Estimating Traffic for Millions of Queries in Real Time

Oliver Dain[*], Tim Hesterberg[†], Yogita Mehta[‡], Laramie Leavitt[§]

Abstract
Google estimates traffic for millions of queries continuously. This is used for many applications, such as caching results for frequent queries, and looking for spikes. We describe how Google efficiently processes a wide variety of queries with frequencies ranging from tens to millions of hits per day, with widely varying activity across time of day and week, and updates these models continuously as new data is observed.

Key Words: Poisson regression, online algorithm, real-time.

1. Introduction

People search on Google for millions of different queries. Our goal in this work is to estimate the expected traffic for each unique query string (excluding those with very little traffic) quickly, using minimal memory and other computational resources. These estimates have many uses including determining which query results to cache for fast access, which queries to suggest to users, and for detecting changes in query rates. The latter may be accomplished by comparing the actual query rates to predicted rates. If the query rate is unusually high we say it is "spiking". This may indicate an interesting development, for example "earthquake Japan" on March 11, 2011. Unexpectedly low query rates may indicate outages.

Many queries have a regular pattern that includes large daily and weekly rate changes. We must accurately model the regular patterns for accurate planning and to distinguish spiking queries from queries whose whose rate is increasing as part of its normal pattern.

Different query strings exhibit very different patterns. For example, Figure 1 shows one week's data for three different queries. While each query exhibits strong regular patterns, the patterns are quite different. The query "mail.yahoo.com.vn" (Yahoo's mail service in Vietnam) has two strong daily peaks; for the first the query volume increases by more than an order of magnitude in less than an hour. This is not unusual. In contrast, "the bachelorette" has a strong peak once a week, just before the show. The query for "tv guide" also exhibits two daily peaks, though with a different shape than "mail.yahoo.com.vn".

There are many papers in the literature about modeling periodic time series data [3, 10, 6, 2]. This application has a number of unusual properties that make applying those algorithms difficult or impossible. We describe those properties in Section 2.

We describe a new algorithm in Section 3. This is a general purpose incremental Poisson regression that can be applied to a wide variety of data. As with standard Poisson regression the framework is quite flexible and capable of supporting periodic components, trend terms, indicator variables, and polynomials. However, unlike standard techniques, it supports incremental updates which makes it particularly well suited for modeling a continuous stream of time series data.

[*]MIT
[†]Google
[‡]Google
[§]Google

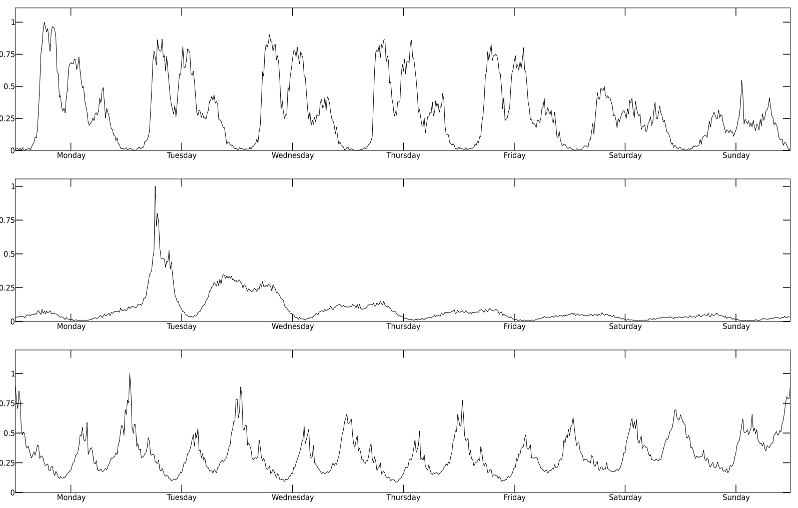

Figure 1: One week of data for three three different queries, "mail.yahoo.com.vn", "the bachelorette", and "tv guide" from top to bottom. All data is for the week of June 5, 2011. Data is bucketed into 15 minute buckets. The y-axis of each plot is scaled so that 1.0 corresponds to the maximum query rate.

The algorithm developed in Section 3 bears some resemblance to Kalman Filters. We discuss the similarities and differences between our technique and the Kalman Filter in Section 4.

We discuss results in Section 5, and discuss possible improvements to the algorithms in Section 6.

2. Data Properties

The raw data for each query consist of times the query occurs; $S = [t_1, t_2, \ldots]$. In principle this is an infinite sequence of events, though in practice we only observe the queries up to the current time T, and cannot store the times of all earlier queries.

Important properties of these data include:

1. This is an online application, to be updated with new data, indefinitely (unless users stop typing a query string).
2. There are millions of time series, one for each unique query string.
3. Different queries have very different patterns.
4. Many queries exhibit regular, dramatic rate changes of more than an order of magnitude in a very short time span, often less than an hour.
5. The vast majority of queries exhibit a pattern that repeats weekly.
6. The periodic pattern for a query can change but it usually changes slowly.
7. When the query pattern changes abruptly it is impossible to tell from the data if the change is temporary or permanent.
8. In our experience, the number of queries can be modeled by a non-homogeneous Poisson process, possibly with over-dispersion.

Modeling a changing, infinitely long time series requires an algorithm that can be updated on-line. There does not appear to be any known algorithm that can maintain accurate models for the number of queries we work with that is fast enough to keep up with the data stream on a single CPU. Since the model can be fit for each query's data independently the data can be separated by query string and each stream handled separately. It is therefore possible to route each query to a different machine and thus get inexpensive parallelism by leveraging a cluster of compute nodes [1]. Since compute jobs might be relocated or machines might fail in such clusters it is necessary to frequently serialize the models to permanent storage so that recovery is possible. Thus, it must be possible to store a small amount of data that fully specifies the model and all the information necessary to update the model when new data arrives. Similarly, although compute clusters make large scale computation less expensive, each core still costs money. It is desirable to minimize the resources required to update models and obtain predictions. These considerations rule out most nonparametric techniques as these generally require storing a large portion of the historical data [13].

Given the source of the data is it reasonable to assume that the data is generated by a non-homogeneous Poisson process [13]. Experimental evidence bears this out though we find that some queries are better modeled by an over-dispersed Poisson.

