BLOG

Demand Forecasting 2: Machine Learning Approach

By Semantive August 13, 2018 No Comments

This is a third post in our series exploring different options for long-term demand forecasting. Today, we will explore different approaches to applying classical machine learning to forecasting problem. To better understand our journey and problem setting, you might want to check out our **introductory blog post:** Long-Term Demand Forecasting

Step by step vs 90 days at once

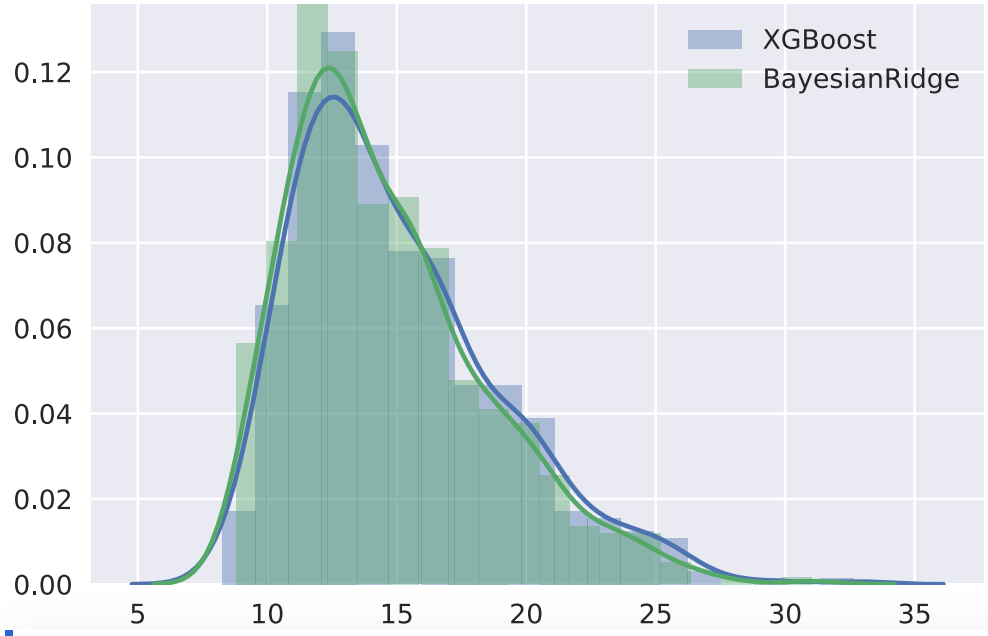
We believed that using models predicting one day ahead and feeding these

set) than doing 90 days at once (15.53). This, however, should be taken with a grain of salt, because feature engineering might give the 90 days at once an advantage.

We haven't found an existing solution that would allow for doing a step-by-step prediction. Thus we created some helper functions that compute new input features based on original time series and previous model forecasts. Having separate functions for generating training data and making predictions saved us from test data leaking into training which happened a few times when computing features like monthly average (mistakenly we took future dates into consideration so that the model could 'peek into the future'). For more information please check our "rolling features" util on GitHub.

Linear regression and XGBoost

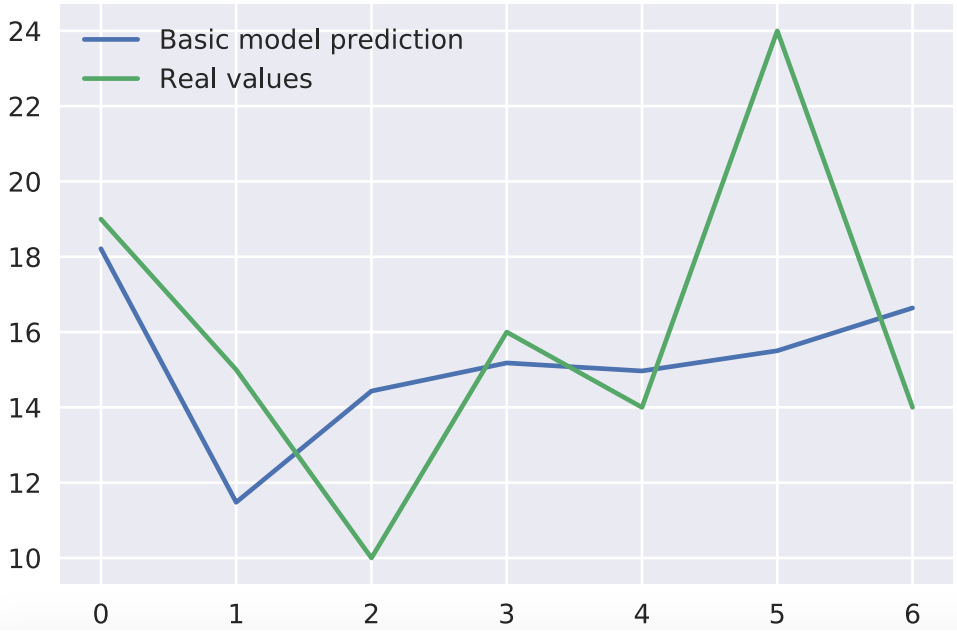
At first, we tried various linear regression models from the sklearn package like LinearRegression, Ridge, ElasticNet and BayesianRidge to quickly establish a baseline for the rest. They achieved validation scores between 14.5 and 18, however after submitting the best one (BayesianRidge) to kaggle, it scored a mere 15.29. Afterwards, we tried gradient boosting with the XGBoost library, however it performed similarly to the linear models in step by step prediction achieving a validation score of 15.04.



