Predicting Query and Data Arrival Rates with Linear and Logistic Regression

William Ma
Department of Computer Science
University of Chicago
Chicago, IL 60637
williamma@uchicago.edu

1 Introduction

Modern decision support systems use databases as their data source. Since these decision support systems are built using a handful of queries, we can create materialized views (e.g. precomputed results that are maintained) around these queries to reduce query latency. As new data arrives, however, these views become stale and need to be updated. By maintaining some of the intermediate state generated when answering the query, the database can incrementally update the query with incremental view maintenance (IVM), which dramatically reduces query latencies. Traditional approaches of IVM [1, 2, 9, 10] fail to consider the dynamic workloads that ML models face [4].

Due to the workloads being dynamic, when we should update a view becomes a critical question. For view maintenance, two parts of the dynamic workload are considered: the query arrival rate, which affects when the materialized view is needed, and the data arrival rate, which affects if, and how much, computation is needed to update a view. View maintenance only depends on these two arrival rates because a view does not need to update if it is not requested (i.e., not queried) or there have been no changes (i.e., no new data). The adaptive IVM algorithm will depend on the arrival rates, which is not known a priori. Thus, we have to be able to predict the arrival rates as the rest of the system will depend on these predicted query arrival rates.

In this work, we narrow our focus on predicting the query arrival rates using only linear and logistic regression. While these models may be simple, we explore a wide variety of adjusting the input and output data by adjusting the:

- **grouping granularity**, which changes the granularity of the binning of the time series data
- **input dimension**, which adjusts the amount of information the model uses to predict each output
- **output dimension**, which adjusts how far into the future the model predicts for each input
- **training size**, which changes how much data is fed into the model
- **training frequency**, which changes how frequently we run new models to predict the new arrival rates

By exploring these differences, we are able to show the true potential of logistic and linear regression. The rest of this paper contains a literature review in section 2. Section 3 describes the specifics of our methods and section 4 explores the results of our experiments.

2 Literature review

While there has been prior work in predicting query arrival rates [7], they used a hybrid of a kernel regression and an ensemble of a linear regression and LSTM. Their approach used the kernel regression to predict future spikes in arrivals and used the ensemble of the linear regression and LSTM to predict the regular arrival rate. Our approach focuses on only kernel regression and linear regression and uses both to predict the arrival rate. Additionally, their work failed to do a comprehensive ablation

Preprint. Under review.

study, which we do in section 4. There also has been prior work [4] that has attempted to predict the arrival rate for machine learning model serving requests, which uses network calculus to predict the arrival rates. While our work is motivated by machine learning model serving requests, we chose not to go with network calculus as we believe that network calculus has too many hyperparameters without the ability to create an ensemble of network calculus models, as we are able to do with machine learning models.

Our work in predicting arrival rates can be boiled down to time series prediction. Prior work in time series prediction includes using support vector machines [8] and neural networks [3]. However, due to time constraints, we focused our efforts on a thorough analysis of linear and kernel regressions.

3 Methods

In order to demonstrate the true effectiveness of linear and logistic regression, we take three data sets from prior work, which we discuss in section 3.1, and search over a large combination of features, which we discuss in section 3.2. We test each of these features into our implementations of linear and kernel regression, which we describe in section 3.3.

3.1 Data Sources

We used three data sources, which were used in prior work [4, 6, 7]. These datasets consisted of

- Queries sent to a **BusTracker** app, which consists of 2 months of data that had a diurnal cyclic pattern.
- Queries sent to a **course listings** web app, which consists of 2 minutes of data sent to a course listings hosted by the computer science department at the University of Chicago.
- Inter-query arrival rate data used in Inferline [4] and **Autoscale** [5].

While the datasets we used are by no means comprehensive of the wide spectrum of arrival rates that exist in the real world, we believe that these datasets capture a good subset for an initial exploration into the space of query arrival rate prediction.

3.2 Knobs to Turn

Even with simple linear and logistic regression models, there are a number of variables that we have to tune when trying to predict the query arrival rates. These variables affect how we aggregate the data, how the model does predictions, and how the training is done.

When we change the *grouping granularity* (e.g., group data by the number of arrivals for every 2 seconds), it changes how the models observe the arrival rates. For instance, if we lower the grouping granularity to be too small (e.g., 1 nanosecond), the arrival rates will be incredibly sparse. In the other extreme, if we raise the grouping granularity to be too large, (e.g., 1 millennium), there will be very little data for the regression models to learn from. Thus, finding the optimal grouping granularity such that there is enough data to learn and make reasonable predictions, while simultaneously not having the arrival rates be too sparse to do predictions on, will significantly effect the prediction power of the regression models.

