Forecasting Cryptocurrency Prices

Capstone Project Report

Udacity Machine Learning Engineer Nanodegree

April 2021
Lluc Cardoner Campi

Contents

Introduction

This report adds together the work done in the final Capstone Project of the Udacity Machine Learning Engineer Nanodegree[1] program. All the working material is available in the Capstone Project github repository[2].

The program

I have been working as a Data Scientist for the past three years. What I have realised during this time is that the machine learning pipeline can be divided in three parts:

- Data warehousing: in this part the raw data is collected, processed and stored. This is done by the Data Engineers.
- Data modeling: here is where Data Scientists use the data to create machine learning models.
- Model deployment: once a model has been trained it is time to deploy the model to production to make inferences for incoming requests. This is the work of the Machine Learning Engineers that know how to package the models in docker containers and serve them in servers through APIs.

This division might not always be clear since one could find himself doing more than one of those things. This usually happens when companies start creating a machine learning team or the team is small. If the company grows, then the team could be divided and specialized on the three points mentioned before.

During my experience as a Data Scientist, I have realised that I lack the tools and the knowledge to build from zero everything that is needed to serve a model. That is why I chose to do the Udacity program. This program taught me how to package and deploy models in a production environment in an easy way using the SageMaker[3] cloud service of Amazon Web Services[4] (AWS) . It also showed me some machine learning use cases that I had no experience with or that I only knew the theory.

The project

This is the final project of the program. The main goal is to “leverage what you’ve learned throughout the Nanodegree program to solve a problem of your choice by applying machine learning algorithms and techniques”. The topic I have chosen for this project is forecasting cryptocurrency prices. Recently, I have been in touch with this field and I have seen an opportunity to apply machine learning. Also, this project would give me the opportunity to work and learn on time series forecasting for the first time and to apply the knowledge learnt on the program case studies.

---

[1] https://www.udacity.com/course/machine-learning-engineer-nanodegree--nd009t
[2] https://github.com/lluccardoner/udacity-machine-learning-engineer-nanodegree
[3] https://aws.amazon.com/sagemaker/
[4] https://aws.amazon.com

Domain Background

A cryptocurrency is a digital asset designed to work as a medium of exchange wherein individual coin ownership records are stored in a ledger existing in a form of computerized database using strong cryptography to secure transaction records, to control the creation of additional coins, and to verify the transfer of coin ownership. Cryptocurrencies typically use decentralized control as opposed to centralized digital currency and central banking systems[5].

Two popular model used for time series forecasting is the ARIMA[6] (Auto Regressive Integrated Moving Average) and exponential smoothing[7] (ETS). Other work on this topic has used LSTM models [1][2][4][5], LightGBM [3] or XGBoost [4].

Problem statement

Cryptocurrency prices (how much USD dollars is one unit) are changing every minute and are usually very volatile. People invest in cryptocurrencies by buying and selling afterwards when they have benefits. The problem is knowing what will be the price of the cryptocurrency in the future. Time series forecasting models could be used to make predictions based on past data to help make decisions on how to invest.

As a solution, a forecasting machine learning algorithm will be trained to be able to predict the price of cryptocurrencies in the near future based on past data. Predictions can be later compared to real values to evaluate the model.

The model proposed for predicting the future price of a cryptocurrency is the SageMaker DeepAR[8] Forecasting Algorithm. This is an autoregressive neural network based model. As a baseline, a linear learner will be also trained to compare results.

[5] https://en.wikipedia.org/wiki/Cryptocurrency
[6] https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average
[7] https://en.wikipedia.org/wiki/Exponential_smoothing
[8] https://docs.aws.amazon.com/sagemaker/latest/dg/deepar.html