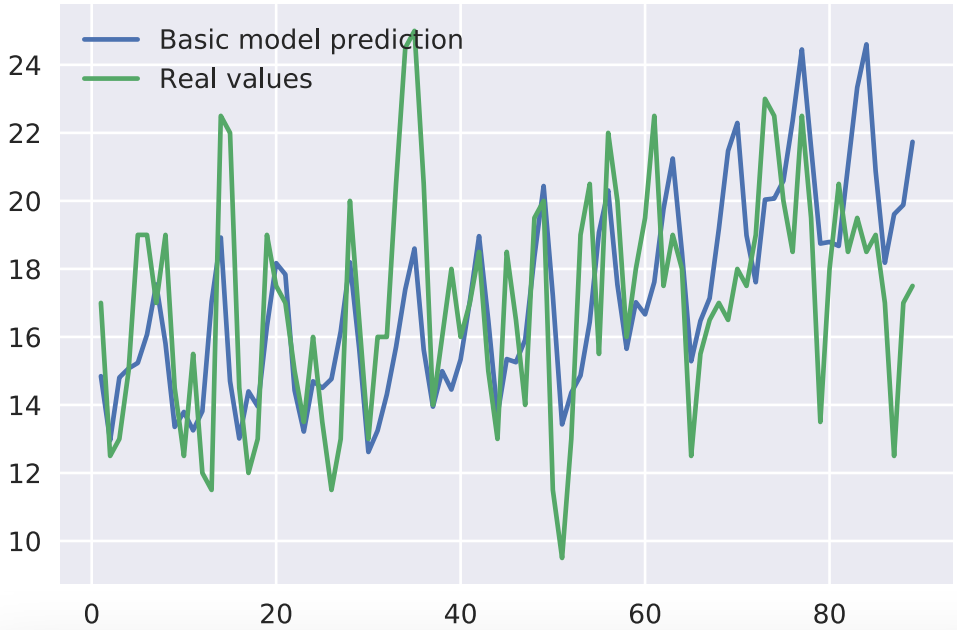
A comparision of SMAPE distributions between XGBoost and BayesianRidge models.

We tried feeding our models sales from previous 90 and 365 days, as well as adding other features from statistics (min, max, mean, variance, standard deviation, median) of sales in some time intervals – last week or last month, but most of the time adding too many features only made things worse.

The linear models often were able to adapt to the seasonality, but adding deseasonalization based on averaging usually improved their results.



A sample Bayesian Ridge one-week ahead model forcast for one of the time series.



A sample Bayesian Ridge model 3-months ahead forecast

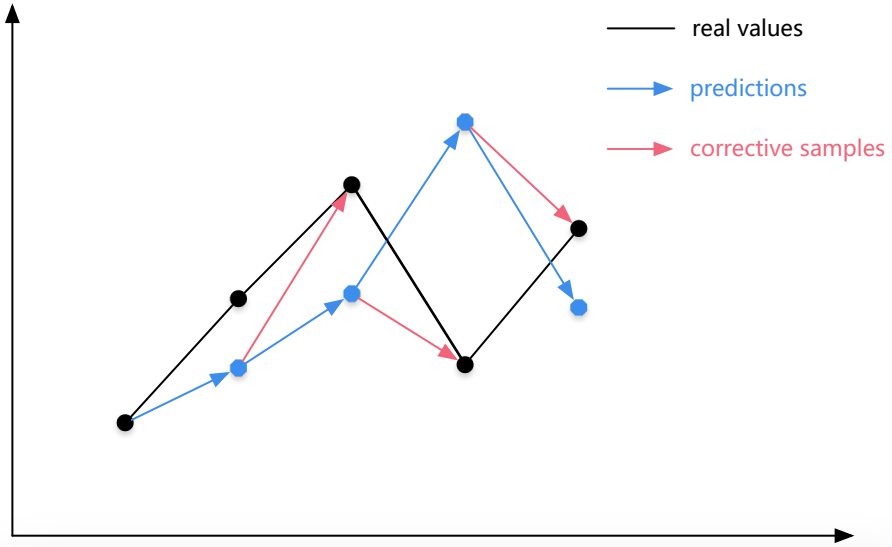
Error accumulation

We were startled that sometimes when our models achieved very small errors while predicting a single day ahead, their 90 day forecast error would grow. The reason for that is a discrepancy between our training target and the test target – we were training the models to predict a single day ahead but used them to make longer predictions. This has two traps:

