simple exponential smoothing is a basic algorithm for time series forecasting and relatively easy to implement. I chose an alpha parameter of 0.1 which makes the prediction very close to just the mean of the past. Therefore this is a great baseline for further development. If another models prediction is not better than a simple exponential smoothing it is definitely not worth the effort. Through the RMSE and RMSPE the results of this Benchmark is directly comparable with the DeepAR model.

Below you can see the SES prediction (green) compared to the actual sales (blue) for one store as example. As you can see this simple prediction seems to capture the average sales quite good, but doesn't account for the demand fluctuation during the weeks. The actual demand is much higher every second week and usually has a spike at the beginning of the week. The simple exponential smoothing only predicts the same amount for each week and doesn't take trend, seasonality or other patterns into consideration.



For the prediction of the next 49 days, the result of the SES was the following:

The absolute mean error (RMSE) is about 2996.0.

The relative (percentage) mean error (RMSPE) is about 28.0%.

If the data is resampled to 7 weeks however (instead of days), the RMSPE looks much better:

The absolute mean error (RMSE) is about 19921.0.

The relative (percentage) mean error (RMSPE) is about 7.0%.

It seems that the sum of demand over the 7 weeks prediction period is very close to the sum of actual demand over 7 weeks. Considering this use case I guess it is more common for a manager to order on a two-weekly or weekly basis than on a daily basis. So this result might actually be kind of good for this use case.

III. Methodology

Data Preprocessing

A) All the insights discovered in Data Exploration (I will not repeat them here) were addressed during data preprocessing. The final preprocessed data frame looked like this:

	Store	DayOfWeek	Date	Sales	Open	Promo	StateHoliday	SchoolHoliday	ID	StoreType	Assortment	CompetitionDistance	Promo_2_active	Open_sunday
0	1	5	2015-07-31	5263	1	1	1	1	0	2	0	1270	0	0
1	2	5	2015-07-31	6064	1	1	1	1	1	0	0	570	1	0
2	3	5	2015-07-31	8314	1	1	1	1	2	0	0	14130	1	0
3	4	5	2015-07-31	13995	1	1	1	1	3	2	2	620	0	0
4	5	5	2015-07-31	4822	1	1	1	1	4	0	0	29910	0	0

B) The DeepAR Model requires additional steps of Data Preprocessing. Therefore the above df was further changed to meet the required format. My prepared data frame held each datapoint as a new row. So for all 1115 stores the sales of day x had their own row and then the sales of the next day where the next 1115 rows, etc. This first of all had to be changed, so each store had one line of all historical information (time series format). For the DeepAR the data frame also needed to be changed into JSON-Lines. How exactly the data needed to look is shown below.

INPUT FOR TRAINING

{ "start": "2009-11-01 ", "target": [4.0, 10.0, 15.0,19.0] , “cat“: [0] }, “dynamic_feat": [8, 20, 30, 38] }
{ "start": "1999-01-30", "target": [2.0, 5.0, 7.0, 10.0], “cat“: [1] }, “dynamic_feat": [4, 10, 15, 19] }

<u>Setup to be aware of:</u>
Each time series starts in a new line!

Categories:
When you model different categories of similar timeseries (like in this case: predicting sales for different stores of the same company) you need to include the „cat“ attribute in the json-object which indicates the different categories. Be aware, that this field has to contain integers starting at zero. So if you have strings (like categories: „jeans“, „jackets“) you have to encode them to

integers. In my case I had store numbers from 1 to 1115 which I had to reduce by once each in order to have numbers starting from zero as needed by the algorithm!

Dynamic Features:
You want to include all the information at hand!

INPUT FOR ENDPOINT PREDICTING

Below is an example of what a JSON query to a DeepAR model endpoint might look like.

{ "instances": [
{ "start": "2009-11-01 ", "target": [4.0, 10.0] , “cat“: [0] }, “dynamic_feat": [8, 20, 30, 38] }
{ "start": "1999-01-30", "target": [2.0, 5.0], “cat“: [1] }, “dynamic_feat": [4, 10, 15, 19] }],
"configuration": {
"num_samples": 50,
"output_types": ["quantiles"],
"quantiles": ["0.5", "0.9"] } }

Setup to be aware of:
First, all time series are listed under instances, than at the end of the file once the configuration is handed over!