Methodology

Dataset

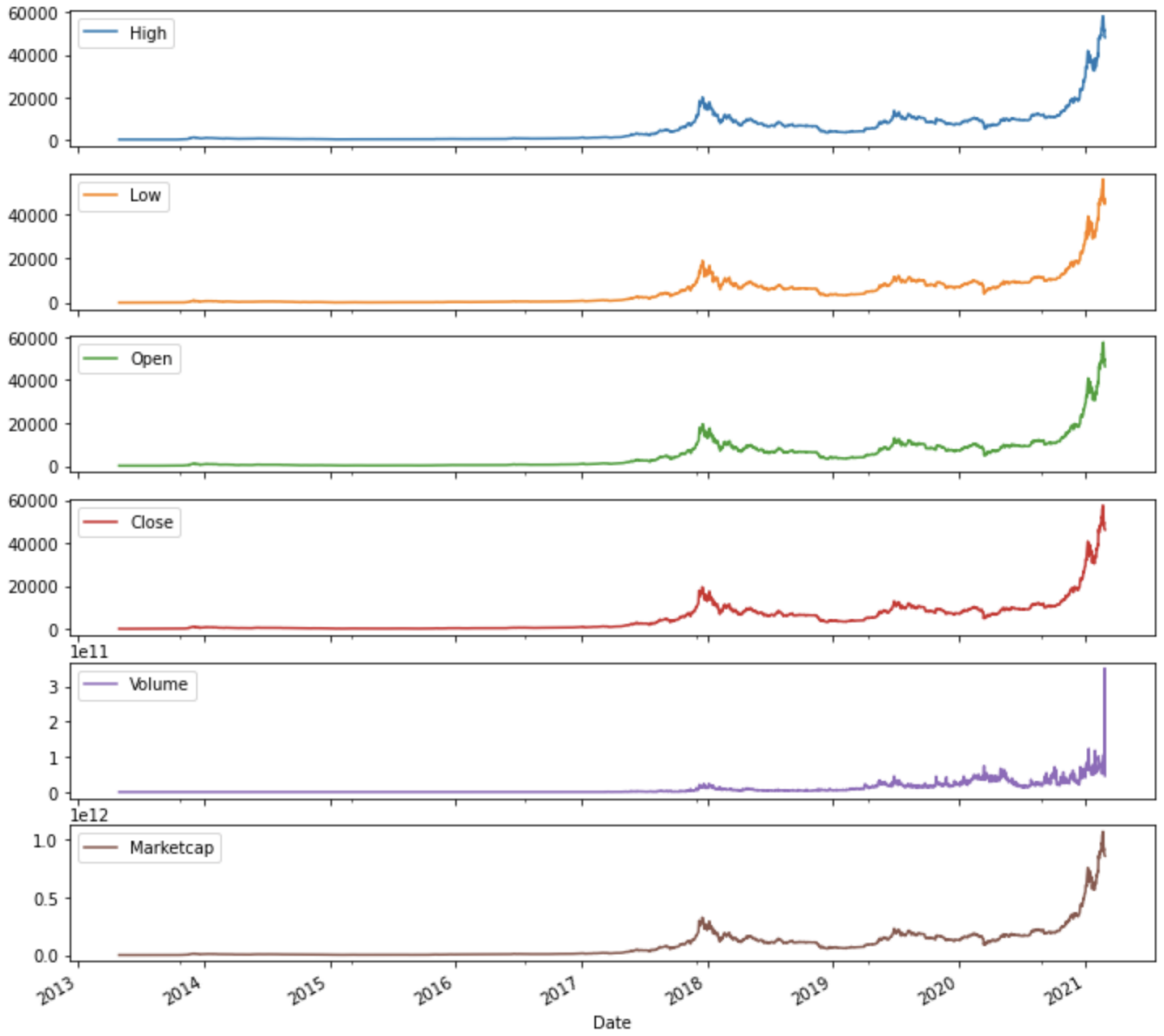
For the dataset, historical cryptocurrency price data[9] from Kaggle will be used. The dataset contains the daily data for 23 different cryptocurrencies.

The columns available are:

- SNo: index
- Name: cryptocurrency name (e.g Bitcoin)
- Symbol: cryptocurrency symbol (e.g BTC)
- Date: timestamp, each row has the values for one day
- High: highest price achieved of the day
- Low: lowest price achieved of the day
- Open: opening price of the day
- Close: closing price of the day
- Volume: volume of transactions on the given day
- Marketcap : Market capitalization in USD

The target time series will be the closing price of the day. For this project we will use bitcoins' available data from 2013-04-29 to 2021-02-27. In this period of 2861 days, the bitcoin has a minimum closing price of 68.431 USD and a maximum of 57539.94 USD.

