Dig deep enough

Features Engineering techniques for Time-series Analysis in Python.

Ali TWIL May 31, 2020 · 6 min read ⋆

Photo by Dragos Gontariu on Unsplash

Before we start digging into our Data, it's very important to have an idea about Time Series.
Basically, A time series is a **sequence of numerical** data points in successive order, In most cases, Time series is a set of observations on the values that a variable takes at **different times**.

In this article, I'm going to focus on some basic techniques of Features Engineering that can enhance your predictions over time.
It's been a long time since collaborating in a time series project, so I decided to have a quick refresh of the techniques I used to improve my prediction and share it with you.

Feature Engineering

Feature engineering is the process of using the domain knowledge to create relevant features in order to make your Machine Learning algorithms more accurate. Feature engineering can impact noticeably the performance of your modeling journey, If it's done correctly, it helps your model to perform very well.

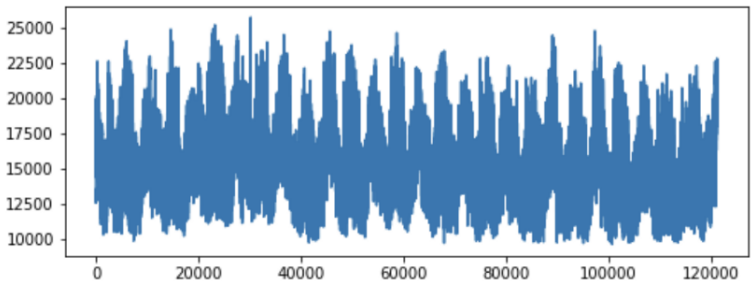
Before involving this step, you need to follow the very basic pipeline of solving a problem in Data Science, which is :

- Defining your goal;
- Extracting Data;
- Cleaning Data;
- Exploring the data and find insight (Exploratory Data Analysis);
- **Feature engineering;**
- Defining your model;
- Training and tuning your model;
- Deploying the model.

Feature engineering is an important step that differentiates between a good and a bad model. In the upcoming sections, we will talk about the application of those techniques in Time Series Analysis.

Let's suppose that we need to predict the energy consumption of some area, then we were given this data:

	Datetime	AEP_MW
0	2004-12-31 01:00:00	13478.0
1	2004-12-31 02:00:00	12865.0
2	2004-12-31 03:00:00	12577.0
3	2004-12-31 04:00:00	12517.0
4	2004-12-31 05:00:00	12670.0



The data is pretty self-explanatory, it's a univariate Time Series, we some repetitive pattern, we have two columns Date and value, the date column is treated as an object, so we need to convert it into a *DateTime* type, this can be done easily using Pandas :

```
import pandas as pd
data = pd.read_csv("AEP_hourly.csv")
data["AEP_MW"].plot(figsize = (8,3))
data.head()
data["Datetime"] = pd.to_datetime(data['Datetime'],format='%Y-%m-%d %H:%M')
```

reading_data.py hosted with ♡ by GitHub view raw

Now that our data is ready, let's have a look about things that we can dig from those variables, as I said, the dataset contains two columns, a date associated with a value, so let's discuss some of the techniques used to engineer this kind of problems.

Date Related Features

Date related features are a special type of categorical variables, a date provides only a specific point on a timeline. A date can be engineered properly if we have some domain knowledge or a deep understanding of the problem.
Since we want to predict energy consumption, let's ask some questions...

- Is there any relationship between Date/Time and consumption?
- Is there at least one trend in the data?

To answer the first question, if our data is related to some personal usage, for example, we know that the use of energy is lower at night compared to daytime; so we can conclude that there can some hidden pattern if we extract hours from the date, and this answers the second question too since we know that there's a repetitive trend in our data.

Extracting weekends and weekdays could be useful to know if there are any changing patterns during the week, and months/years too could be a very helpful measure to see if there's a change in the seasons.

