Frank Rosenthal, Wolfgang Lehner

Efficient In-Database Maintenance of ARIMA Models

Erstveröffentlichung in / First published in:

Scientific and Statistical Database Management 23rd International Conference. Portland, 20.-22.07.2011. Springer, S. 537–545. ISBN 978-3-642-22351-8.

DOI: http://dx.doi.org/10.1007/978-3-642-22351-8_36

Final edited form was published in "Scientific and Statistical Database Management: 23rd International Conference. Portland 2011", S. 537–545, ISBN 978-3-642-22351-8
http://dx.doi.org/10.1007/978-3-642-22351-8_36

Efficient In-Database Maintenance of ARIMA Models

Frank Rosenthal and Wolfgang Lehner

Dresden University of Technology,
Database Technology Group
{frank.rosenthal,wolfgang.lehner}@tu-dresden.de

Abstract. Forecasting is an important analysis task and there is a need of integrating time series models and estimation methods in database systems. The main issue is the computationally expensive maintenance of model parameters when new data is inserted. In this paper, we examine how an important class of time series models, the *AutoRegressive Integrated Moving Average* (ARIMA) models, can be maintained with respect to inserts. Therefore, we propose a novel approach, *on-demand* estimation, for the efficient maintenance of maximum likelihood estimates from numerically implemented estimators. We present an extensive experimental evaluation on both real and synthetic data, which shows that our approach yields a substantial speedup while sacrificing only a limited amount of predictive accuracy.

Keywords: AutoRegressive Integrated Moving Average Models, Parameter Estimation, Integrated Forecasting.

1 Introduction

Time series can be encountered in many domains like business (e.g. sales or inventory), industry (e.g. yield rates or power consumption) and science (e.g. sunspot activity or ambient temperature). One important *time series analysis* task is *forecasting*, which can be achieved by creating a *model* of the time series. A suitable model represents the process that generated the time series in question and reproduces the dynamics of the time series, e.g. deterministic or stochastic trend or seasonal patterns. Forecasts of the time series can then be derived from the model.

A widely used class of time series models are the *AutoRegressive Integrated Moving Average* (ARIMA) models [2] that are able to model a wide range of real-world time series from various domains [6,8]. A key task in the creation of time series models is *parameter estimation*, which is the determination of values of the model parameters that provide an optimal fit to a time series. Optimality can be determined through different optimization criterion, e.g. a least squares approach. However, the best parameter values can usually be obtained through the *maximum likelihood* approach that uses the model specific *likelihood function* as optimization criterion [2]. The maximization of this function is often implemented through computationally expensive numerical procedures, e.g.

Final edited form was published in "Scientific and Statistical Database Management: 23rd International Conference. Portland 2011", S. 537–545, ISBN 978-3-642-22351-8
http://dx.doi.org/10.1007/978-3-642-22351-8_36

Quasi-Newton algorithms, where the parameter values are iteratively adjusted until the likelihood becomes maximal.

There has been a rising research interest in integrating forecasting functionality in database management systems [3,5,1] and there are also first practical implementations, e.g. in the analysis services in SQL Server 2008 [9]. Integrated forecasting offers the key advantage of reusing models to answer repeated or similar queries [4]. The expensive parameter estimation is performed only once and its costs amortize through using the model for answering many queries.

What has not been considered so far is the issue of *maintenance*, i.e. how to update parameter estimates when real time passes and new data becomes available. Simply ignoring the new time series elements will cause the model to become outdated and not reflect the true process any more. *Reestimation* of the parameters after every update guarantees that model parameters are always based on all available information, but it is also the most expensive possible maintenance procedure. A simple approach to maintenance is *periodic reestimation*, where parameters are reestimated after a certain number of new tuples have been inserted. This is less expensive, but omitting estimation without any consideration of the impact of the new tuples on the estimated parameters can still increase the forecast error arbitrarily [10].

In this paper, we propose *on-demand* estimation, a novel approach for efficiently maintaining maximum likelihood parameter estimates acquired through numerical optimization. We focus our presentation on `ARIMA` models, but the approach is in principal applicable to any estimation scheme using numerical optimization.