For our modeling, we transform the series of times of queries $[t_1, t_2, \ldots]$, where t_i is the time at which the i^{th} instance of the query is observed, into a time series of counts, $[y_1, y_2, \ldots,]$ where y_i is the number of times the query was observed in bucket (time interval) i, e.g. y_1 corresponds to the first bucket, y_2 the second, etc. Depending on the query volume, buckets could be say one second long, one minute, or one hour.

2.1 Desirable Characteristics for Models

One use for the models described here is to detect spikes in queries as quickly as possible. For example, if the query rate for "bin laden" is suddenly higher than its historical pattern the search engine can take action like presenting news articles to users. The ability to do this seconds after an event is valuable. Thus it is desirable for the fitted model to closely track regular volume shifts so that it is possible to accurately discriminate usual and unusual rate shifts. This precludes many common modeling techniques that treat recurring events discretely instead of smoothly. For example, simple ARMA models that have terms for the hours of the week [2] admit patterns that recur periodically and can thus predict that the volume on a Monday at noon might differ dramatically from the volume on a Monday at 1:00 pm. However, since this is due to the influence of a single hourly term the model will predict a large step function at the hour boundary. Fast spike detection would be difficult with this model. Getting flexible, smooth behavior out of such models requires an enormous number of terms, which violates our storage constraints and is prone to over-fitting.

Most queries exhibit patterns that repeat weekly and change very little from week to week. Some queries exhibit monthly or yearly patterns, but the vast majority have a weekly pattern. Occasionally a query will experience a brief change in pattern, as when a news story breaks or a crossword puzzle clue relates to the query, but these are generally short term disruptions (spikes) after which the time series resumes its normal pattern. ARMA, ARIMA, and common parameterizations of the Kalman Filter [6] use data from the recent past to make predictions about the future and are thus prone to over-fit short-duration spikes. Instead, a successful model must incorporate knowledge of many cycles of the pattern so that short duration shifts don't unduly influence predictions.

Predictions should remain stable in the face of short-term spikes, but query patterns do shift over time and the model must be able to evolve with the pattern. For example, queries

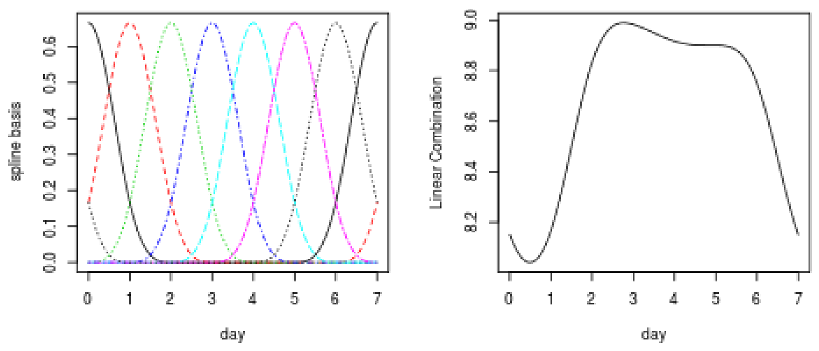

Figure 2: Periodic basis functions for days of the week, and an example linear combination. The corresponding weekly pattern is $\exp(\text{linear combination})$.

for "android" were rare until Google released the Android operating system. When the operating system was announced the volume of "android" queries increased dramatically and the increased interest was sustained. On the other hand queries for "leesburg va" increased dramatically after a magazine article listed it as one of the top five places to live but within hours the volume returned to normal. Other queries fall in between. For example, queries for "susan boyle" experienced a huge spike after her appearance on the television show "Britain's Got Talent" and interest slowly declined over the next several months. It is not possible to tell if a query has permanently changed its pattern or is experiencing a brief, one-time shift just by looking at the time series data. Having observed data that differs from the historical pattern it is difficult to know what prediction to make though any reasonable prediction probably lies between the historical values and the observed anomaly.

The number of queries is continually changing over time; new queries become popular, and some old ones decline. The system must be able to handle such changes. And given the number of queries Google serves, the system must be computationally efficient.

2.2 Incremental Poisson Regression Properties

Our basic approach is a Poisson regression algorithm that puts more weight on recent observations. We model daily and weekly patterns using two periodic B-splines, one with a period of one day and one with a period of one week. Our model is multiplicative so predictions are the product of the daily and weekly components. The splines allow us to fit flexible, smooth curves to the data [4] that match most observed weekly patterns reasonably well. Figure 2 shows seven basis functions, which combine to form the weekly pattern.

Unfortunately it is not possible to simply store several periods of data and periodically run the regression as the cost of storing the data would be prohibitive. In Section 3 we describe an incremental Poisson regression algorithm that compactly maintains a representation of the historical data; this avoids the cost of storing many periods of historical data while maintaining most of the relevant information in that data. We downweight old data to achieve a balance between quickly adapting to changes and remaining stable in the face of short term shifts.

The framework described in Section 3 treats the periodicity as a parameter, so could model time series with any periodicity. There are techniques for automatically determining the periodicity of a time series [2]. Such techniques could be used with the framework presented here but will not be addressed in this paper. Furthermore, while we use this algorithm with periodic B-splines, it is a general regression algorithm that allows other models.

Many powerful modeling techniques require data with a Gaussian distribution. Cao [3] use a variance stabilizing transform on count data to take advantage of Gaussian methods. Unfortunately, for many queries, the expected rate is extremely low which makes it difficult to transform the data such that an ordinary least squares approach would produce satisfactory results [12]. An alternative is to re-bucket the low rate data so as to increase the Poisson parameter and thus make variance stabilization feasible. There are two disadvantages to this approach. First, many queries have highly variable rates so the query has an extremely low rate for part of its cycle but a much higher rate for the rest of the cycle. Re-bucketing the data such that counts are sufficiently large when the query rate is at its minimum would require buckets so large that key variations would be lost during times when the query rate is higher. Second, we would like to be able to use these models to detect deviations from expected rates as quickly as possible, and using longer buckets would slow that detection. In order to avoid these difficulties, we do not transform the data. Instead we develop an incremental regression technique in Section 3 that is appropriate for data with a Poisson distribution, possibly over-dispersed.