Extracting these features is very simple with Python :

```
data['year']=data['Datetime'].dt.year
data['month']=data['Datetime'].dt.month
data['day']=data['Datetime'].dt.day
data['dayofweek']=data['Datetime'].dt.dayofweek
data['dayofweek_name']=data['Datetime'].dt.day_name()
data.head()
```

	Datetime	AEP_MW	year	month	day	dayofweek	day_name
0	2004-12-31 01:00:00	13478.0	2004	12	31	4	Friday
1	2004-12-31 02:00:00	12865.0	2004	12	31	4	Friday
2	2004-12-31 03:00:00	12577.0	2004	12	31	4	Friday
3	2004-12-31 04:00:00	12517.0	2004	12	31	4	Friday
4	2004-12-31 05:00:00	12670.0	2004	12	31	4	Friday

Features extracted from Date variable

We can extract more than that, for example :

- Quarter
- Semester
- Is start of a month
- Is start of year
- Is end of month
- Is end of year

```
import pandas as pd
data = pd.read_csv("AEP_hourly.csv")

data["Datetime"] = pd.to_datetime(data['Datetime'],format='%Y-%m-%d %H:%M')

data['year']=data['Datetime'].dt.year
data['month']=data['Datetime'].dt.month
data['day']=data['Datetime'].dt.day
```

```
data['dayofweek']=data['Datetime'].dt.dayofweek
data['day_name']=data['Datetime'].dt.day_name()
data['quarter']=data['Datetime'].dt.quarter
data['is_month_end']=data['Datetime'].dt.is_month_end
data['is_month_start']=data['Datetime'].dt.is_month_start
data['is_year_start']=data['Datetime'].dt.is_month_start
data['is_year_end']=data['Datetime'].dt.is_month_start
```

date_extraction.py hosted with ♡ by GitHub view raw

Code for some date extractions

Time-Based Features

In the same way, we can extract more features using the timestamp part of the date, for example, we can extract hours, minutes, and even seconds since the date is hourly so the only feature that we can extract here is hours.

```
data['Hour'] = data['Datetime'].dt.hour
```

We can generate a lot of features using the date column, here's the documentation link if you want to explore more about Date/time-based features.

Lag features

The lag features are basically the target variable but shifted with a period of time, it is used to know the behavior of our target value in the past, maybe a day before, a week or a month.

> *Using lag features can be sometimes a double-edged sword, since using the target variable* ***is very tricky and it can lead sometimes to overfitting*** *if not used properly.*

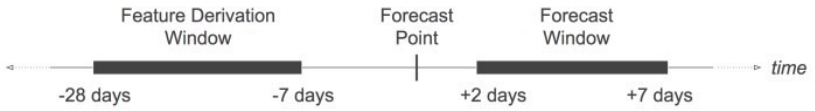
```
data["Lag1"] = data['AEP_MW'].shift(3) # Lag with 3 hours
data["Lag5"] = data['AEP_MW'].shift(5) # Lag with 5 hours
data["Lag8"] = data['AEP_MW'].shift(8) # Lag with 8 hours
```

```
data[["Datetime", "AEP_MW", "Lag1", "Lag2", "Lag3"]].head(10)
```

	Datetime	AEP_MW	Lag1	Lag2	Lag3
0	2004-12-31 01:00:00	13478.0	NaN	NaN	NaN
1	2004-12-31 02:00:00	12865.0	NaN	NaN	NaN
2	2004-12-31 03:00:00	12577.0	NaN	NaN	NaN
3	2004-12-31 04:00:00	12517.0	13478.0	NaN	NaN
4	2004-12-31 05:00:00	12670.0	12865.0	NaN	NaN
5	2004-12-31 06:00:00	13038.0	12577.0	13478.0	NaN
6	2004-12-31 07:00:00	13692.0	12517.0	12865.0	NaN
7	2004-12-31 08:00:00	14297.0	12670.0	12577.0	NaN
8	2004-12-31 09:00:00	14719.0	13038.0	12517.0	13478.0
9	2004-12-31 10:00:00	14941.0	13692.0	12670.0	12865.0

Lag features

Rolling Window Features



Rolling window features are calculating some statistics based on past values :

- **The Forecast Point:** Point in time that a prediction is being made;
- **Feature Derivation Window (FDW):** A rolling window, relative to the Forecast Point;
- **Forecast Window (FW):** Range of future values we wish to predict, called Forecast Distances (FDs).