INPUT FOR BATCH TRANSFORM PREDICTING

{ "start": "2009-11-01 ", "target": [4.0, 10.0] , “cat“: [0] }, “dynamic_feat": [8, 20, 30, 38] }
{ "start": "1999-01-30", "target": [2.0, 5.0], “cat“: [1] }, “dynamic_feat": [4, 10, 15, 19] }

Setup to be aware of:
For Batch Transform the format is almost the same as for training.
The data that is passed in includes the historical data of the to be predicted time series (in the example above two measures) as well as the features for the prediction period.
The difference to the training data is, that the target values are only included for the historical data, not for the prediction period. For Training the model, the target values were included for historical and prediction period. The features however are included for the whole timeframe. Thus the features list is longer than the target list (e.g. above 4 figures).

See DeepAR Documentation for additional info:
https://docs.aws.amazon.com/sagemaker/latest/dg/deepar-in-formats.html

Implementation

Below is an overview of my implementation process:

1. Data Understanding

1.1 Load in the raw data "train.csv" and "store.csv"

1.2 Getting a feeling of the data

2. Data Preparation

2.1 Cleaning

2.2 Transformation

3. Feature Engineering

3.1 Combine Features

SES

4. Modelling the benchmark model (SES)

4.1 Loading the prepared data (from 4.1)

4.2 Predict next 49 days through Simple Exponential Smoothing without considering features (only target column)

5. Evaluation of the benchmark model (SES)

5.1 Combing Prediction- and Test-Data to an evaluation df

5.2 Visual Evaluation

5.3 Examination of results resampled to 7 weeks (instead of 49days)

5.4 Scores - calculating the prediction quality scores defined in the Project Proposal

DeepAR

6. Modelling the DeepAR Model

6.1 Loading the prepared data (from 4.1)

6.2 Prepare df for the DeepAR Algorithm

6.3 Creating training and test sets

6.4 Formatting data as JSON files and uploading to S3

6.5 Instantiating and training a DeepAR estimator

7. Evaluation & Deployment of the DeepAR Model

7.1 Prepare input data for Batch Transform Job

7.2 Apply the estimator with Batch Transform on the input data

7.3 Combing Prediction- and Test-Data to an evaluation df

7.4 Visual Evaluation

7.5 Examination of results resampled to 7 weeks (instead of 49days)

7.6 Scores - calculating the prediction quality scores defined in the Project Proposal

Refinement

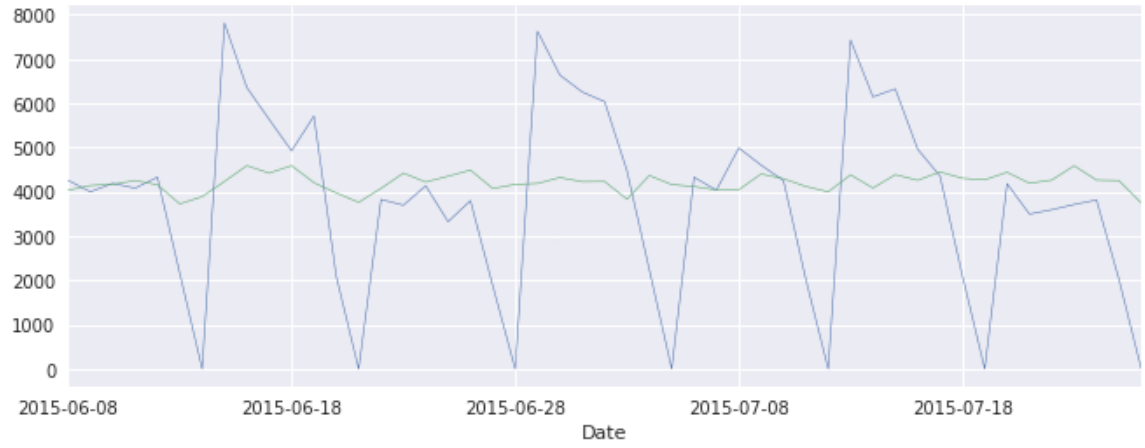
Initially the DeepAR prediction was tried by endpoint predictor but the amount and complexity (rows and columns) of data was much too big for the prediction task to be executed this way. Therefore, I switched to a Batch Transform Prediction approach which was tricky because the input data for the Batch Transform is not equal to the input data for the endpoint prediction. Consequently, after setting up a DeepAR Batch Transform Job, I also had to change my input data to the required format and store it in s3. Besides this, once the model was running, it was not refined due to the project deadline.