3. Incremental Poisson Regression

In this section we develop an updating algorithm for Poisson regression. Let $[y_1, y_2, \ldots]$ represent the number of observed queries within fixed-time buckets, say one second. Nominally we assume that the observations follow a Poisson distribution with rate given by a function $f(t)$

$$y_t \sim \text{Pois}\left(f(t)\right). \tag{1}$$

In practice we only require that the variance of y_t is proportional to the expected rate, $f(t)$, so over-dispersion or under-dispersion presents no problem. Over-dispersion could occur due to due to social behavior, for example groups of friends issuing the same query; under-dispersion could occur if there is a relatively small population of "regulars", people who issue a particular query frequently but at most once within a time bucket.

We consider $f(t)$ of the form

$$f(t) = e^{g(t)}$$

where

$$g(t) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_p x_p,$$

$x_i = x_i(t)$ is a function of time, and β_i is a coefficient whose value to be determined by the data (the β_i are known as *weights* in the machine learning literature). We use B-splines—the $x(t)$ functions are a basis for piece-wise polynomials, periodic over periods of one day or week. These splines provide the flexibility necessary to accurately model the variety of patterns observed in the query stream while still providing a compact model that is inexpensive to store. We normally use 30 basis functions (intercept, $7-1$ daily terms, and $24-1$ hourly terms). However, the general framework could incorporate trend terms, indicator variables, polynomials, and other regression functions in place of splines.

3.1 Incremental Least-Squares Regression

We begin with an incremental least-squares regression model. This material is not new, but is included here to help understand the incremental Poisson regression models presented later.

We begin with the simple case that all n data points in the time series are available at once, with no downweighting for old observations. Let $y = [y_1, \, y_2, \, \ldots, \, y_n]$ as the vector of observed outcomes, and

$$X = \begin{bmatrix} X_{1\bullet} \\ X_{2\bullet} \\ \vdots \\ X_{n\bullet} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ & \cdots & & \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}$$

be the $n \times p$ design matrix. Here $X_{i\bullet}$ is the i^{th} row of X (we also use $X_{\bullet j}$ to denote the j^{th} column of X). This can be solved with the standard least squares solution [13]

$$\hat{\beta} = \left(X^T X\right)^{-1} X^T y. \tag{2}$$

There are more numerically-accurate alternatives to this equation, though they are not needed in our application due to the low correlations between columns of X. For later use, call $X^T X$ and $X^T y$ the *cross-product matrices*.

This solution admits a wide variety of regression functions. If x_{ij} is the value of the j^{th} periodic basis spline at time i then the above fits a smoothing spline to the data as desired [4, 3]. Furthermore, we can introduce weights to this scheme so that older data points have less influence on the result than more recent observations. If we replace $X_{i\bullet}$ with $\sqrt{\alpha_i} X_{i\bullet}$ and y_i with $\alpha_i y_i$ for some $0 < \alpha_i \leq 1$ we can adjust the weight given to observation i. If $\alpha_1 \leq \alpha_2 \leq \ldots \leq \alpha_n$ then less weight is given to older data.

Now consider the case where not all data is available at once; instead we see data in blocks. The first block, B_1 includes data from time $t = 1, \ldots, T_1$, B_2 includes $t = T_1 + 1, \ldots, T_2$, and in general B_k goes from $t = T_k + 1, \ldots, T_k$.

Let $y_{B_k} = \left[y_{T_{k-1}+1}, \, y_{T_{k-1}+2}, \, \ldots, \, y_{T_k}\right]$ be the observations for block k, and X_{B_k} the corresponding rows of the design matrix. For convenience, define cumulative blocks C_k to include data from $t = 1$ to $t = T_k$, with corresponding observations $y_{C_k} = [y_1, \, y_2, \, \ldots, \, y_{T_k}]$ and X_{C_k} the corresponding rows of the design matrix.

The estimate for β using Equation (2) after the kth block of data uses the cross-product matrices to that point: $\hat{\beta}_{C_k} = \left(X_{C_k}^T X_{C_k}\right)^{-1} X_{C_k}^T y_{C_k}$.

Those cross-product matrices can be calculated as a sum across blocks:

$$\begin{aligned} X_{C_k}^T X_{C_2} &= X_{B_1}^T X_{B_1} + \ldots + X_{B_k}^T X_{B_k} \\ &= X_{C_{k-1}}^T X_{C_{k-1}} + X_{B_k}^T X_{B_k} \end{aligned}$$

and

$$\begin{aligned} X_{C_k}^T y_{C_k} &= X_{B_1}^T y_{B_1} + \ldots + X_{B_k}^T y_{B_k} \\ &= X_{C_{k-1}}^T y_{C_{k-1}} + X_{B_k}^T y_{B_k}. \end{aligned}$$

After the kth block, we need store only the cross-product matrices to that point, which requires storage $O(p^2)$, independent of the number of observations to that point.

Furthermore we can add downweighting to this scheme. We implement a block exponential downweighting of older data. Let

$$\begin{aligned} S_{C_k} &= \alpha^{k-1} X_{B_1}^T X_{B_1} + \alpha^{k-2} X_{B_2}^T X_{B_2} + \ldots + \alpha X_{B_k}^T X_{B_k} \\ &= \alpha S_{C_{k-1}} + X_{B_k}^T X_{B_k} \end{aligned}$$

Algorithm 1 Incremental Least-Squares Regression

Initialize S_{C_0} as a $p \times p$ matrix of zeroes, and r_{C_0} as a $p \times 1$ matrix of zeroes. Given S_{C_k}, r_{C_k}, $X_{B_{k+1}}$, and $y_{B_{k+1}}$, compute $\hat{\beta}_{C_{k+1}}$, $S_{C_{k+1}}$, and $r_{C_{k+1}}$:

```
1. $S_{C_{k+1}} \leftarrow \alpha S_{C+k} + X_{B_{k+1}}^T X_{B_{k+1}}$
2. $r_{C_{k+1}} \leftarrow \alpha r_{C_k} + X_{B_{k+1}}^T y_{B_{k+1}}$
3. $\hat{\beta}_{C_{k+1}} \leftarrow \left(S_{C_{k+1}}\right)^{-1} r_{C_{k+1}}$
```

and

$$\begin{aligned} r_{C_k} &= \alpha^{k-1} X_{B_1}^T y_{B_1} + \alpha^{k-2} X_{B_2}^T y_{B_2} + \ldots + X_{B_k}^T y_{B_k} \\ &= \alpha r_{C_{k-1}} + X_{B_k}^T y_{B_k}. \end{aligned}$$

Then $\hat{\beta}_{C_k} = S_{C_k}^{-1} r_{C_k}$ is an incremental regression update with block exponential down-weighting of older data. This only requires storing $p \times p$ symmetric matrix $S_{C_{k-1}}$ and the $1 \times p$ vector $r_{C_{k-1}}$ to carry information from all previous blocks. This yields Algorithm 1.