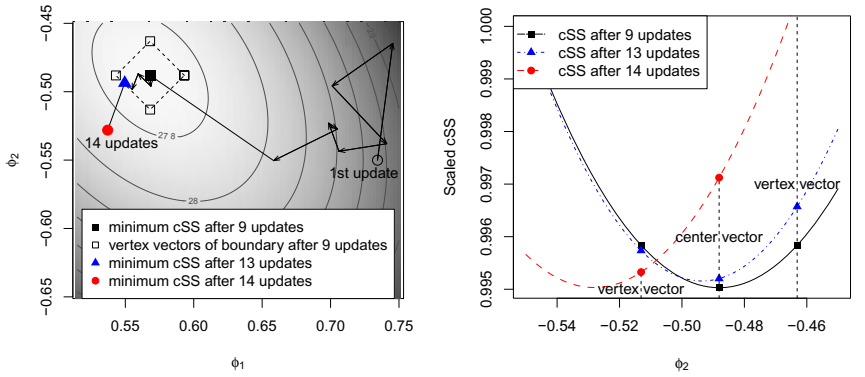
To make maintenance more efficient, on-demand estimation involves constructing a *boundary synopsis* around a found optimum on the likelihood function. The boundary helps to decide whether updates will change the optimum and thereby the estimated parameters significantly. Note that we can decide on the impact of the update without actually estimating the parameters anew. If the update will not change the optimum we can skip the expensive reestimation and we perform it otherwise.

2 On-Demand Estimation

Parameter estimation is the task to find parameter values ζ that provide the optimal fit to a given time series $x_{1:t}$. Updates to a time series expand it, e.g. an update x_t to a time series $x_{1:t}$ yields $x_{1:t+1}$. Parameter estimation using numerical procedure involves the evaluation of a function on each element of the time series at least once, but normally several times. Hence, estimating parameters is computationally expensive and becomes more expensive with each update.

We focus our discussion on parameter estimation for $\texttt{ARIMA}(p, d, q)$ models. The structural parameters p, d and q are set a priori and can not be estimated. Only p and q have an influence on the likelihood function, while d controls the preprocessing step *differencing* (not relevant for this paper). We employ one

Final edited form was published in "Scientific and Statistical Database Management: 23rd International Conference. Portland 2011", S. 537–545, ISBN 978-3-642-22351-8
http://dx.doi.org/10.1007/978-3-642-22351-8_36



(a) Estimates and cSS distribution. (b) Projection of cSS for ϕ_2.

Fig. 1. Example evolution of $\texttt{ARIMA}(2,0,0)$ estimates

variant of the likelihood function of `ARIMA` models, the *conditional likelihood* [2]. Maximization of it can be realized by minimizing the *conditional sum of squares* $cSS(\phi,\theta)$, where ϕ and θ form the parameter vector ζ to be estimated:

$$cSS(\phi,\theta) = \sum_{i=p}^{t}(a_i(\phi,\theta,x_{1:t}))^2 = \sum_{i=p}^{t}\left(x_i - \sum_{j=1}^{p}\phi_j x_{i-j} - \sum_{k=1}^{q}\theta_k a_{i-k}(\phi,\theta,x_{1:t})\right)^2$$

An $a_i(\phi,\theta,x_{1:t})$ term is the difference of x_i, the prediction from the autoregressive part of the model that uses a linear combination of the past $x_{i-p:i-1}$ and the prediction from the moving average part that uses a linear combination of the past $a_{i-q:i-1}$. Hence, the cSS can be interpreted as the sum of the *one-step-ahead prediction errors*. An important property of the cSS is that we can incrementally maintain it for fixed parameter values (ϕ,θ): $cSS_{t+1}(\phi,\theta) = cSS_t(\phi,\theta) + (a_{t+1}(\phi,\theta,x_{t+1}))^2$. Note that incremental maintenance of the objective function for fixed parameter values can be derived for many other parameter estimation tasks, e.g. for any numerical least squares estimator. Therefore, our approach could be applied there as well.

Overview. The basic idea of on demand estimation is to decide whether an update x_t, to a time series $x_{1:t-1}$ is influential enough to change the location of the optimum on the objective function significantly and hence justifies the use of estimation on the expanded time series $x_{1:t}$ to get the current parameter estimates. Hence, for `ARIMA` models we need to assess how heavily an update changes the distribution of the cSS.

