Prediction of Bike Rentals

Tanner Gilligan – tanner12@stanford.edu
Jean Kono – jkono@stanford.edu

Abstract

We constructed a custom linear regression model to try and impute missing data from a time series of bike sharing rentals. The dataset contained some time features (e.g. months 1-12) which we translated to indicator variables, as well as weather information for that day. We customized the linear regression model by experimenting with different loss functions and attempting to exploit structure in the data. We found that the data was very noisy, and therefore it was difficult to make accurate predictions, but our best results came when we split the data by month and created a linear regression model for each month. This method correctly predicted within 25 of the true value on 25.3% of the test set.

1 – Introduction

1.1 – Objective

Our objective is to try to accurately impute missing data from time series bike-rental records. Although predicting bike rentals has very few real-world applications, the idea of trying to impute missing information in a time series is quite valuable. Imputation of time series information is useful any time data can become corrupted, or a sensor can malfunction, leaving holes in the dataset.

1.2 – Dataset

Our dataset contains hourly shared bike rental information for Washington D.C. from 2011 and 2012. We obtained our dataset from the UC Irvine Machine Learning Repository, but it is also available as one of the competitions on Kaggle. Our dataset was composed of 17,379 different data points, each of which contained 13 different input features and 3 response features.

1.3 – Features

The dataset was very complete including features about time and weather. We also noted the existence of implicit neural networks, such as “Feeling Temperature” being a function of the “Temperature”, “Humidity”, and “Wind speed” features. In addition, many of the feature were already normalized (divided by the largest value), allowing the feature’s domain to be fully contained in [0, 1]. The Dataset contained the following features:

Time

- Date (MM/DD/YYYY)
- Hour {0-23}
- Day of the week {0-6}
- Month {1-12}
- Year {0, 1}

Time Meta-Information:

- Government Holiday {0, 1}
- Working Day {0, 1}
- Season {1-4}

Weather:

- Precipitation {1-4}
- Temperature – normalized [0, 1]
- Humidity – normalized [0, 1]
- Wind speed – normalized [0 1]
- Feeling Temperature (e.g. windchill)

1.4 – Evaluation Metrics

We divided our data into a training set and a test set, approximately 70% and 30% of the data were used in each set accordingly. The training set was randomly sampled without replacement from our full data set and the test set was the remainder. This was done to simulate random data removal. All models were fit on the training set. We used K-fold validation on the training set to generate validation errors to compare the models and measured test error by the performance on the test set.

We chose two different evaluation metrics for analyzing how well the models were functioning. Our primary evaluation metric is the mean squared error (MSE) of the data points, which gives us an idea of how well the regression model is fitting the data. Our secondary evaluation metric is prediction proximity (VE), which tries to capture how many of the prediction values were "close" to the actual response value. We chose to have this secondary evaluation metric because our objective is to try and impute the missing data, and this provides us with a way to measure how good the predictions are, as opposed to just how well we fit the model. This idea of "close" is a rather subjective term, so for our project we ran our initial baseline once and checked the proximity of the predictions against various proximity values, and learned the following:

Proximity	VE	MSE	Test Error
±10	.892	22002	.968
±25	.846	22002	.919
±50	.713	22002	.856
±100	.517	22002	.759

We selected 25 as the value we would use for our project since had plenty of room for improvement, and was a reasonably wide range. This choice of 25 was still a relatively arbitrary choice, however, and one could select any proximity value they wanted.

2 – Experimentation

2.1 – Feature Generation

The first thing we decided to do was translate the given time variables, such as month and day, into indicator variables. This is because a feature like month should not assign 12 times more weight to December (12) than to January (1); they are very similar and should be treated as such. In addition, we found that some of the features were non-monotonic; that is, their effect on the response did not increase as the feature's value increased (e.g. temperature). In order to rectify this, we replaced the temperature and feeling temperature features with a normalized feature representing the distance from room temperature (68°F).