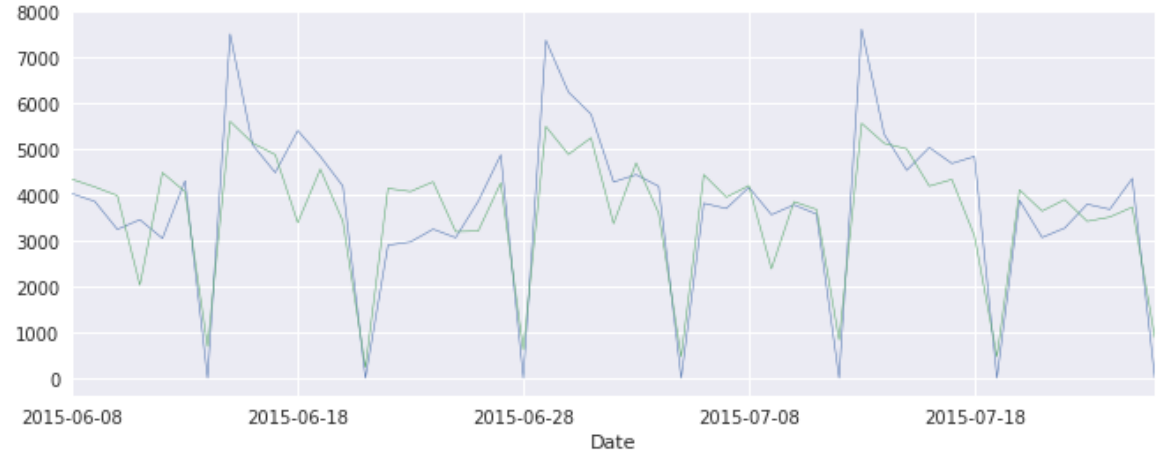
IV. Results

Model Evaluation and Validation

In this case the model was trained with the same time series that needed to be predicted. It is possible to use the trained model for other time series but there was no other similar drug store time series available to test this and this was out of scope for this project as well.

Below you can see two examples of the DeepAR prediction (green) compared to the actual sales (blue) for one store.





The two randomly chosen examples seem very interesting to me. The first one (for store no.5) is similar to my benchmark model with an almost linear prediction that seems to do a decent job in the mean but doesn't really predict the fluctuation of the stores demand. The second visualization however seems astonishingly precise to me. The prediction doesn't reach the highest peaks on Mondays but it seemed to have recognized the overall idea of every second week having higher demand and Sundays being without sales usually. This second prediction is much better than the benchmark model.

<u>For the prediction of the next 49 days, the result of the DeepAR was the following:</u>

The absolute mean error (RMSE) is about 1544.0.

The relative (percentage) mean error (RMSPE) is about 23.0%.

<u>If the data is resampled to 7 weeks however (instead of days), the RMSPE looks much better:</u>

The absolute mean error (RMSE) is about 25012.0.

The relative (percentage) mean error (RMSPE) is about 9.0%.

Justification

Comparing the results with the benchmark model, it is clear, that on a daily basis forecast, the DeepAr prediction is much better. The RMSE is only about half the size of the benchmark model! The difference in the RMSPE is only 5% between the models though. This is because for calculating this score we divide by the actual value and in order to due so without error I had to filter all the days with actual sales zero out of the RMSPE calculation. Especially at these days the DeepAR was very good at recognizing the reoccurring pattern of zero sales on regular Sundays while the benchmark model predicted the same for each day.