The first problem is that for the long-term predictions it's possible that the whole input to the latter data points is entirely based on previous prediction thus the error may grow exponentially. It would be reasonable to assume that when the error of our model on a single prediction decreases, the long-term error should decrease as well, because the model is more precise. This was, however, not the case: during training we were validating our models performance on single-day predictions, however later we wanted a good performance in the long-term. Predicting a single day and multiple days is not the same and our model was trained for the former while we expected it to be good at the latter. These two tasks have some differences, for example considering the long-term trend and seasonality is not as important for single-day predictions so the model may ignore it, while it is essential for the long-term forecasts.

Data as demonstrator

We've decided to implement an algorithm from a paper about improving multi-s predictions. The idea is to train the model multiple times, each time expanding t training set with data points that are meant to correct the errors the model has made



A visualization of the Data as demonstrator algorithm.

After each training round we use the model to make a multi-step prediction and extend the training set with pairs of inputs created from the predictions of the model and output being the true values that were to be predicted.

This approach is similar to imitation learning– training data provides corrections for the errors that arise in multi-step predictions. The improvement the algorithm provided was, however, slightly random – sometimes the error decreased just after 1-2 epochs, but other times the error skyrocketed. Unfortunately, the algorithm is quite slow, and it did not improve accuracy as much as we had expected (for example, XGBoost only from 14.5 down to 14.15).

Time series as cross-sectional data

Inspired by Kaggle kernels that achieved high scores on the leaderboards by encoding weekdays and months by the mean value of their respective period, we

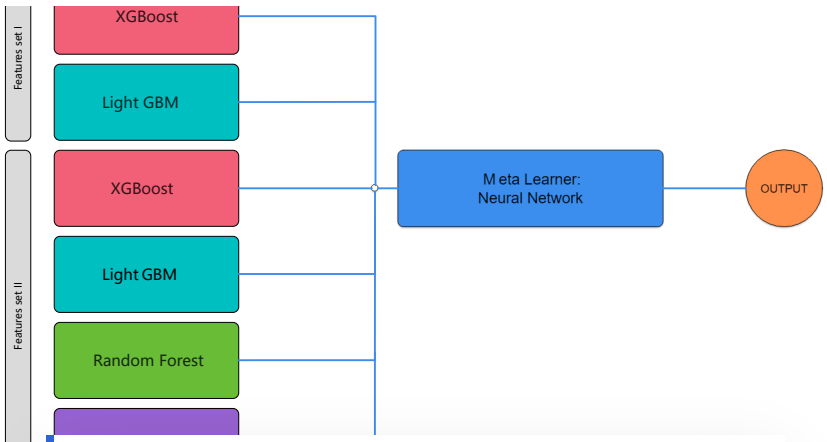
predictions are independent of each other, there's no error to accumulate, regardless of the forecast length. On the other hand, this method cannot recognize long-term trends. It's certainly not a universal approach, but it works well in this case, thanks to the very regular data. While it potentially gets rid of the error accumulation, it stands no chance of predicting spikes and other more complicated features.

To replicate this approach, we extracted features like minimum, maximum, mean, median, variance and standard deviation. Combining median or mean with variance seems to work best. It was important not to fall into the trap of adding too many features, because this actually worsened the scores.

Combining multiple models

In an effort to optimize the kaggle score, we tried stacking, in particular blending with XGBoost, LightGBM, k-NN and RandomForests as base models, and neural network as a meta learner. We split dataset into 3 parts: a training set for models, a training set for meta learner and

a validation set. After the training we achieved a score of 14.07 on Kaggle, which is an improvement, but its complexity is usually too high to be viable in production environment.



An example blending architecture used by our team.

Next time in our series we will go through deep neural networks that we used to tackle this problem. Stay tuned not to miss it, and in the meantime feel free to check out our code on GitHub.

Got a project idea? Let's schedule a quick, 15-minutes call to discuss how Big Data & Data Science services may give you the edge your business needs. Get in touch →

RECENT POSTS

4 modern AI solutions for manufacturing

Targi pracy IT 2019 w Warszawie. Co, gdzie, kiedy?

High-Performance computation in Python | NumPy

Text Summarization in Python

Data Science internship, and why Semantive program is worth it?

Previous Post
3: Neural networks

Leave a Reply

My comment is..

CONTACT US

+48 510 002 513 | contact@semantive.com

ul. Nowogrodzka 42/41, 00-695 Warsaw, Poland

SEMANTIVE

Big data | AI & Data Science | Cloud

Services that make your organization data informed