Model	VE	MSE	TE
Baseline	.852	22002	.919
New Vars	.729	7404	.843

We also experimented with clustering the data and adding the cluster assignment from K-means Clustering as a feature:

Clusters	VE	MSE	TE
2	.715	7434	.838
3	.719	7432	.837
4	.722	7390	.792
5	.72	7363	.818

2.2 – Feature Selection

In order to help us select the most relevant features, we introduced a Lasso Penalty term to our loss function. This term penalizes the sum of the weights being too large, causing the algorithm to assign very low weights to those features that provide little insight. We ran the algorithm multiple times with different lambda values (constant in front of weight sum) in order to find those variables that were consistently pushed near 0. We found that the feature corresponding to "Heavy rain/storm" was consistently push to 0, which is likely due to this feature being active in very few data points. It would also consistently push either "temperature" or "feeling temperature" to 0, which is due to them being so closely related. However, once we removed these features and re-ran our algorithm, the results we got were so close to our original results that we didn't attribute a positive change in our model performance to the removal of features.

2.3 – Loss functions

Due to the fact that the majority of our error metrics rely on proximity and closely fitting the data points, having our linear model optimize mean squared error did the best of the basic loss functions we tried (Mean squared error, Huber Loss, Absolute Loss). However, since we also had the second error metric of being within 25 of the response value, we chose to construct a customized loss function to optimize this. Our customized loss function follows the same structure as mean squared loss (since this was already generally optimizing what we wanted), but we altered it by assigning 0 loss to anything that fell within the margin of 25. The custom loss function overall performed comparable to mean squared loss, out-performing it in some cases, and under-performing it in others in terms of test error.

Model	**VE**	**MSE**	**TE**
Baseline - M	.852	22002	.919
Baseline - C	.836	25532	.887
2 Clusters - M	.715	7434	.838
2 Clusters - C	.718	7409	.865

* C = Custom Loss, M = Mean Squared Loss

Since the difference between our custom loss function and mean squared loss was so small, we chose to only use mean-square loss for our testing and comparisons of hyper-parameters (e.g. # of clusters)

2.4 – Local Regression

The technique that performed the best out of everything we tried was the usage of localized regression. We divided the data into segments based on the month the data was recorded in, and created a regression model for each segment (each segment was also subjected to k-fold cross validation as part of the regression). The reason we chose to split up the data this way is because each segment could be modeled independently, thus allowing closer points (i.e. more relevant points) to have greater effect on the prediction. We experimented with different numbers of evenly distributed segments (starting from January and ending in December) and obtained the following results (note that this includes the modified features, excluding clustering):

Segments	**VE**	**MSE**	**TE**
1 (baseline)	.729	7404	.843
2	.729	6932	.783
3	.709	6647	.782
4	.704	6572	.771
6	.693	6498	.756
12	.684	6435	.746

Once we established that 12 segments performed optimally, we introduced cluster features and re-ran the local regression:

Clusters	VE	MSE	TE
2	.687	6438	.816
3	.687	6403	.7744
4	.687	6481	.804
5	.689	6440	.821

2.5 – Cluster Regression

After we developed our local regression models, we realized that we were essentially trying to group data points that were alike, and perform regression on these groups. Since it turned out reasonably well for splitting on month segments, we then decided to take this idea to the extreme and perform localized regression across the clusters that we had found when we used K-means clustering to implement cluster features.

Clusters	VE	MSE	TE
2	.712	5291	.847
3	.756	5068	.814
4	.733	4982	.84
5	.619	4584	.82
6	.761	5218	.861
7	.791	5437	.896
8	.672	5785	.772
9	.766	5622	.838
10	.822	5947	.874
11	.646	5486	.833
12	.806	5529	.887