[9] https://www.kaggle.com/sudalairajkumar/cryptocurrencypricehistory



If we take a closer look, the data looks like this:

Baseline

A linear model is chosen to be the baseline to be compared to the DeepAR model results. The model used will be the Linear Learner[10] available in SageMaker.

The features used to train the model are the following:

- Day of the week
- Day of the month
- Day of the year
- The lagged values of the closing price for 1, 2 and 3 weeks

These features have been chosen with the intention to replicate what the DeepAR model does internally. This will be explained in the following section.

The data is split in train and test sets. Since this is time series data, we will use the first 60% (1717) for training, the next 20% (572) for validation and the last 20% (573) for testing.

32 linear regressors are trained in parallel over 100 epochs with mini batches of 100 samples. The best model is found with the following hyper-parameters:

- Optimizer: adam
- Learning rate: 0.1
- L1 regularization: 0.0
- L2 regularization: 0.0001
- The number of steps between decreases of the learning rate: 10
- The decrease of learning rate by a factor of: 0.99
- The minimum learning rate: 0.00001

Model

The model chosen for forecasting cryptocurrency prices is the DeepAR[11] model from AWS SageMaker. It is a supervised learning algorithm for forecasting scalar (one-dimensional) time series using recurrent neural networks (RNN).

Time series

To train the DeepAR model, time series need to be created. The target time series will be created using bitcoin's daily closing price data. Two different approaches have been taken:

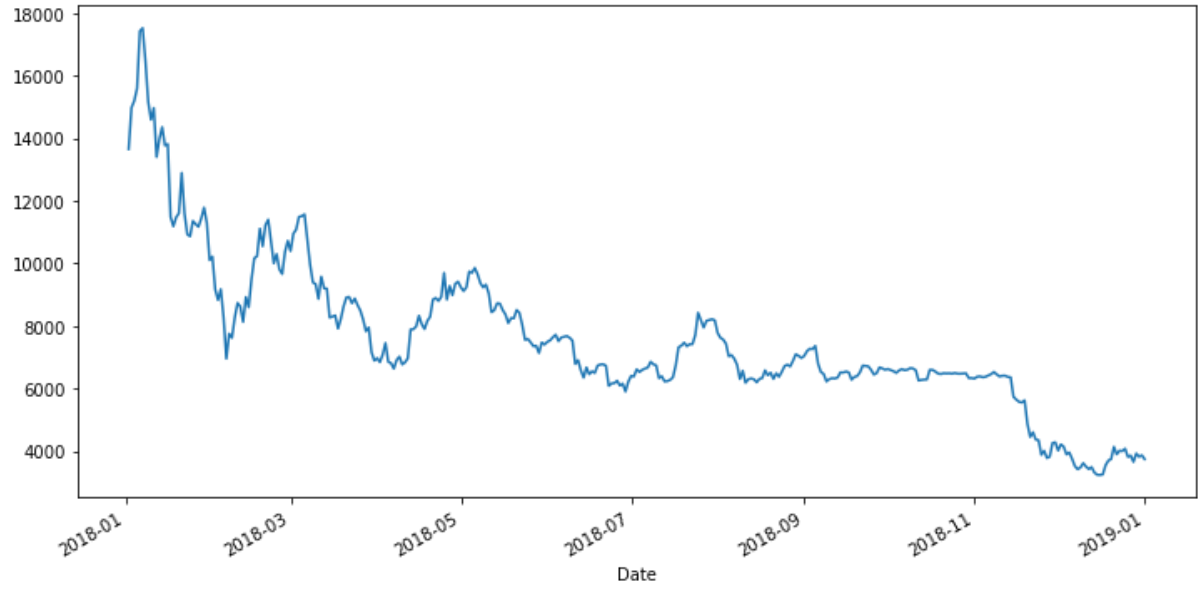
- Yearly data time series
- Sliding window data time series

Before creating the time series, the last 20% (573) of data points will be saved for later evaluation of the model and will not be used during training.

