Biking in the Rain: Supervised Learning with Temporal Data

Alex E. Yuan
Program in Molecular and Cellular Biology
University of Washington
Seattle WA 98195
aeyuan@uw.edu

Abstract

I addressed the problem of predicting precipitation from traffic along the Burke-Gilman Trail. Without data preprocessing, the performance of techniques popular in the literature (random forest and k-nearest neighbors with dynamic time warping) was poor. Adding preprocessing steps dramatically improved performance, allowing the humble ridge regression to outshine more sophisticated predictors.

1 Problem and Datasets

I am interested in temporal data as I encounter this type of data often in my research. I would like to see how well I can predict the precipitation in Seattle on a given day based on that day's traffic along the Burke-Gilman trail. The City of Seattle maintains a set of sensors on the Burke-Gilman trail near NE 70th that count pedestrians and cyclists, and per-hour counts of travelers are available online [1]. The dataset has several features; I am using as my features the total number of detected travelers for each hour of a day. For the response variable, I found a record of Seattle daily precipitation levels on Kaggle.com [2]. The datasets have a few years of overlap (2014-2017). After removing days with missing sensor data I have 1413 days with full data. Specifically, I have a feature matrix $\mathbf{X} \in \mathbb{R}^{1413\times 24}$ where each day is a row and each column is the total traffic count in a particular hour. The precipitation can be represented as a vector $\mathbf{y} \in \mathbb{R}^{1413}$ where each entry is the daily precipitation.

Note that seasonality of weather, the effects of weekends, and the fact lots of days have no precipitation are all demons that could create structure in my data. For simplicity, I do not explicitly use data such as the date or day of the week for my predictions and instead shuffle samples to partially alleviate temporal trends.

For my data splits, I first shuffled and divided the 1413 days into 3 sections: (1) a 'development' set containing 76 % of the data, (2) an 'interactive' test set containing 12 % of the data, and (3) a 'final' test set containing the remaining 12 % of the data. I used the development set for hyperparameter tuning, used the 'interactive' test set to interactively assess performance of rainfall predictors, and used the 'final' test for a final unbiased assessment of the performance of my predictors.

2 Initial approach

In my initial approach, I assessed the accuracy of three predictors (ridge regression, k-nearest neighbors, and random forest) on the interactive test set. For hyperparameter tuning I partitioned 85 % of the development set for training and the remaining 15 % for a single validation split. I tuned the λ term in ridge regression, k in k-nearest neighbors (KNN) and the number of features used at each split in the random forest using the validation set. I kept the number of estimators in the random forest at 100, which will be the default in sklearn in version 0.22, and I kept any other hyperparameters for

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

Table 1: MSE on "interactive" test set

Precipitation predictor	Ridge regression	KNN (Euclidean)	KNN (DTW; $w = 5$)	Random forest	$\hat{y} = 0$
Test MSE	0.14	0.080	0.080	0.11	0.096

the random forest as their defaults in sklearn [6]. I then computed mean squared error (MSE) of the predictors with tuned hyperparameters on the interactive test set.

Since we have seen ridge regression and random forests in class, I will not introduce them here. We discussed KNN classifiers in class, but not KNN regressors. My KNN regressor uses the following rule for estimation: To predict the precipitation of a new sample with features $\tilde{x}_i^T$, the KNN regressor first finds the k nearest training samples using some distance metric and estimates the precipitation of the new sample as a weighted average of the precipitation values of these k nearest training samples, where the weights are the inverse distances. I found this intuitive rule recommended on WIkipedia [3]. I tried two different distance metrics: euclidean distance and local dynamic time warping (DTW). At a conceptual level, dynamic time warping finds a shortest distances between two sequences, allowing for these sequences to be nonlinearly stretched or compressed in time to find an optimal alignment. A hyperparameter that can be added is the window size w, which constrains how far a time point in a sequence can be stretched from it's original position. I used $w = 5$. I tried DTW because it's popular in the literature for sequence classification [4]. A more technical description of DTW is available at [5].

The results of this initial test are shown in Table 1. On the rightmost column of Table 1 I report the test MSE that you would have if you just predicted every day as 0 precipitation. I thought this would be. a natural benchmark since 55 % of days in the interactive test set did not have precipitation. Since DTW did not improve accuracy beyond the Euclidean distance metric, I decided abandon the DTW approach and instead explore different ways to preprocess the data to perhaps improve prediction accuracy.

3 Preprocessing strategies

I explored several different approaches to preprocess the data. I used ridge regression for this analysis since its speed allows me to run leave-one-out cross-validation (loocv) for more robust results. For each preprocessing approach, I tuned the λ hyperparameter for ridge regression by loocv on the development set and then trained on the entire development set and evaluated MSE on the interactive test set. In what follows I will describe the different preprocessing approaches (results are given in Table 2). As a control I tried ridge regression without any preprocessing. (far left column of Table 2). Note that using loocv alone (as compared to a single validation set in Table 1) allowed ridge regression to cut test MSE by 50 %, suggesting that this dataset is quite noisy.

