Prediction of Wikipedia Traffic

Marina Zavalina (mz2476)

December 19, 2018

1 Introduction

Predicting the traffic of the website is an important task. First, one can get insights about consumers' behaviour by analyzing the data. Second, forecasting can help to make efficient economic decisions. In this case, the ability to forecast Wikipedia traffic allows to estimate the capacity of server that should be maintained.

In my project I use Wikipedia traffic data. The aim is to predict traffic for each page for next 60 days. I try models such as ARIMA, Kalman filter, Holt-Winters forecasting to make predictions for each page separately, as well as LSTM – one model for all pages. Only LSTM could beat the baseline prediction (median). Other models predict well for weekly seasonal data (Kalman filter with seasonality even beats LSTM), but only 9% of the data is weekly seasonal. Trained LSTM make predictions for new data fast, while for ARIMA, Kalman filter, Holt-Winters forecasting much more time is needed as they need to fit on new data first.

2 Problem definition and algorithms

2.1 Task

The task is to predict future traffic for each page for next 60 days. Common machine learning techniques (such as categorical feature engineering and trees) do not work here as we need to make prediction for long period, not for 1-2 days. So if we use such feature as value from yesterday, we have it only for one out of 60 days. Thus, time series models must be used for forecasting in this case.

The evaluation metric is symmetric mean absolute percentage error (SMAPE):

$$SMAPE = \frac{1}{n}\sum_{t=1}^{n}\frac{|\hat{y}_t - y_t|}{(|\hat{y}_t| + |y_t|)/2}$$

2.2 Algorithms

First group of approaches that I used aim at predicting traffic separately for each page.

2.2.1 Constant prediction

Simple baseline is a constant prediction by the last day value for all 60 days.

Also the baseline can naturally be mean or median. I use grid search for simple mean, median to obtain optimal number of days for which they are calculated.

Another idea is to use more sophisticated median: weighted median. It works similar to median of medians that was used by many participants. We know that day $t-1$ should have more weight than day $t-30$. We take median of last 150 days, 100 days, 50 days, 30 days, ... and finally take the median of all of them (participants used 1, 1, 2, 3, 5, 8, ... weeks...). The figure 1 shows the approach.

Weighted median allows to apply float weights for data and performs better. I used exponential decay for weights, giving preference to the last one, and tuned smoothing parameter.

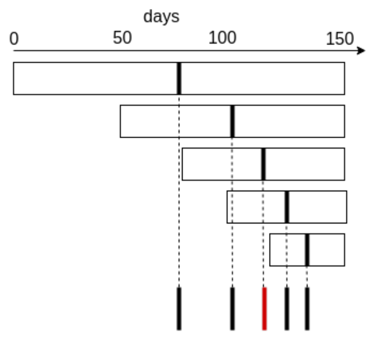

Figure 1: Weighted median: median of each batch is drawn in black, median of all medians — in red.

2.2.2 ARIMA

Autoregressive integrated moving average (ARIMA) model has several parameters: p is the number of time lags of the autoregressive model, d is the degree of differencing, q is the number of time lags of the moving-average model. One more parameter is the number of last days to train the model on. I used 100 in all cases as the data for a page can be very unstable (so more lags may worsen the predictions) and also it is faster to train.

As I couldn't tune manually parameters for each page, I wrote a function that chooses p and q based on autocorrelation function (ACF) and partial autocorrelation function (PACF). As for d, I checked two specifications: 0 and 1.

2.2.3 Kalman Filter

First, I used simple Kalman smoother (as it is doing forward and backward steps, so it works better) with one latent state to smooth train data and EM algorithm to train the parameters (transition matrix, observation matrix, transition covariance, observation covariance). Then I varied the number of latent states.