For the first approach, each time series has data from only one year. The following image is an example of time series with data from bitcoin's daily closing price in 2018. A total of 7

[10] https://docs.aws.amazon.com/sagemaker/latest/dg/linear-learner.html
[11] https://docs.aws.amazon.com/sagemaker/latest/dg/deepar.html

time series are created. The first and last time series might not have the full year of data, but the DeepAR model can handle missing data so no preprocessing is needed.



For the second approach, series are created by taking a window of 150 data points with a superposition of 30 data points. These values could be tuned to find the best ones, but for now they have been chosen with intuition. A total of 72 time series are created.

Training and optimizing

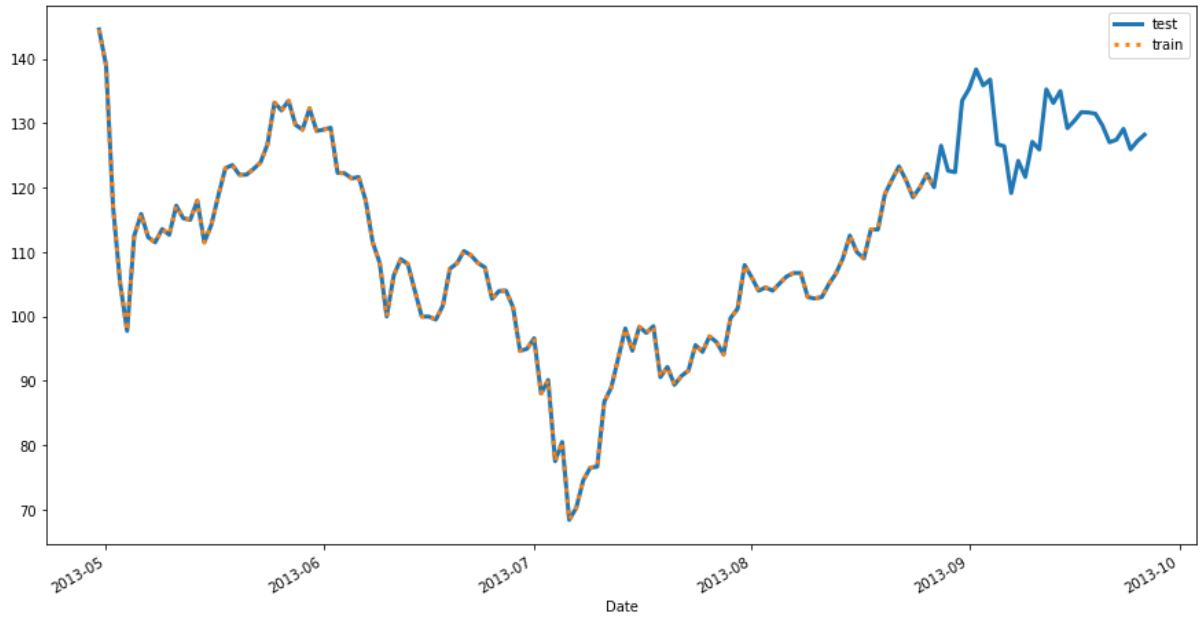
SageMaker trains the DeepAR model by randomly sampling training examples from each target time series in the training dataset. Each training example consists of a pair of adjacent context and prediction windows with fixed predefined lengths. To control how far in the past the network can see, use the `context_length` hyperparameter. To control how far in the future predictions can be made, use the `prediction_length` hyperparameter.

The DeepAR model automatically generates feature time series to facilitate learning time-dependent patterns. Since our data has daily frequency, the feature time series that will

be created are: `day-of-week`, `day-of-month` and `day-of-year`. To capture seasonality patterns, DeepAR also automatically feeds lagged values from the target time series. Therefore the time series can be longer than the `context_lengt` specified.