3.2 Poisson Regression

For count data, instead of the least-squares model just described, we use a Poisson model. Instead of predictions $X\beta$, which can be negative, we use $e^{X\beta}$. This also makes the model multiplicative instead of additive. For example, a model with components for the daily and the weekly effects $e^{X\beta} = e^{X_d\beta_d + X_w\beta_w} = e^{X_d\beta_d} e^{X_w\beta_w}$, where the subscripts d and w denote the daily and weekly portion of the model respectively. This agrees with our intuition that such effects are multiplicative rather than additive. For example, Wednesday's traffic being double the normal rate (with each minute double the normal rate) seems more likely to be correct than Wednesday's traffic being 5,000 more queries than normal (with an equal increase of $5000/24/60$ for each minute of the day).

The Poisson model also assumes that the variance of observations is proportional to the mean; this allows for larger random fluctuations during high-traffic periods, so that the model does not try to over-fit high-traffic periods at the expense of low-traffic periods.

One way to fit the model is to maximize the likelihood function, or equivalently the log-likelihood. Given an estimate $\hat{\beta}$, the estimated mean for the i^{th} observation is $\hat{\lambda}_i = \exp(X_{i\bullet}\hat{\beta})$. For n observations $y = [y_1,\, y_2,\, \ldots,\, y_n]$, the log likelihood is

$$\begin{aligned} \mathcal{L}\left(\hat{\beta}|X,\, y\right) &= \sum_{i=1}^{n} y_i \log(\hat{\lambda}_i) - \hat{\lambda}_i - \log(y_i!) && (3) \\ &= \sum_{i=1}^{n} y_i X_{i\bullet}\hat{\beta} - \exp(X_{i\bullet}\hat{\beta}) - \log(y_i!) && (4) \end{aligned}$$

We maximize this by taking partial derivatives and using a Newton–Raphson procedure to find the root of the derivative.

$$\begin{aligned} \frac{\partial \mathcal{L}\left(\hat{\beta}|X,\, y\right)}{\partial \hat{\beta}_j} &= \sum_{i=1}^{n} (y_i - \hat{\lambda}_i) x_{ij} \\ &= X_{\bullet j}^T (y - \hat{\lambda}) \end{aligned}$$

Algorithm 2 Iteratively Re-weighted Least Squares (IRLS)

```
Given observed data $y$, design matrix $X$, and an initial estimate for $\hat{\beta}$, to compute a better estimate for $\hat{\beta}$:

1. Repeat until convergence:
    (a) $\hat{\lambda} \leftarrow e^{X\hat{\beta}}$
    (b) $\hat{\beta} \leftarrow \hat{\beta} - H^{-1}(\hat{\beta})G(\hat{\beta}) = \hat{\beta} + \left(X^T\hat{\Lambda}X\right)^{-1} X^T(y - \hat{\lambda})$
```

and

$$\frac{\partial^2 \mathcal{L}\left(\hat{\beta}|X,\, y\right)}{\partial\hat{\beta}_j \partial\hat{\beta}_k} = -\sum_{i=1}^{n} x_{ij}x_{ik}\hat{\lambda}_i.$$

For notational convenience, let $\hat{\Lambda} = \operatorname{diag}\left(\hat{\lambda}\right)$ be the diagonal matrix with $\hat{\Lambda}_{i,i} = \lambda_i$. (In practice we do not create this matrix; instead we perform matrix multiplications that involve $\hat{\Lambda}$ using element-wise multiplications with the corresponding values of $\hat{\lambda}$.) The gradient is

$$G(\hat{\beta}) = \nabla\mathcal{L}\left(\hat{\beta}|X,\, y\right) = X^T(y - \hat{\lambda})$$

and Hessian is

$$H(\hat{\beta}) = -X^T\hat{\Lambda}X.$$

This is invertible if the columns of X are linearly independent (as they are in our application). A Newton–Raphson step is

$$\hat{\beta} \leftarrow \hat{\beta} - H^{-1}(\hat{\beta})G(\hat{\beta}) = \hat{\beta} + (X^T\hat{\Lambda}X)^{-1}X^T(y - \hat{\lambda}) \tag{5}$$

Using Newton–Raphson steps until convergence gives Algorithm 2, the well known *Iteratively Re-weighted Least Squares* (IRLS) solution for Generalized Linear Models [5].

If the columns of X are linearly independent then $\mathcal{L}\left(\hat{\beta}|X,\, y\right)$ is a convex function with a single maximum, where the gradient is zero. We thus need not worry about converging to a local minimum [5]. If $y_i = 0$ for some i then the maximum could occur when one or more of the $\hat{\beta}$ are infinite.

It is interesting to look at the behavior of the algorithm in the simple case of a single observation and single column in the design matrix with $x_{11} = 1$; then $\hat{\lambda} = e^{\hat{\beta}}$ and $\hat{\beta} \leftarrow \hat{\beta} + (y - e^{\hat{\beta}})/e^{\hat{\beta}}$. When $\hat{\beta}$ (and $\hat{\lambda}$) is too small in one iteration, it is too large in the next iteration. When too large in one iteration, it smaller but still too large in the next iteration, and with successive iterations converges downward to $\hat{\lambda} = y$. But if stopped before convergence, the estimates tend to be too large.

3.3 Incremental Poisson Regression

We begin by describing two unsuccessful attempts, whose failures motivate our current algorithm. The first attempt does a one-step upgrade, using gradient and Hessian terms computed using the old value of β. The second attempt iterates, letting parts of the gradient and Hessian that correspond to new data depend on the current value of β, but with the parts corresponding to the old data not changing as β changes. Our current algorithm modifies the second attempt by also letting the part of the gradient that depends on old data change as β changes.