Second, as we know the type and structure of data, we can directly encode weekly seasonality [1] We can use 8 latent states: first state will store the smoothed mean, other seven will store smoothed residuals for seven days of the week. The choice of the matrix guarantees that these residuals sum up to zero, thus, the first state stores the true mean as in a simple one state model. (Otherwise, one can subtract 100 from the first state and add 100 to all other states. We want some stability and interpretability of the states.) To recover observations from the latent states, we will need to get mean state and state that corresponds to the value from the same day of last week. Then the transition matrix (8x8) and observation matrix (1x8) that allows to update the states as described above are:

$$A_{z_{t-1}\to z_t} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & -1 & -1 & -1 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad C_{z_t\to x_t} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

[1] I used the idea presented in this discussion on kaggle: `https://www.kaggle.com/c/web-traffic-time-series-forecasting/discussion/43727`.

2.2.4 Holt-Winters Forecasting

Holt-Winters model can be used for forecasting when the data show trend and seasonality.[2] The idea is to apply triple exponential smoothing to capture mean, trend and seasonality. As we know the period of seasonality (7 days), this method should work good with seasonal data.

2.3 Neural Networks

Though some other architectures work and show good results (indeed, CNNs with WaveNet type architecture were used in winning solutions) I used RNN, LSTM in particular.

LSTM is trained on all data and can learn some patterns from the data. For example, while analyzing the data, I noticed that pages in Spanish are more weekly seasonal than pages in other languages. LSTM can theoretically learn it from data.

I use seq-to-seq 3-layer LSTM model with skip connection between layers[3]; I additionally normalize the data before putting it through LSTM and apply inverse transformation to the predictions. The transformation is needed to make all data uniform, therefore, increasing the training data as now neural network does not need to learn mean and variance of each page.

The biggest advantage of the neural networks is that we can use other categorical features aside from the time series while the other classical models do not allow it. It is also possible to "help" neural network by passing weekly or yearly lags so the model can better predict big spikes which are crucial for articles about people (spikes around birthdays) or holidays: other models take into account only last days and do not have such a long memory and even LSTM and other RNN do not learn yearly lags. Though I have not used such an architecture as it was not my main goal, the winning solutions incorporated it by passing tuples like this (last day, 7 days lag, 365 day lag, embedding for day of the week, embedding for month of the year, embeddings for other categorical features).

Finally, neural networks require less knowledge about the data and work well without any feature construction, however they are less interpretable and require skills in Deep Learning as well as more time for each experiment.

3 Experimental evaluation

3.1 Data

The data comes from kaggle competition and is provided by Wikipedia[4]. The dataset contains number of views per day for 145,000 pages for the period of 803 days. Additionally, for each page we know its language, name, type of access ('all-access', 'desktop', 'mobile-web'), and type of agent ('spider', 'all-agents').

The data is not Gaussian, it follows exponential distribution with high spike at zero. So I apply $\log(1 + x)$ transformation to it. The data contains nans. I use forward fill operation (if there is nan, we put previous value instead of it). If there are nans at the beginning, it means that the page was not created yet. Thus, I fill such nans with mean of transformed data, which gives no information about this period to forecast models and avoids decrease in the mean.

Some of time series possess weekly (9% of all the data) and yearly (4%) seasonality. There is no increasing trend. Also there are pages that have many zeros. The graph 2 shows these different patterns.

3.2 Methodology

As the dataset contains very different instances (some are seasonal, some are not popular and contain a lot of zeros), I chose three subsets from all given pages: 1,300 random, 500 weekly seasonal, 200

[2]Explanation of the model can be found here: `https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc435.htm`

[3]I follow the architecture proposed on `https://github.com/fehiepsi/web-traffic-time-series-forecasting/blob/master/executable/model_rnn.ipynb`

[4]Link to data: `https://www.kaggle.com/c/web-traffic-time-series-forecasting/data`.