Once the time series have been defined, the training time series are created by removing the last `prediction_length` values. An optional test dataset can be passed to the algorithm to evaluate the model. The test dataset will use the full test series and the algorithm will automatically predict the last `prediction_length` values and evaluate against the real ones. The `prediction_length` used is 30 days.

Here is an example of a train and test time series:



To find the best model, hyper-parameter tuning is used. A total of 40 models are trained (20 using yearly time series and 20 using sliding window time series). Following the documentation of tunable hyper-parameters[12] for the DeepAr model, the parameter space used for tuning is:

- `epochs`: min 100, max 500
- `context_length`: min 30, max 60
- `num_cells`: min 40, max 50

From this parameter space, values are taken with a Bayesian strategy. The other parameters use the given values or the default values. The finally used hyper-parameters[13] are the following:

[12] https://docs.aws.amazon.com/sagemaker/latest/dg/deepar-tuning.html
[13] https://docs.aws.amazon.com/sagemaker/latest/dg/deepar_hyperparameters.html

Parameter	Trained with yearly times series	Trained with sliding window time series
prediction_length	30	30
epochs	284	168
context_length	50	60
num_cells	43	41
time_freq	D	D
dropout_rate	0.1	0.1
early_stopping_patience	10	10
learning_rate	0.001	0.001
mini_batch_size	128	128
num_layers	2	2

Results

At the end, three models are trained and deployed to an endpoint where inferences in new data can be made. The models used are:

- A linear model used as baseline
- A DeepAR model trained with yearly time series
- A DeepAR model trained with sliding window time series

To be able to compare models, predictions are made on the 573 data points reserved for evaluation that were not used during training.

Evaluation metric

The evaluation metric used is the Root Mean Squared Error (RMSE) between the predicted and actual value. Its definition is the following:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}\left(predicted_i - actual_i\right)^2}{N}}$$

One-step-ahead predictions

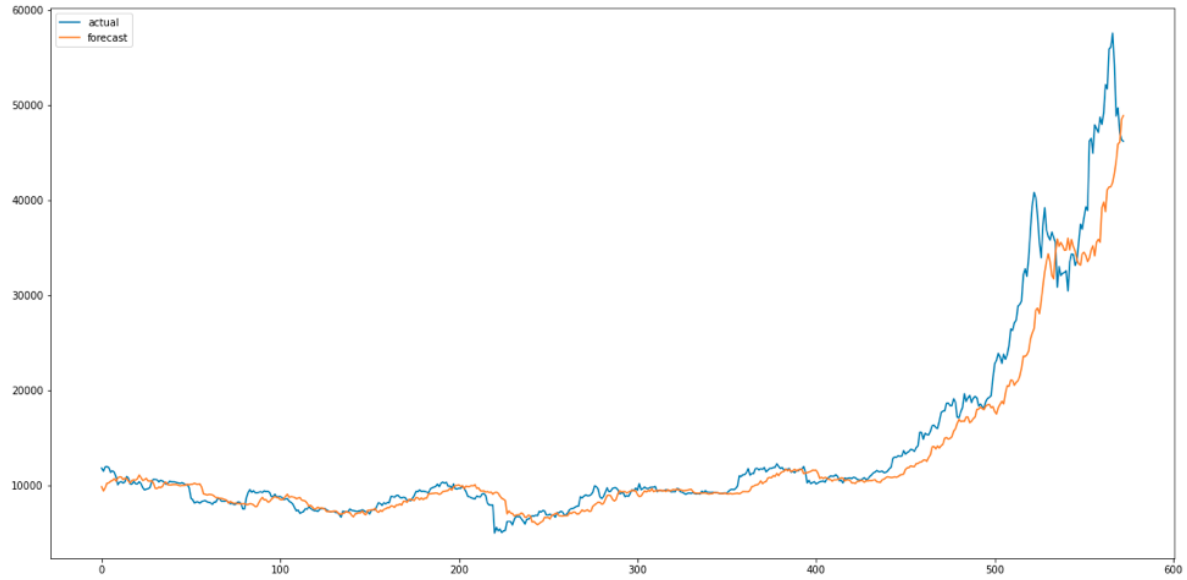
Predictions on the evaluation data are done generating one-step-ahead predictions. This is done by only looking at the prediction for the next day and using the ground truth for the input data, meaning that the predictions made are not used as inputs for the next prediction.