We introduce our approach using an example. Figure 1(a) shows how the estimates of the parameters ϕ_1 and ϕ_2 of an $\texttt{AR}(2)$ process change after successive updates. The contour lines represent the cSS distribution after nine updates were applied and the filled square marks the minimum cSS after these nine updates. We see how nearly all earlier estimates varied heavily after updates, which

Final edited form was published in "Scientific and Statistical Database Management: 23rd International Conference. Portland 2011", S. 537–545, ISBN 978-3-642-22351-8
http://dx.doi.org/10.1007/978-3-642-22351-8_36

Algorithm 1. On-Demand Estimation

cv: center vector; vv_i: vertex vector; *tol*: tolerance

```
 1: for each update tuple x_t do
 2:   if exists(syn) then
 3:     cSS_{cv,t} ← cSS_{cv,t−1} + calculate_cSS(cv, x_t)
 4:     for each vv in syn do
 5:       cSS_{vv,t} ← cSS_{vv,t−1} + calculate_cSS(vv, x_t)
 6:       if cSS_{vv,t} < cSS_{cv,t} then
 7:         syn ← build_synopsis(estimate_parameters(x_{1:t}), tol)
 8:       end if
 9:     end for
10:   else
11:     syn ← build_synopsis(estimate_parameters(x_{1:t}), tol)
12:   end if
13: end for
```

requires reestimation. However, after the ninth update, the next four updates result only in small changes.

To decide on the impact of the next update we construct a boundary synopsis that consists of the four *vertex vectors vv* (empty squares in Figure 1(a)) around the *center vector cv* (filled square). The center vector is the minimum of the cSS found by the last estimation, while the vertex vectors represent acceptable tolerance in the parameter estimates. Figure 1(b) shows a projection of the cSS depending only on ϕ_2 with $\phi_1 \approx 0.57$, i.e. a vertical cut through the cSS from top to bottom. The continuous line with squares is a scaled version of the cSS after nine updates and with the center vector at the minimum. After each of the next four updates, estimates change only slightly as can be seen in Figure 1(b) where the cSS after the four updates (dash-dotted line with triangles) still has its minimum inside of the boundaries, although the true minimum is not at the *cv* any more. A significant change happens after the next update. We can see in Figure 1(b) that the minimum of the dashed line with circles is now outside the boundary. When we look at the left vertex vector, we notice that cSS is now smaller than at the center vector. This is our indication that the minimum has shifted and that we must apply estimation to find the new minimum. This approach is efficient since we can maintain the cSS incrementally at all parameter vectors, requiring to evaluate the objective function only on the update and adding it to the stored cSS.

Formal Definition. Algorithm 1 formalizes these considerations. On the first run of on-demand estimation we have no synopsis yet. It stores center and vertex vectors as well as the cSS at these vectors. The synopsis is constructed from a parameter estimate yielded by $estimate\_parameters(x_{1:t})$ and a user defined tolerance *tol* (line 11).

For any successive update, we update $cSS_{cv,t}$ for the center vector (line 3). Since cSS can be updated incrementally, we need to evaluate the objective function $calculate\_cSS$ for a parameter vector (*cv* and the *vv*) only on the update

Final edited form was published in "Scientific and Statistical Database Management: 23rd International Conference. Portland 2011", S. 537–545, ISBN 978-3-642-22351-8
http://dx.doi.org/10.1007/978-3-642-22351-8_36

x_t. We also update $cSS_{vv,t}$ for each vertex vector (line 5) and stop if we find a vertex vector where $cSS_{vv,t} < cSS_{cv,t}$, i.e. when we have an indication of a significant change. We then proceed as in the initial construction of the synopsis. If we update all vertex vectors without detecting a significant change, we can stop processing the update, since the old estimate is still good with respect to the tolerance.

Form of the Boundary. We propose two boundary structures: *Hypercube* and *Simplex*. While the hypercube offers absolute uniform coverage of all dimensions of the parameter space, the simplex is the structure with the smallest possible number of vertices to bound the center vector in all dimensions [11]. The hypercube can simply be constructed from a new parameter estimate, i.e. the d-dimensional center vector $cv = (cv_1, \ldots, cv_d)$ and tolerance tol by creating a pair of new vectors $vp_i = \{(cv_1, \ldots, cv_i + tol, \ldots, cv_d), (cv_1, \ldots, cv_i - tol, \ldots, cv_d)\}$ for each dimension $i = \{1, \ldots, d\}$. The boundary is the union $\bigcup_{i=1}^{d} \{vp_i\}$ and consists of $2d$ vectors.