Algorithm 3 Incremental Poisson Regression, by One-Step Updating

```
Initialize $\hat{\beta}_{C_1}$ by iterating Algorithm 2 to convergence for the first block of data, and let
$Q_{C_1} = X_{C_1}^T \Lambda_{C_1} X_{C_1}$ and $\gamma_{C_1} = 0$, Then, for subsequent blocks of data,
    1. $\lambda_{B_{k+1}} = e^{X_{B_{k+1}}^T \hat{\beta}_{C_k}}$
    2. $Q_{C_{k+1}} = \alpha Q_{C_k} + X_{B_{k+1}}^T \Lambda_{B_{k+1}} X_{B_{k+1}}$
    3. $\gamma_{C_{k+1}} = 0 + X_{B_{k+1}}^T (y_{B_{k+1}} - \lambda_{B_{k+1}})$
The $Q$ terms correspond to negative Hessian terms and $\gamma$ terms to gradient terms.
```

3.3.1 One-Step Updating

Recall that in the incremental least-squares algorithm, the cross-product matrices are sums across blocks. Similarly, in the IRLS (Newton-Raphson) algorithm for Poisson regression, the key terms are sums across blocks. Let Q indicate the negative Hessian, then

$$X^T \hat{\Lambda} X = Q_{C_{k+1}} = Q_{C_k} + Q_{B_{k+1}}$$

where

$$\begin{aligned} Q_{C_k} &= X_{C_k}^T \Lambda_{C_k} X_{C_k} \\ Q_{B_{k+1}} &= X_{B_{k+1}}^T \Lambda_{B_{k+1}} X_{B_{k+1}} \end{aligned}$$

correspond to old and new data (where $\lambda_{C_k} = e^{X_{C_k}\hat{\beta}}$ and $\lambda_{B_{k+1}} = e^{X_{B_{k+1}}\hat{\beta}}$ depend on a common $\hat{\beta}$ together with the data from their respective blocks). Similarly, the second term, the gradient, can be decomposed as

$$X^T(y - \hat{\lambda}) = \gamma_{C_{k+1}} = \gamma_{C_k} + \gamma_{B_{k+1}}$$

where

$$\begin{aligned} \gamma_{C_k} &= X_{C_k}^T (y_{C_k} - \lambda_{C_k}) \\ \gamma_{B_{k+1}} &= X_{B_{k+1}}^T (y_{B_{k+1}} - \lambda_{B_{k+1}}) \end{aligned}$$

Now suppose that $\hat{\beta}$ is equal to the Poisson regression estimate through block C_k, so that the gradient $\gamma_{C_k} = 0$. Then a single iteration of IRLS would be $\hat{\beta} \leftarrow \hat{\beta} + (Q_{C_k} + Q_{B_{k+1}})^{-1}\gamma_{B_{k+1}}$. This idea, together with downweighting previous blocks, results in a one-step updating approach, Algorithm 3.

Unfortunately, this one-step updating algorithm is not accurate in our application. A different look at the algorithm may illustrate why, and hint at improvements. In somewhat less formal notation, let $\gamma_{\text{data}}(\beta) = \gamma_{\text{old}}(\beta) + \gamma_{\text{new}}(\beta)$ and $Q_{\text{data}}(\beta) = Q_{\text{old}}(\beta) + Q_{\text{new}}(\beta)$ be the gradient and negative Hessian for the data, expressed as a sum across the old and new data, evaluated at β. The one-step update corresponds to

$$\hat{\beta}_{\text{new}} = \hat{\beta}_{\text{old}} + \left(\alpha Q_{\text{old}}(\hat{\beta}_{\text{old}}) + Q_{\text{new}}(\hat{\beta}_{\text{old}})\right)^{-1} \left(0 + \gamma_{\text{new}}(\hat{\beta}_{\text{old}})\right)$$

where $\hat{\beta}_{\text{old}}$ is the previous estimate and $\hat{\beta}_{\text{new}}$ is the new estimate. Compared to the usual IRLS algorithm, this (1) fixes $\hat{\beta}_{\text{old}}$ on the right side rather than updating it iteratively, and (2) assumes that $\gamma_{\text{old}}(\hat{\beta}_{\text{old}}) = 0$, which may not be true after the first update.

3.3.2 Iterative Updating, Changing Terms For New Data But Not Old Data

The next version attempts to get better estimates by iterating until convergence, and letting the gradient and Hessian terms for the new data depend on the current value of $\hat{\beta}$. The gradient and Hessian terms for the old data are unchanged, because that data is no longer available. This algorithm starts with $\hat{\beta}_{\text{new}} = \hat{\beta}_{\text{old}}$, then iterates

$$\hat{\beta}_{\text{new}} \leftarrow \hat{\beta}_{\text{new}} + \left(\alpha Q_{\text{old}}(\hat{\beta}_{\text{old}}) + Q_{\text{new}}(\hat{\beta}_{\text{new}})\right)^{-1} \left(0 + \gamma_{\text{new}}(\hat{\beta}_{\text{new}})\right)$$

This is a complete failure. It converges when $\gamma_{\text{new}}(\hat{\beta}_{\text{new}}) = 0$, so the algorithm effectively gives zero weight to old observations.

This algorithm implicitly assumes that $\gamma_{\text{old}}(\hat{\beta}_{\text{new}}) = 0$, i.e. that the gradient for the old data is the same at $\hat{\beta}_{\text{old}}$ and $\hat{\beta}_{\text{new}}$. This is not reasonable.

3.3.3 Iterative Updating, Changing Terms For New Data And Old Gradient

Our proposed solution is to update the gradient term for old data using the relationship between the gradient and Hessian—the Hessian is the matrix of partial derivatives of the gradient. We use the approximation $\gamma_{\text{old}}(\hat{\beta}_{\text{new}}) \approx \gamma_{\text{old}}(\hat{\beta}_{\text{old}}) + H_{\text{old}}(\hat{\beta}_{\text{new}} - \hat{\beta}_{\text{old}}) \approx 0 + H_{\text{old}}(\hat{\beta}_{\text{new}} - \hat{\beta}_{\text{old}})$.