We can implement that using this line of code :

```
data['rollingMean'] = data['AEP_MW'].rolling(window=6).mean()
```

	Datetime	AEP_MW	rollingMean
0	2004-12-31 01:00:00	13478.0	NaN
1	2004-12-31 02:00:00	12865.0	NaN
2	2004-12-31 03:00:00	12577.0	NaN
3	2004-12-31 04:00:00	12517.0	NaN
4	2004-12-31 05:00:00	12670.0	NaN
5	2004-12-31 06:00:00	13038.0	12857.500000
6	2004-12-31 07:00:00	13692.0	12893.166667
7	2004-12-31 08:00:00	14297.0	13131.833333
8	2004-12-31 09:00:00	14719.0	13488.833333
9	2004-12-31 10:00:00	14941.0	13892.833333

Similarly, we can generate a number of different time series features that can be useful to predict different forecast distances :

- Rolling mean, min, max, ... statistics
- Bollinger bands and statistics
- Rolling entropy, or rolling majority, for categorical features

Expanding Window Statistics

Another type of window that includes **all previous** data in the series.

This can be implemented easily in Python by using the *expanding()* function :

```
data['ExpandingMean'] = data['AEP_MW'].expanding(6).mean()
```

	Datetime	AEP_MW	rollingMean	ExpandingMean
0	2004-12-31 01:00:00	13478.0	NaN	NaN
1	2004-12-31 02:00:00	12865.0	NaN	NaN
2	2004-12-31 03:00:00	12577.0	NaN	NaN
3	2004-12-31 04:00:00	12517.0	NaN	NaN
4	2004-12-31 05:00:00	12670.0	NaN	NaN
5	2004-12-31 06:00:00	13038.0	12857.500000	12857.500000
6	2004-12-31 07:00:00	13692.0	12893.166667	12976.714286
7	2004-12-31 08:00:00	14297.0	13131.833333	13141.750000
8	2004-12-31 09:00:00	14719.0	13488.833333	13317.000000
9	2004-12-31 10:00:00	14941.0	13892.833333	13479.400000

Rolling mean vs Expanding mean

All those features can be useful when we have a good understanding of the problem, and this will lead us to make meaningful insights and avoid features explosion.

There's some other useful Features Engineering techniques for non-Time Series data that I like to share with you.

Interaction Features

There's some other type of feature engineering that involves revealing interactions between features, some features can be combined to extract a piece of specific information, so basically, we can sum, product, difference or quotient between two variables.

> *A general tip is to look at each pair of features and ask yourself, "could I combine this information in any way that might be even more useful?"*

- **Sum of to variables:** Let's imagine we have two variables, `selling_food_amount` and `selling_gadgets_amount`, we could sum those if we only care about the overall amount of the expenses.
- **Difference between two features:** Let's say we have two features `expiration_date` and `manufacturing_date` we can create a new feature called `product_validity`.
- **Product of two features:** We have a problem of defining the best houses in some area, and we have those two features `number_of_houses` : number of houses within 10 km of the area, `median_houses` : the median quality score of those houses, and what is important, is to have many options for houses but only if they're good enough.

To create this interaction we could create a simple product between those two variables : `house_score = number_of_houses x median_houses .`

- **The quotient of two features:** Let's use an example from this coronavirus pandemic, we can generate the mortality rate feature by : `mortality_rate = numberOfDeaths / numberOfCases .`

Conclusion

Feature engineering turns your inputs into things the algorithm can understand.

Generating too many features can confuse your model and lower the performance, we should always choose the most contributing features and remove unused and redundant ones, and most importantly don't ever use your target variable as an indicator (this includes a very close lag feature), this could give you very misleading results.

That's all Folks ! this was my very first article by the way, I hope you enjoy it.

Don't forget .. Believe the data, not opinions.

Data source: https://www.kaggle.com/robikscube/hourly-energy-consumption

Other sources :

- Alice Zheng & Amanda Casari, Feature Engineering for Machine Learning, *Principals and techniques for Data Scientists* (2018), http://oreilly.com/catalog/errata.csp?isbn=9781491953242
- Chapman & Hall/CRC, Feature engineering for Machine Learning and Data Analytics (2018), https://www.crcpress.com/Chapman--HallCRC-Data-Mining-and-Knowledge-Discovery-Series/book-series