The results show that the model trained with the sliding window time series gives the best results by reducing the baseline error more than a half and having a better score than the other DeepAR model trained with yearly data.

Model	**RMSE**	**Improvement**
Linear model (Baseline)	2949	0%
DeepAR trained with yearly time series	1414	52.05%
DeepAR trained with sliding window time series	**1250**	**57.61%**

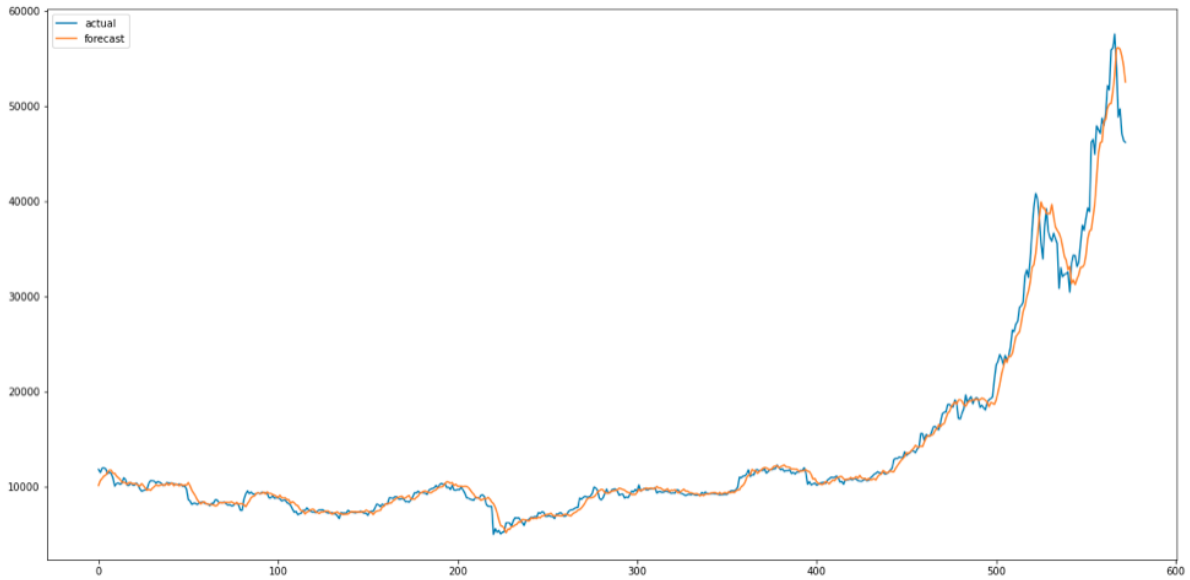
Linear model predictions

```
One-step-ahead RMSE =  2948.7851200624755
```