A simplex in d dimensions is composed of $d + 1$ vectors. For example, in two dimensions the simplex is a triangle. The boundary is looser than with the hypercube, since we have fewer vertices at which to check for a shift. A simplex can be constructed from a center vector $cv = (cv_1, \ldots, cv_d)$ by creating a new vector v_i for each dimension $i = \{1, \ldots, d\}$: $v_i = \{(cv_1, \ldots, cv_i + tol * o, \ldots, cv_d)\}$, where $o = 1$ if $cv_i > 0$ and $o = -1$ if $cv_i < 0$. This yields d vectors with an orientation o away from the coordinate origin. The last vector is acquired by scaling cv in the direction of the origin in a way that the new vector v_{d+1} has a distance of tol to cv. The simplex is made up of $\bigcup_{i=1}^{d+1} \{v_i\}$.

Adaptation. We now introduce an approach for the hypercube that adapts the boundary to the variance of the parameter estimates. We achieve this by increasing the tolerance for parameters that vary heavier than others, i.e. by constructing vertex vectors that are further away from the center vector, and vice versa. An adapted hypercube can be constructed from a center vector $cv = (cv_1, \ldots, cv_d)$, tolerance tol and a vector of scale factors f_i with $i = 1, \ldots, d$ by creating a pair of new vectors $vp_i = \{(cv_1, \ldots, cv_i + f_i * tol, \ldots, cv_d), (cv_1, \ldots, cv_i - f_i * tol, \ldots, cv_d)\}$ for each dimension. To determine the f_i, we calculate the empirical variance Var_i per dimension i from the elements of the parameter estimates $cv_{1,i}, \ldots, cv_{m,i}$ in that dimension i. The scale factor f_i is defined as:

$$f_i = d * \frac{Var_i(cv_{1,i}, \ldots, cv_{m,i})}{\sum_{j=1}^{d} Var_j(cv_{1,j}, \ldots, cv_{m,j})}$$

Hence, the scale factor f_i is a measure of the variance in dimension i relative to the overall variance of the estimates. We normalize the fraction by multiplying with d and gain a factor f_i that is larger than one if the variance is above average and vice versa. If all dimensions vary equal, all scale factors are $f_i = 1$.

Internal Estimation. Internal estimation is a heuristic that seeks to improve on the base strategy of using the center point as a fixed parameter estimate. We

Final edited form was published in "Scientific and Statistical Database Management: 23rd International Conference. Portland 2011", S. 537–545, ISBN 978-3-642-22351-8
http://dx.doi.org/10.1007/978-3-642-22351-8_36

introduce it for the hypercube. The basic idea is to shift our parameter estimate from the center vector cv in the likely direction of the true cSS minimum. Our shifted estimate ζ is determined as:

$$\zeta = cv + \sum_{i=1}^{2d} \frac{d_i}{\sum_{i=1}^{2d} d_i} (vv_i - cv)$$

where $d_i = (cSS_{vv_i} - cSS_{cv})^{-1}$, i.e. the reciprocal of the cSS difference between vertex and center. This gives the most weight to vertices that have a relative low cSS which is an indication that the true estimate lies in the direction of that vertex. We normalize the d_i to sum to one and multiply the resulting weights with the directional vectors $(vv_i - cv)$. The weighted sum of all $2d$ directional vectors gives the final estimate.