This algorithm starts with $\hat{\beta}_{\text{new}} = \hat{\beta}_{\text{old}}$, then iterates

$$\hat{\beta}_{\text{new}} \leftarrow \hat{\beta}_{\text{new}} + \left(\alpha Q_{\text{old}} + Q_{\text{new}}(\hat{\beta}_{\text{new}})\right)^{-1} \left(-\alpha Q_{\text{old}}(\hat{\beta}_{\text{new}} - \hat{\beta}_{\text{old}}) + \gamma_{\text{new}}(\hat{\beta}_{\text{new}})\right)$$

More formally, we approximate the gradient as

$$\gamma_{C_{k+1}} \approx 0 + \gamma_{B_{k+1}} + H_{C_k} \Delta\hat{\beta}$$

where $\Delta\hat{\beta}$ is the change in $\hat{\beta}$ from its value at the end of the previous iteration. Thus after processing the first block of data we can store $Q_{C_1} = -H_{C_1}$. This allows us to process the second block of data and get an excellent fit as we have the exact derivative for γ_{C_1} with respect to β. Furthermore, combining Q_{C_1} with Q_{B_2} yields a reasonable approximation to Q_{C_2}. Thus, storing $\widetilde{Q}_{C_2} = \alpha Q_{C_1} + Q_{B_2}$, which is an approximation to the true value of Q_{C_2} (with downweighting) allows the algorithm to proceed iteratively. Note that $\widetilde{Q}_{C_2}$ is an approximation using approximate values for $\lambda_{B_1} = \lambda_{C_1}$ because the predictions changed when the second block of data is processed. However, provided the predictions didn't change dramatically, this should give a decent approximation. Furthermore, the errors are smaller for newer data. And we downweight older data, so errors tend not to compound. Algorithm 4 describes this procedure in detail.

The algorithm corresponds to Newton-Raphson steps for an objective function that consists of the log-likelihood for the new data plus a quadratic approximation to the down-weighted log-likelihood for the old data. We believe that convergence is guaranteed (though have not proved this). Section 5 presents results on real data that indicate the effectiveness of this procedure.

4. Kalman Filters

One popular technique for on-line learning of parametric models is the Kalman Filter. The basic Kalman Filter model is that y_t is a linear function of some unobservable state vector, β_t. The state vector is not constant and also evolves with time. Specifically $E(\beta_t)$, the

Algorithm 4 Incremental Poisson Regression, iteratively changing terms for new data and old gradient based on current estimates for β.

```
Given $\hat{\beta}_{C_k}$, $\tilde{Q}_{C_k}$, $y_{B_{k+1}}$, and $X_{B_{k+1}}$ compute $\tilde{Q}_{C_{k+1}}$, and $\hat{\beta}_{C_{k+1}}$
  1. $\hat{\beta}_{C_{k+1}} \leftarrow \hat{\beta}_{C_k}$
  2. Repeat until convergence:
      (a) $\lambda_{B_{k+1}} \leftarrow e^{X_{B_{k+1}}\hat{\beta}_{C_{k+1}}}$
      (b) $Q_{B_{k+1}} \leftarrow X_{B_{k+1}}^T \Lambda_{B_{k+1}} X_{B_{k+1}}$
      (c) $\tilde{Q}_{C_{k+1}} \leftarrow \alpha \tilde{Q}_{C_k} + Q_{B_{k+1}}$
      (d) $\Delta\hat{\beta} \leftarrow \hat{\beta}_{C_{k+1}} - \hat{\beta}_{C_k}$
      (e) $\gamma_{C_{k+1}} \leftarrow X_{B_{k+1}}^T \left(y_{B_{k+1}} - \lambda_{B_{k+1}}\right) - \alpha \widetilde{Q}_{C_k} \Delta\hat{\beta}$
      (f) $\hat{\beta}_{C_{k+1}} \leftarrow \hat{\beta}_{C_{k+1}} + Q_{C_{k+1}}^{-1} \gamma_{C_{k+1}}$
The $Q$ terms correspond to negative Hessian terms and $\gamma$ terms to gradient terms.
```

expected value of β_t, is assumed to be a linear function of the previous state β_{t-1} plus some random noise. These assumptions yield the following system of equations:

$$\begin{aligned} y_t &= X_t\beta_t + v_t \\ \beta_t &= G_t\beta_{t-1} + w_t \end{aligned} \tag{6}$$

where v_t and w_t are both random variables with zero mean. If we further assume that both v_t and w_t have a Gaussian distribution, $v_t \sim N(0,\, V_t)$ and $w_t \sim N(0,\, W_t)$, the standard Kalman Filter algorithm described in [6] can then be applied. Assuming an estimate for $\beta_{t-1} \sim N\left(\hat{\beta}_{t-1},\, P_{t-1}\right)$, where P_{t-1}, is the covariance matrix for β_{t-1}, and a single new observation, y_t, the update is as follows:

1. $r_t = y_t - X_t\beta_{t-1}$
2. $S_t = X_t W_{t-1} X_t^T + V_t$
3. $Q_t = G_t P_{t-1} G_t^T + W_t$
4. $K_t = Q_t X_t^T S_t^{-1}$
5. $\hat{\beta}_t = \hat{\beta}_{t-1} + K_t r_t$
6. $P_t = (I - K_t X_t) P_{t-1}$

For the application described here G_t would always be the identity matrix as we have no a priori knowledge about how $\hat{\beta}$ might change over time. As noted in [11] the Kalman Filter is equivalent to a Bayesian regression where the prior distribution for β_t is given by $\beta_t \sim N(G_t\beta_{t-1},\, W_t)$ where W_t is the covariance matrix of w_t. It is not clear what values to use for the prior distribution before we have observed any data for a query. One common choice in such situations is to use an improper prior. [9] show that this choice for the prior results in an algorithm that produces results identical to standard linear regression. In other words, if G_t is set to be the identity and n data points are processed sequentially using the standard Kalman Filter algorithm beginning with an improper prior the resulting estimate for β is given by $\hat{\beta} = \left(X^TX\right)^{-1} X^Ty$. Thus the Kalman Filter is equivalent to Algorithm 1 without downweighting. In order to place more weight on new data the Kalman Filter approach can be modified so that W_t is constrained so that its entries never become too small thus ensuring that the prior distribution for β_t maintains enough variability to allow for continued evolution. This has roughly the same effect as downweighting older data in Algorithm 1.