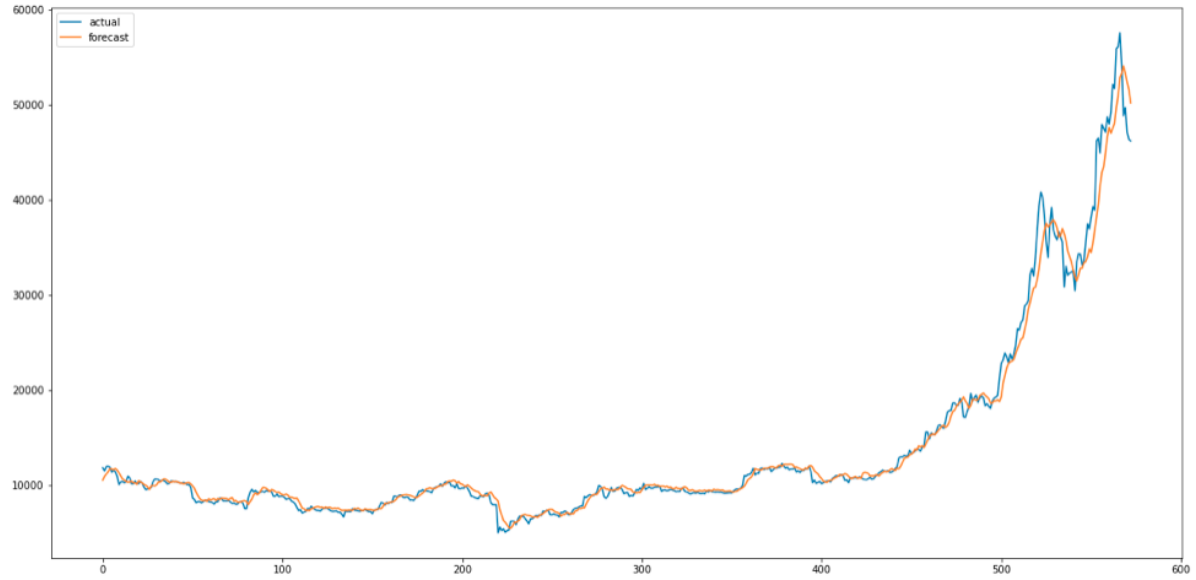
DeepAR trained with yearly time series predictions

```
One-step-ahead RMSE =  1413.6709448718364
```



DeepAR trained with sliding window time series predictions

```
One-step-ahead RMSE =  1250.1798693922492
```



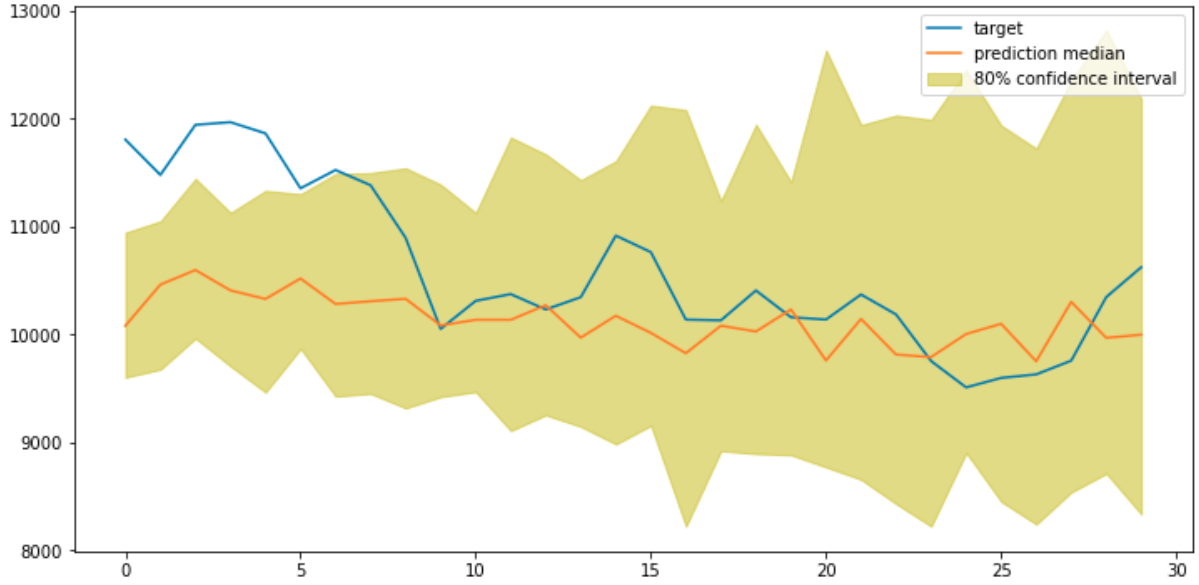
Probabilistic forecasting

The one-step-ahead approach was used to be able to compare the linear model with the DeepAR model. The output of the DeepAR model gives more than a single value prediction, it gives an estimate of the median and the specified quantiles for the following prediction length points. In our case, the estimated prediction of the flowing 30 days is computed with 50 different sample predictions. The 10th and 90th quantiles are computed along with the median. The median is the one used for the one-step-ahead predictions.