By changing the *input dimension* and *output dimension*, which is the amount of data that the model sees and has to predict per data point, respectively, directly affects how the model does its predictions. The intuition behind the input dimension is changing the number of possibilities for an arrival rate to match. For instance, if the input dimension is two, then the model will only have the number of arrivals for two groups. Using the two arrival rates, the model is forced to predict the future arrival rates. However, if we increase the input dimension, there is a larger number of possible arrivals to match the input to, which allows the arrivals to match a wider variety of outputs. The output dimension is the same except it allows the inputs to match to a larger number of outputs. The input and output dimensions are tightly linked and will be inter-dependent on each other.

Finally, it is obvious that changing the *training size*, the number of input and output points, and the *training frequency*, after how many training points before we retrain the regression model, will heavily effect the performance of the regression models.

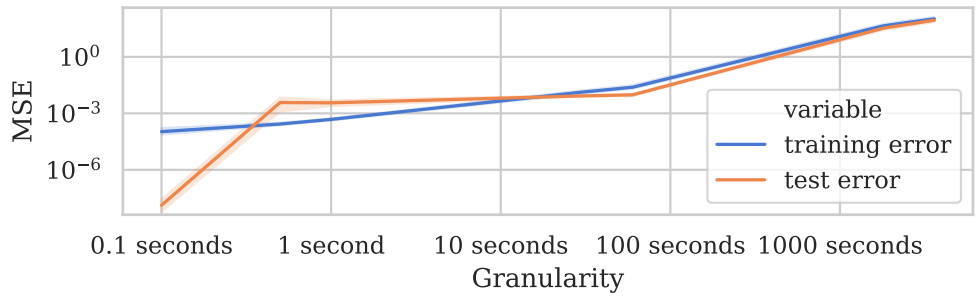

Figure 1: MSE of granularities for BusTracker

3.3 Model Implementations

For both models, we implemented it with ridge regularization. For linear regression, we implemented the standard ridge linear regression model. For kernel regression, we used a Gaussian kernel as follows:

$$\exp \frac{-(dist(\vec{x}, \vec{y}))}{2\sigma},$$

where $dist$ is the standardized Euclidean distance between the two vectors.

4 Experiments

For our experiments, we tested a regularization factor of 0.01, 0.5, 0.99, which captures having no weight to the weights (0.01), some weight (0.5), and evenly splitting weight (0.99). For the σ, used in the Gaussian kernel, we tested 0.25, 0.5, 2.5, 5, and 10. All the results we report are the with the optimal regularization factors and σ's.

Since, we do not see any significant differences between the two models so we simply report the mean results. Due to limitations of our data and computational power available, we are only able to report the results of each experiment with one or two of the datasets from section 3.1. Finally, all of our results are measured in terms of the mean squared error (MSE) with respect to the differences between the actual arrival rate and the estimated arrival rate.

4.1 Data Aggregation Size

In figure 1, we can see that, as we increase the granularity, the MSE increases. This is as we expected and can see that there is only marginal gains to be had after we reduce the granularity to about a minute. It is noteworthy that the test error does better with a lower granularity than the training error. We note that we only have this for the BusTracker data set as the other datasets were limited in their time range to do this experiment with. Thus, when designing an arrival rate predictor in a database, we should use a granularity that is at least within one minute. While this does have secondary implications, such as having to wait until we have enough data to do predictions, that is out of scope of this project.

4.2 How to Predict

From figure 2 and 5, we can see that there is a significant difference in the MSE between different input and output dimensions. If we only derive our conclusions from those two figures from the BusTracker dataset, then we would be inclined to argue that the input and output dimensions should be as small as possible. However, from figures 3, 4, 6, and 7, we can see that there is no one rule that can determine the input and output dimensions. Thus, when implementing the predictor in a database, we should maintain multiple regression models and use a weighted ensemble to ensure optimal accuracy.