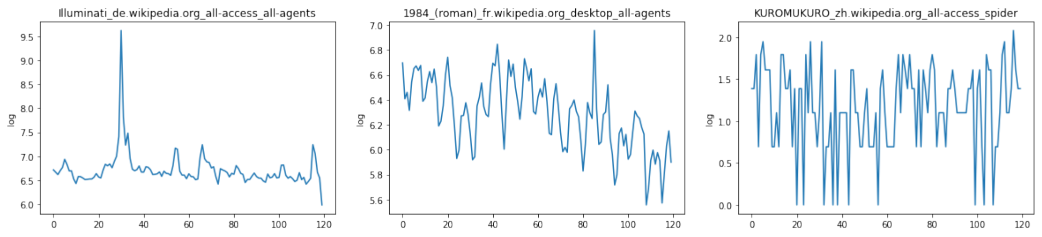

Figure 2: Examples of log-data: random page, weekly-seasonal page, page with many zeros.

containing “many zeros” (2,000 in total). Also it was done because models such as ARIMA are trained for a couple of minutes on one page, so it is impossible to evaluate them on the whole dataset of 145,000 articles.

I report SMAPE score for weekly seasonal and "many zeros" subsets, as well as the score for all chosen 2,000 pages. I also estimate the time (based on how many GPUs I use and how much time it takes the model to fit and give predictions for chosen 2,000 pages).

I post-process the results obtained from models: I remove nans (if any), negative values and round the values to closest integer (the target variable is of integer type) as it gives better results. (The same idea as with MAE metric in predicting probabilities.)

3.3 Results and Discussion

	SMAPE				
	full	subset_2000	many_zeros	weekly	time
last_elem	47.08	47.78	89.68	33.20	0 seconds
nanmedian_on_last_16	41.99	43.84	79.89	32.81	0.04 seconds
weighted_median_092_decay	41.12	43.10	80.10	31.78	10 seconds
kalmam_filter	-	47.75	84.44	34.71	555 hours
arima_d_0	-	56.11	96.69	47.51	4833 hours
arima_d_1	-	50.36	87.88	36.88	2900 hours
kalmam_filter_with_seasonality_learn_all	-	157.38	86.12	197.38	60 hours
kalmam_filter_with_seasonality_learn_cov	-	62.73	89.59	50.54	50 hours
kalmam_filter_with_seasonality_no_learn	-	42.65	82.98	28.40	19 hours
holt_winters	-	53.02	87.65	42.27	120 hours
lstm_3_layers_skip_coonections	41.61	43.49	81.80	32.78	-1
medians+lstm	40.43	42.20	79.45	30.82	-1

Figure 3: Caption

The table 3 summarizes the results. Each specification is stored in a row. There is SMAPE score for predictions for all data (not available for some models because of high computational complexity), for selected 2,000 pages and subsets of selected data – weekly seasonal, many zeros. "Time" column shows estimated time of training the model on all 145,000 pages.

Based on SMAPE score for 2,000 selected pages, Kalman filter with seasonality (and with constructed matrices that I described above) and LSTM+median models perform better than other models.

As for weekly seasonal data, Kalman filter with seasonality outperforms other models. The score is better when we do not use EM algorithm to train the parameters but rather use constructed matrices described above. The figure 4 shows predictions of different models for a weekly seasonal page.

We can see that almost for all models the score drops significantly when it comes to forecasting pages with many zeros. It didn’t happen with Kalman filter with seasonality and learned parameters as it may have tried to overfit on "many zeros" pages, and thus, performs worse on all pages.