Algorithm 1 and the Kalman Filter produce essentially identical results, but we prefer Algorithm 1. The storage requirements of the Kalman Filter are β_t and the covariance matrix of β_t, the same as Algorithm 1. If we update $\hat{\beta}$ with each new observation the time complexity of the Kalman filter is dominated by the time to compute P_t which requires $O\left(p^3\right)$ operations per update. This too is the same as Algorithm 1, which is dominated by the time to compute $\left(X^TX\right)^{-1}$. However, The Kalman Filter approach requires many more matrix multiplications than Algorithm 1 so the constants are worse. More importantly, it is possible to trade storage for speed with Algorithm 1 by processing the data in batches, while this is not possible with the Kalman Filter. Specifically, if n observations are processed in batches of size m, $m > p$, the time for Algorithm 1 to process a single batch is dominated by the time to compute X^TX which is $O\left(mp^2\right)$ though each batch still requires the inversion of the $p \times p$ matrix X^TX which is $O\left(p^3\right)$. Thus, since there will be $\frac{n}{m}$ batches, the time to process all n observations is $O\left(\frac{n}{m}\left(mp^2 + p^3\right)\right) = O\left(np^2 + \frac{n}{m}p^3\right)$. Since m is a constant and the data stream is infinitely long the total complexity is still $O\left(np^3\right)$, but as m approaches n the complexity reduces to $O\left(np^2\right)$. Thus increasing m can significantly reduce the amount of CPU utilized in any fixed period of time. On the other hand treating y_t as a vector of observations in Equation (6) actually slows down the filter so it is better to process each data point separately. Thus the time to process all n observation with a Kalman Filter is $O\left(np^3\right)$. In addition to the time savings Algorithm 1 is easier to understand and it requires less code to implement.

Above we argued that Algorithm 1 is preferable to the Kalman Filter approach for Gaussian data when $G_t = I$. In our case the data is not Gaussian distributed and, as argued in Section 2, it is not feasible to transform the data so that it is. There are proposals for applying the general ideas of a Kalman Filter to non-Gaussian data [7, 8] which, like Algorithm 3, are approximate and iterative. The comparison between these techniques and Algorithm 3 is similar to the Gaussian case; Algorithm 3 produces the same results, is faster, easier to understand, and requires less code to implement.

5. Results

We have been running Algorithm 4 to maintain a model of every web query received by Google.com as long as the rate of the query exceeds one query per hour in the past twenty four hours. Queries whose rate fall below one per hour have their model garbage collected and any new observations of that query start a new model. The deployed model consists of two periodic splines: one the repeats daily and one that repeats weekly. The daily spline has twenty four equally spaced knots, one per hour, and the weekly spline has seven equally spaced knots, one per day. With period splines, the number of basis functions equals the number of knots, minus one when used with an intercept.

Figure 3 shows the results on the three queries from Figure 1. In Figure 3 each model was trained on three weeks of data. The data are provided to the model bucketed by minute with a batch size of five hours. The model was trained with no downweighting ($\alpha = 1$). The plot shows the actual data bucketed into fifteen minute buckets. Each data point is the average of the value for that bucket over the three weeks of data. The model shown is the final learned model after all three weeks of data have been observed.

In general we get an excellent fit to the complex queries shown in the figure though there are a few places where the sparse, fixed knots in the model do not provide sufficient flexibility to match the query. For example, the pattern on Saturday for “mail.yahoo.vn” is

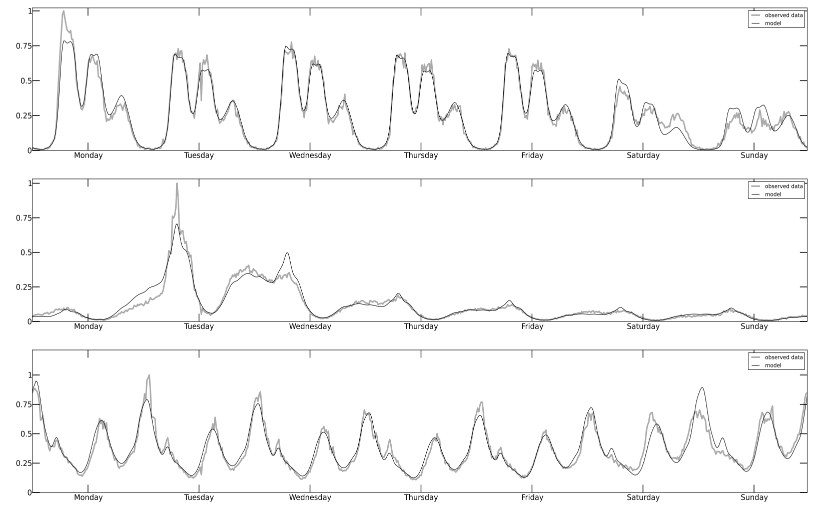

Figure 3: One full cycle for three different queries. From top to bottom the plots correspond to the queries “mail.yahoo.com.vn”, “the bachelorette”, and “tv guide”. Each data point is the average of three week’s worth of data starting on June 5, 2011. Data is bucketed into 15 minute buckets. The y-axis of each plot has been scaled so that 1.0 corresponds to the maximum query rate.

significantly different from the other days. While the weekly part of the spline can correct for some of this the resulting model predicts too low a value for the third “hump” in the day. We are considering dynamically adding and removing knots to improve predictions.

As noted above, the model consumes data in five hour batches. The model is able to learn the entire weekly pattern for the query despite the fact that the training data never consists of more than five hours of data and the old data is not stored.