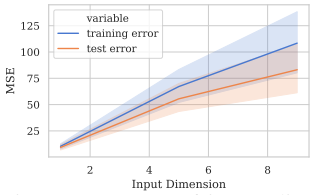

Figure 2: MSE of input dim. for BusTracker

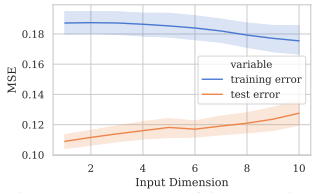

Figure 3: MSE of input dim. for course

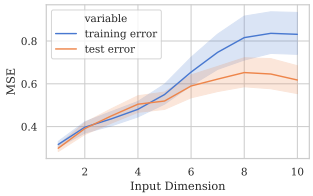

Figure 4: MSE of input dim. for autoscale

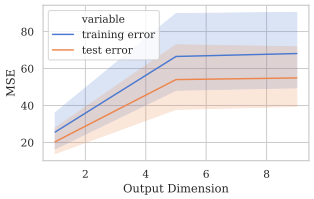

Figure 5: MSE of output dim. for BusTracker

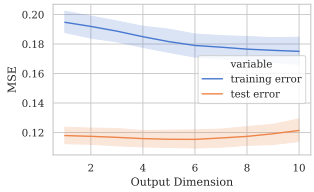

Figure 6: MSE of output dim. for course

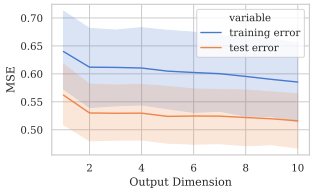

Figure 7: MSE of output dim. for autoscale

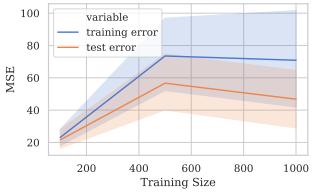

Figure 8: MSE of training size for BusTracker

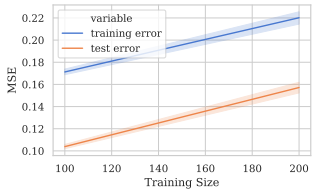

Figure 9: MSE of training size for course

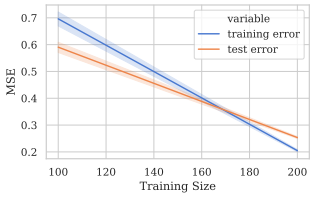

Figure 10: MSE of training size for autoscale

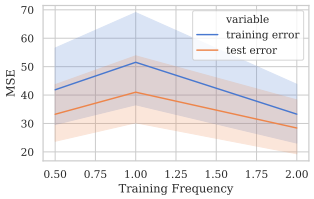

Figure 11: MSE of training frequency for BusTracker

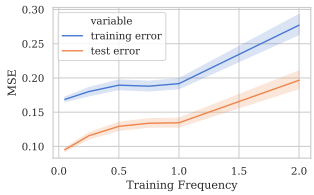

Figure 12: MSE of training frequency for course

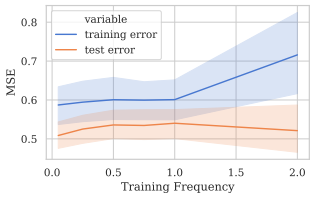

Figure 13: MSE of training frequency for autoscale

4.3 How to Train

To calculate the training frequency, in figures 11 - 13, we train the dataset for every training frequency times training set size. To calculate the MSE, we predict the training set size, if the training frequency is less than 1. Otherwise, we predict the training frequency times the training set size.

From figure 8 - 13, we can see that we get a similar conclusion as section 4.2 in that there is no patter that is consistent throughout different datasets. The sweetspot for both training size and training frequency is both dependent on the data.

5 Future Work and Conclusion

While this work has focused only on linear and logisitic regression, we plan to extend this type of analysis to SVM and neural networks, based on prior work [7, 8]. Additionally, we intend to use this type of analysis to build accurate arrival rate predictors into database systems directly. Since this is used in a database system, the creation of ensembles over the different hyperparameters discussed above will have to be handled based on the system's requirements. This work, however, has shown that we cannot simply pick one of these hyperparameters and have to tune these hyperparameters to the dataset.

References

[1] Yanif Ahmad and Christoph Koch. "DBToaster: A SQL compiler for high-performance delta processing in main-memory databases". In: *Proceedings of the VLDB Endowment* 2.2 (2009).

[2] Rada Chirkova and Jun Yang. “Materialized views”. In: *Databases* 4.4 (2011), pp. 295–405.

[3] Jerome T Connor, R Douglas Martin, and Les E Atlas. “Recurrent neural networks and robust time series prediction”. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 240–254.

[4] Daniel Crankshaw et al. “InferLine: Latency-Aware Provisioning and Scaling for Prediction Serving Pipelines”. In: *Proceedings of the 11th ACM Symposium on Cloud Computing*. 2020.

[5] A. Gandhi et al. “AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers”. In: *ACM Trans. Comput. Syst.* 30 (2012), 14:1–14:26.

[6] Xi Liang, Aaron J Elmore, and Sanjay Krishnan. “Opportunistic view materialization with deep reinforcement learning”. In: *arXiv preprint arXiv:1903.01363* (2019).

[7] Lin Ma et al. “Query-based workload forecasting for self-driving database management systems”. In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, pp. 631–645.

[8] Nicholas I Sapankevych and Ravi Sankar. “Time series prediction using support vector machines: a survey”. In: *IEEE Computational Intelligence Magazine* 4.2 (2009), pp. 24–38.

[9] Dixin Tang et al. “Intermittent query processing”. In: *Proceedings of VLDB Endowment* 12.11 (2019).

[10] Dixin Tang et al. “Thrifty Query Execution via Incrementability”. In: *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2020, pp. 1241–1256.