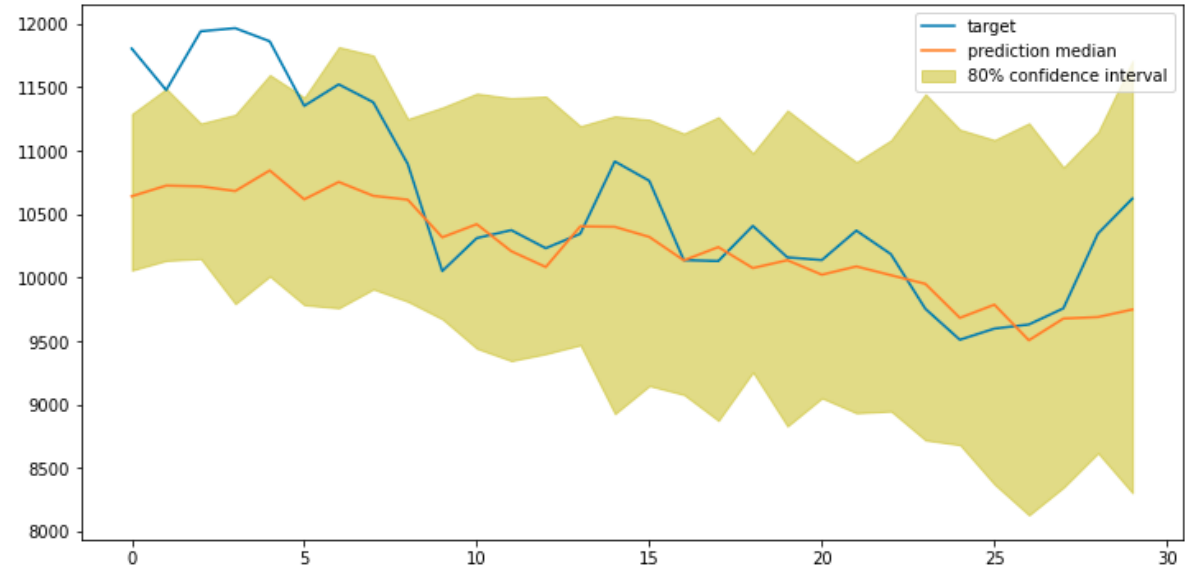
When doing the tests, different length input data had different prediction outcomes. Since I don’t know the theory and I did not have time to experiment more, instead of using the whole training data, only the last context length used in training is used as input data points.

In the following graphs the actual values can be seen in blue. The median of the output is colored in orange and the area between the 10th and 90th percentile is colored in green.

DeepAR trained with sliding window time series predictions
context_length=50, prediction_lenght=30



DeepAR trained with yearly time series predictions
context_length=60, prediction_lenght=30



Conclusions

Time series forecasting is an apasionating field of machine learning. During this project I have been able to learn how to create time series and how to train, deploy and evaluate a forecasting algorithm.

For forecasting cryptocurrency prices, a DeepAR model has been trained and optimized to find the best hyper-parameters. The time series used for training have been created with two different approaches: taking yearly data and taking sliding window data. Moreover, a linear model has been trained to use as a baseline.

The final better performant model on the evaluation data in terms of one-step-ahead RMSE is the DeepAR model trained with the sliding window time series. This model improves the baseline error by 57%.

Future work

There are some things about this project that can be further explored. For instance, model optimization could be more exhaustive to find a better model. This will be time and resource consuming.

In a real case scenario, the probabilistic forecasting of the DeepAR model could be used to perform actual decisions. In that case a different evaluation metric and `prediction_length` may be used.

The DeepAR model also allows as input dynamic features that can be used along the created time series to give more information to the model. For example, the number of daily tweets containing the word Bitcoin.

Another feature of the DeepAR model is that categorical features can be used to encode the groups that the record belongs to. This way the algorithm can be trained with time series belonging to different groups. A good application for that would be to train a single DeepAR model with the data of all cryptocurrencies instead of just using Bitcoin's data.