2.6 – Hyper-parameters

In addition to coming up with techniques to better fit our data, we also needed to tune our hyper-parameters for optimal results. The three main hyper-parameters we needed to try and optimize were our number of iterations for stochastic gradient descent, the learning rate, and the coefficient for our lasso penalty term. We began by trying to optimize our learning rate, but found that if we increased it to .01 or greater, it would periodically diverge and crash our algorithm. Since we must ensure our algorithm doesn't diverge, we set out learning rate to .001 and tried to optimize our number of iterations based on this learning rate:

Iterations	Validation Error
10	.839
50	.767
100	.747
200	.743

From these results, we were able to deduce that 100 iterations was sufficient, as doubling the run time was not worth such a marginal increase. We then optimized our lasso penalty coefficient on the 12-segment local-regression model:

Coefficient	VE	TE
1	.681	.764
.5	.682	.754
.1	.685	.75
.01	.685	.749

From these results, we selected .1 as our coefficient value since it had the lowest discrepancy between VE and TE, and it was large enough that it could push feature weights to 0.

3 – Conclusion

3.1 – Final Results

For our final results, we present only the best result for each category based on validation error:

Algorithm	VE	MSE	TE
Baseline	.852	22002	.919
Baseline w/ I	.729	7404	.843
Baseline w/ (C = 2) & I	.715	7433	.838
Local Reg.	.684	6435	.746
Local Reg. w/ (C = 3)	.687	6402	.774

Cluster Reg. (C = 5)	.619	4584	.82

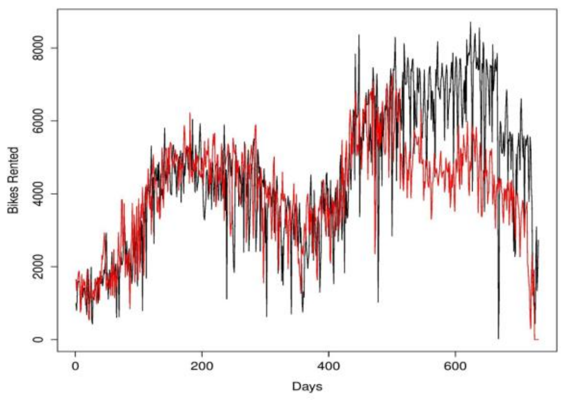
* I = indicator variables

* (C = 'X') = X clusters used

Below is a plot of the best model, Local Reg. (localized regression across months). Please note that our model was trained and predicts on hourly data, however, in order to make a more comprehensible graph, we show the cumulative totals for each day instead of the hourly values. Thus, this graph depicts less variance of the response and less disparity between predictions and reality.

Predictions vs. Reality

Red line = Prediction | Black line = Reality



3.2 – Analysis

We see from the table above, that our first type of localized regression (localized regression across months) performed the best on our test set. However, regression using K-means cluster assignments as indicator features performed the best on our validation error. Considering that localized regression across months does considerably better than Cluster Regression on the test set, this is surprising and undesirable: ideally, we want the model with the best validation error to have the best test error. One highly possible explanation is that since the clusters were fit on the entire training set before it was split into K-folds, we unknowingly gained information that allowed us to more successfully predict on the hold-out folds in K-fold cross validation. In the future, we will perform clustering on each iteration of K-fold cross validation instead of performing it once in the beginning.

Judging by the change in validation MSE, (form 22002 to 7404) it seems that the indicator variables we added were crucial for success. Our interpretation is that the way we represented our data with indicators allowed for much more information from the data to be used. Lastly, it seems that localized regression across months was a powerful way to model the data. Since our data is a time series where data points that are close together time wise are highly correlated, we conclude that doing localized regression across time variables is an effective way of capturing the time series structure.

3.3 – Future Work

If we were to continue working on this project, we would likely focus on creating features that are functions of existing variables. In addition, we would learn about regression trees, and see how they could be used to improve our results.

3.4 – References

Fanaee-T, Hadi, and Gama, Joao, *Event labeling combining ensemble detectors and background knowledge*, Progress in Artificial Intelligence (2013): pp. 1-15, Springer Berlin Heidelberg,

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning Data Mining, Inference, and Prediction.* Springer, 2009.