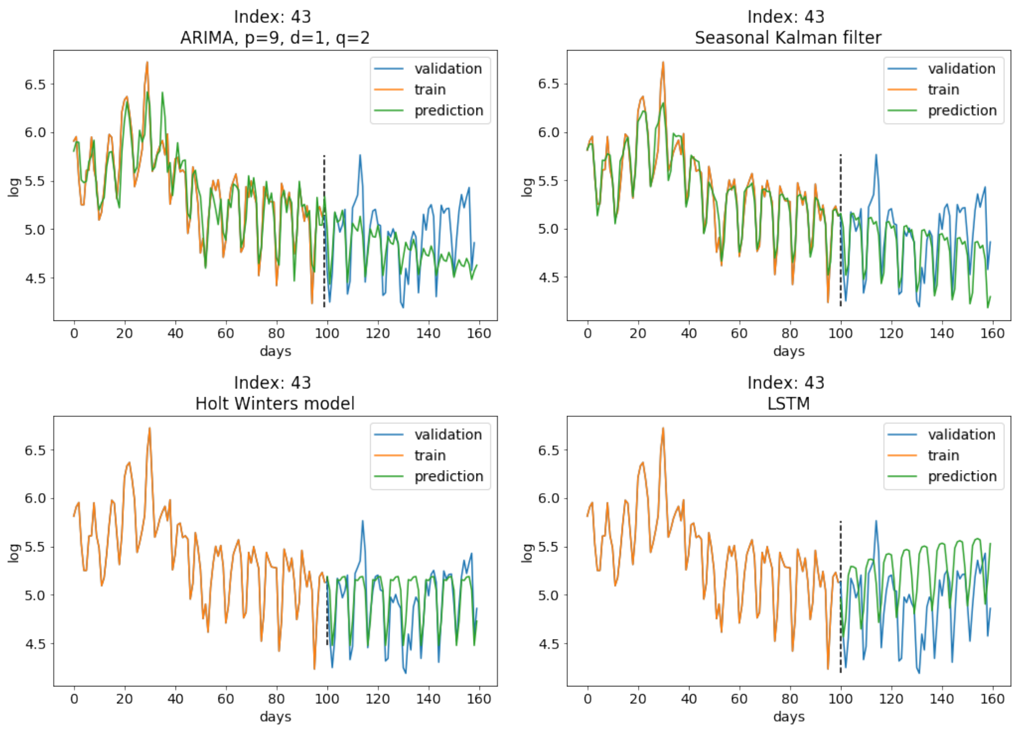

Figure 4: Predictions of different models for weekly seasonal data. Though predictions of these models look good for weekly seasonal data, it gets much worse when we take “many zeros” or random data.

We can observe that median performs quite well in this case. It happens because of the choice of metric. SMAPE is approximately average of absolute errors of log of data. We know that in this case the best log-likelihood estimator is median. I also ran a simulation to test it (figure 5).

Considering the time for fitting and predicting, ARIMA takes at least ten times more than Kalman filter with seasonality (without training), while predicts worse. The latter can be coded in a more efficient way to work even faster: we do not need to learn the parameters, also we can parallelize the computations for different pages (using numpy).

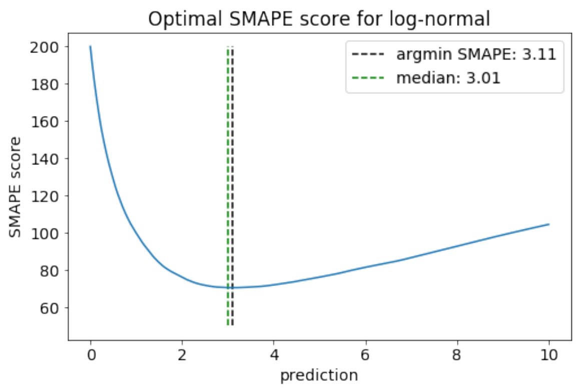

Figure 5: SMAPE optimization.

4 Conclusion and future work

As I have described in the Models section, the input of LSTM can be changed to take into account categorical features and to stress the importance of such values as a week ago, a year ago. Also one can think about different architecture of the model that will allow the network to memorize information from year ago.

Gaussian process (GP) model can be tried on seasonal data. GP with spectral mixture kernel are good at forecasting, though it will take much time to fit it for all the data, as with Kalman filter and other models I used.

We saw that almost all the models are doing worse when it comes to predicting traffic for pages with many zeros. So future research can be done in this direction in order to find a model that predicts such data well and then combine it with Kalman filter or LSTM. Also a way of improving LSTM architecture to deal with "many zeros" pages can be thought of.

5 References

DS-GA 3001.001/.002 Probabilistic time series analysis course materials: `https://github.com/charlieblue17/pTSAFall2018`

Data and ideas: `https://www.kaggle.com/c/web-traffic-time-series-forecasting`

LSTM architecture: `https://github.com/fehiepsi/web-traffic-time-series-forecasting/blob/master/executable/model_rnn.ipynb`