Figure 4 shows how the model evolves over time. In each sub-figure the actual data, bucketed into 15 minute buckets, is plotted in light gray and the solid, thin black line indicates the prediction. In the top sub-figure the model has only observed the data from midnight on Monday until noon on Monday (indicated by the dashed vertical line). The prediction plotted for all time points is the prediction after the model has observed all data from the left-edge of the plot up to the vertical, dashed line. Not surprisingly the predictions to the right of the dashed line are terrible; the model has observed no data past this point. In the second sub-figure of Figure 4 the dashed line has moved to midnight on Tuesday indicating that the model has now seen data, available only in 5 hour batches, between midnight on Monday (the left-hand edge of the plot) and midnight on Tuesday. Since Algorithm 4 accounts for older data, the new information that arrived between noon on Monday and midnight on Tuesday did not adversely affect the earlier predictions even though that data was not stored. Similarly, the third sub-figure show the predictions after data up to the end of the day on Wednesday has been observed. Note that the prediction for the third sharp peak on Monday has changed slightly with the addition of new data. This is because the model incorporates a single daily spline (in addition to the weekly spline) that is shared by all the days of the week so the prediction for Monday is informed, in part, by the data observed on Tuesday and Wednesday. Appropriately, the prediction is now roughly the average of what was observed at this time on the three days of observed data.

Algorithm 4 is very efficient. With twenty four knots for the daily component, seven knots for the weekly component, one constant offset term, and two components removed to prevent over-specification our model has only $p = 30$ components. Thus we store only

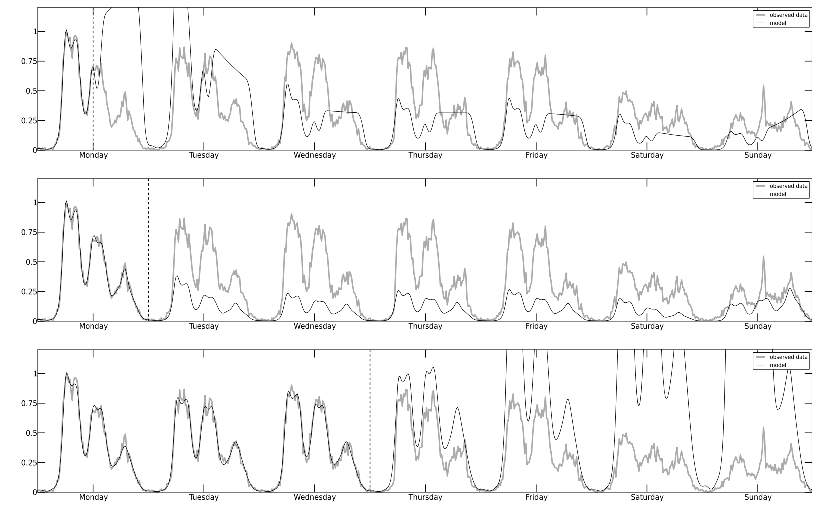

Figure 4: Shows the model evolving on data starting on June 5, 2011. The model has only observed data from the left edge of the plot up to the vertical dashed line in each sub-figure.

$p(p+1)/2 + 2p = 515$ doubles to maintain all of the historical data. In addition, updates are performed in five hour batches so we must store up to $5 \times 60 = 300$ integers of data for the next update.

The algorithm's CPU requirements are low enough that we are able to maintain models for many millions of queries using fewer than 100 commodity CPU cores.

6. Discussion

As demonstrated, Algorithm 4 is capable of maintaining accurate predictive models for millions of unique query strings using relatively inexpensive commodity hardware. Our experience with the algorithm has been excellent as we have found its predictions to generally be accurate and the performance adequate.

However, there are several areas in which in could be improved. In the current implementation the locations of the spline knots are fixed. For most queries this means that we have more flexibility than necessary which results in wasted storage and CPU. For a very few queries the knots are not in the right places and the model is not flexible enough to fit the data as closely as we would like (for example, "mail.yahoo.vn" in Figure 3). We are considering dynamically determining knot positions for each query string.

Similarly our model assumes a weekly pattern for all queries. While this is accurate for the vast majority of query strings, some exhibit longer patterns. Here too we may use dynamic knot positions, to provide accurate models for queries with long periodicity (like "full moon" or "santa claus") without significantly increasing CPU or storage requirements.

It may be possible to improve our current algorithm described in Section 3.3.3 by letting all terms, including the Hessian term for old data, depend on the current estimate for β. Note that the Hessian includes a term Λ, hence as predictions increase the negative Hessian also increases. In one simple case this would be easy. Suppose that only the intercept term in β changes, increasing by a. Then every value of λ increases by a factor of e^a, and Q_{C_k} would increase by the same factor. If terms other than the intercept change the situation becomes more complicated. If the Hessian for old data is updated, the derivative estimate for the old data could be based on an average of the original and updated Hessian terms for old data.

References

[1] L.A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The google cluster architecture. *Micro, IEEE*, 23(2):22–28, 2003.

[2] George Box, Gwilym M. Jenkins, and Gregory Reinsel. *Time Series Analysis, Forecasting, and Control*. Prentice-Hall, 3rd edition, 1994.

[3] J. Cao, A. Chen, T. Bu, and A. Buvaneswari. Monitoring time-varying network streams using state-space models. In *INFOCOM 2009, IEEE*, pages 2721–2725. IEEE, 2008.

[4] Carl De Boor. *A Practical Guide to Splines*. Applied mathematical sciences. Springer, 2001.

[5] A.J. Dobson. *An Introduction to Generalized Linear Models*. Texts in statistical science. Chapman & Hall/CRC, 2002.

[6] J. Durbin and S. J. Koopman. *Time Series Analysis by State Space Methods*. Oxford University Press, 2001.

[7] L. Fahrmeir and S. Wagenpfeil. Penalized likelihood estimation and iterative kalman smoothing for non-gaussian dynamic regression models. *Computational Statistics & Data Analysis*, 24(3):295–320, 1997.

[8] M.A.R. Ferreira and D. Gamerman. Dynamic generalized linear models. In *Generalized Linear Models: a Bayesian Perspective*, pages 57–72. Marcel Dekker, New York, 2000.

[9] S. Jackman. *Bayesian Analysis for the Social Sciences*. Wiley Series in Probability and Statistics. Wiley, 2009.

[10] M.E. Kuhl and J.R. Wilson. Least squares estimation of nonhomogeneous poisson processes. *Journal of Statistical Computation and Simulation*, 67(1):699–712, 2000.

[11] J. Misra and D. Gries. Finding repeated elements. *Science of Computer Programming*, 2(2):143–152, 1982.

[12] A. Veevers and MCK Tweedie. Variance-stabilizing transformation of a poisson variate by a beta function. *Applied Statistics*, pages 304–308, 1971.

[13] L. Wasserman. *All of statistics*. Springer New York, 2004.