The first preprocessing approach I tried was to divide the traffic counts at each hour of the kth day by the sum total of the traffic counts in the kth day. That is, $\tilde{x}_{i,j} := x_{i,j} / \sum_j x_{i,j}$. I refer to this as 'horizontal normalization' or 'hz norm' for short, and apply this transformation both during training and at test time. The idea behind this is that there is some cohort of travelers along the Burke Gilman trail whose behavior is deterministic and unaffected by weather (e.g. needs to get to work or home regardless of conditions) and another cohort whose behavior is more random and also weather-sensitive (recreational users). Thus weather patterns might show up in the fold-differences between traditionally high-traffic times and traditionally low-traffic times. This pre-processing step was quite successful, reducing error to 0.61. Using z-scores (subtracting the mean and dividing by standard deviation) worked well also.

I wondered whether the hours themselves really contained useful information, or merely the values of their rank ordering. To test this idea, I sorted the hours within each day by traffic counts after applying horizontal normalization. The test MSE increased slightly, suggesting that the rank ordering preserves much, but not all of the information that ridge regression uses to infer precipitation. The success of hz norm and z-scores is also consistent with the explanation that outliers in the dataset are impairing the generalization of regression rules between subsamples (e.g. from the development set to the test set). If this is true, we should expect improved test accuracy if we simply 'clipped'

Table 2: MSE with ridge regression and preprocessing on "interactive" test set

Preprocessing	none	hz norm	z-score	hz norm + sort	clip85	$y = 0$
Test MSE	0.071	0.061	0.063	0.067	0.63	0.096

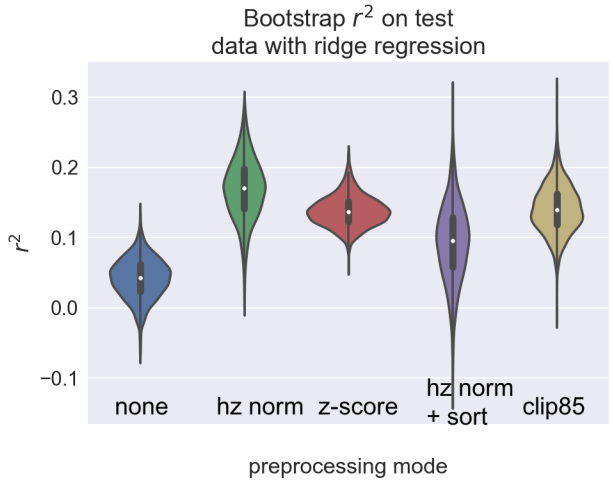

Figure 1: Interactive test r^2 of ridge regression with preprocessing

the data by first computing the pth percentile of traffic counts at each hour and then, for each count $\tilde{x}_{i,j}$, taking the minimum of $\tilde{x}_{i,j}$ and the pth percentile of counts in that hour. This approach, where $p = 85$, was equally effective as taking z-scores.

To get a sense of the confidence of the performance of these approaches and also look at the data another way, I used bootstrap resampling with 1000 to compute the r-squared values that these preprocessing strategies achieve on the interactive test set. The result is shown in Fig. 1. Horizontal normalization looks like the winner here.

Encouraged by these results, I moved to try these preprocessing strategies with the KNN and random forest regressors. Using hz norm and z-score preprocessing I performed hyperparameter tuning for the k value of (Euclidean) KNN and the number of splitting features in the random forest regressor using the same training and validation sets as previously (i.e. as in Table 1). With these tuned parameters I evaluated test MSE with hz norm and z-score preprocessing with the interactive test set. KNN achieved a MSE of 0.076 and 0.077 with hz norm and z-score respectively, which was a small improvement. The random forest did not improve, achieving an MSE of 0.11 for both preprocessing steps. This result is consistent with the idea that these preprocessing steps were helping to reduce the influence of outliers but the random forest could already do this since it is composed of decision trees.

The best-performing predictors so far have been ridge regression with hz norm, z-score, and clipping. To get an unbiased estimate of the error these predictors will have on new data, I trained them on the development set and tested on the 'final' test set. I found that ridge regression with hz norm, z-score, and clipping preprocessing steps achieved an r-squared correlation coefficient of 0.11, 0.018, and 0.12 on the final test set, suggesting that hz norm and clipping may be more robust approaches to processing the data than taking z-scores.

3.1 Code

I used numpy, pandas, matplotlib, seaborn and os. I implemented ridge regression, DTW, and KNN, bootstrapping, and all preprocessing steps. I used only two classes form sklearn: those for the r-squared metric and for the random forest. My project is at https://www.dropbox.com/sh/uroi01ax70w2b2e/AACziiMphHOtvNphlby4f_pFa?dl=0.

4 Next Steps

It would be interesting to see whether a neural approach can automate the preprocessing step. That is one could train a neural network to solve this regression problem and then use the first couple layers from that network as the input to any of the predictors used here.

References

[1] https://www.kaggle.com/city-of-seattle/seattle-burke-gilman-trail/home

[2] https://www.kaggle.com/rtatman/did-it-rain-in-seattle-19482017/home

[3] Wikipedia. (2018). K-nearest-neighbors algorithm. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[4] Ratanamahatana, C. A., & Keogh, E. (2005, April). Three myths about dynamic time warping data mining. In Proceedings of the 2005 SIAM international conference on data mining (pp. 506-510). Society for Industrial and Applied Mathematics.

[5] Muller, M. (2007). Dynamic time warping. Information retrieval for music and motion, 69-84.

[6] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html