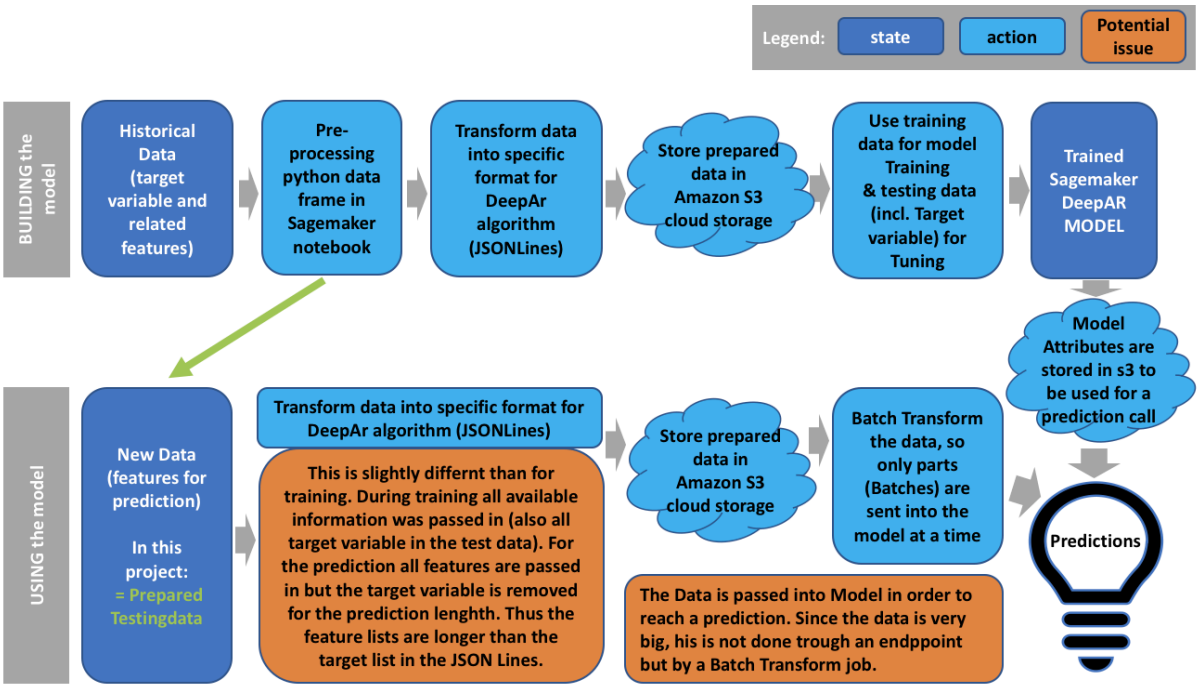
As I don't know what the drug store is currently suing to predict it's sales, I can't tell if the DeepAR outcome is good enough for the real world but for this project (which had the main goal to learn how to use an AWS DeepAR algorithm) it justifies all set criteria.

V. Conclusion

Free-Form Visualization

In this section, I want to present a visualization of my own making which hopefully helps others to quickly understand how the prediction with DeepAR works, as it took me a little to wrap my head around it.

DeepAR application in this project:



Reflection

What is a surprise to me, is, that on prediction length of one week (combined 7 day predictions to one week prediction) the simple benchmark model seems to be more accurate.

Thus my final brief recommendation at this point would be to use the DeepAR for exact predictions per day and the benchmark if weekly forecasts are sufficient.

Improvement

As many data science projects and projects in general also I hit a point where had to focus on getting through with the project and leave some stones along the road unturned in order to finish on time. Thus I have quite a few ideas left as to how this prediction could be improved:

- Include more data (online data sources, etc.) to create more useful features
- Apply a dimensionality reduction (like PCA) in order to present the Model with better prepared data
- Split up the data and build a model for each cluster: building subgroups of stores that seems to have a similar demand pattern and training a model for each of these subgroups could lead to an improvement over the current situation where one model is trained with all stores.
- Apply Hyperparameter Tuning for the Model