3 Evaluation

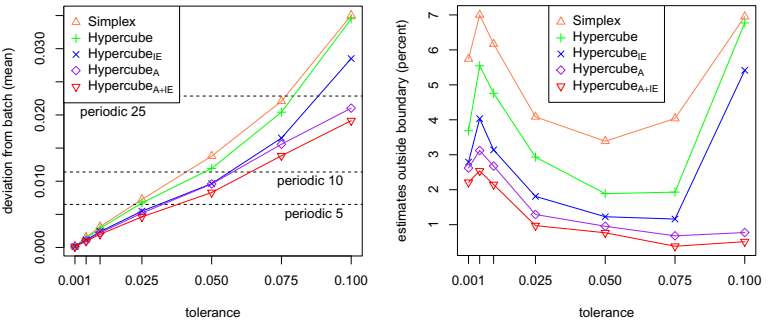
We implemented all presented maintenance approaches in the `R` programming language. Our evaluation consists of a series of simulations to evaluate run time behavior and accuracy. A simulation run is composed of an initial batch estimation on a starting window $x_{1:S}$ of the time series at hand. S is set large enough to give a first parameter estimate (e.g. $S = p + 1$ for `ARIMA`$(p, 0, 0)$). We then expand the time series to $x_{1:S+1}$, $x_{1:S+2}$ and so on until the end of the series and we call the maintenance procedure after each expansion.

The *baseline* approach consists in using estimation for each update. We also report results for *periodic estimation* for specific rates of estimation. In the figures, `periodic x` means that after each `x` tuples parameters were estimated. We measured accuracy and runtime behavior and we present the latter as *speedup* in terms of the average estimation time in comparison to the baseline.

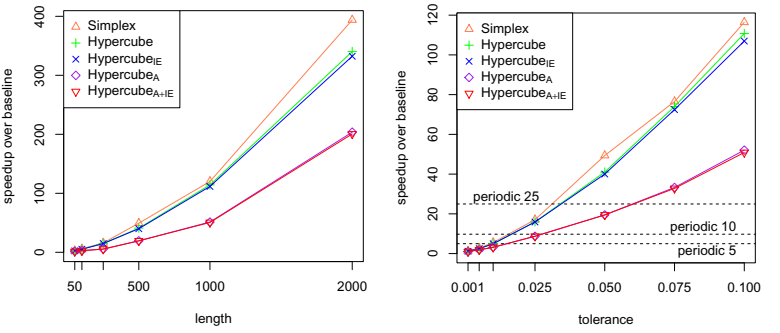
We used both real and synthetic data for our evaluation. The synthetic set is composed of several simulated time series of different length that behave according to given `ARIMA` models of different order and parameterization. The first real dataset are the 645 yearly time series from the M3 competition [8], an extensive empirical evaluation of the predictive performance of a large number of forecasting methods. The second real data set is derived from the half hourly measured electricity demand in Great Britain, which is published by *National Grid* [7]. We used the *Initial Demand Outturn* of 2008 and aggregated it to the sum per day yielding a 347 element time series with seasonal patterns per week as well as over the year.

Synthetic Data. To evaluate the loss in accuracy caused by skipping estimation we present the deviation of estimates for periodic and on-demand estimation with respect to the estimates yielded by the baseline. As can be seen from Figure 2(a), the mean deviation, as root mean squared error, lies well inside the bounds of the used tolerance. We can see that the Hypercube estimators have tighter bounds. Note that the internal estimation `(IE)`, the adaptation `(A)` and the

Final edited form was published in "Scientific and Statistical Database Management: 23rd International Conference. Portland 2011", S. 537–545, ISBN 978-3-642-22351-8
http://dx.doi.org/10.1007/978-3-642-22351-8_36



(a) Mean deviation from baseline. (b) Percentage outside of boundary.



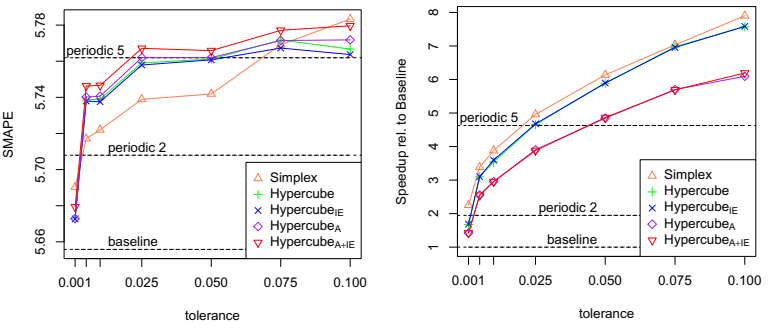
(c) Speedup vs. length of time series. (d) Speedup vs. tolerance.

Fig. 2. Experimental Results on Synthetic Data

combination `(A+IE)` of these two modifications each reduce the mean deviation. Because we can only check cSS values at the vertex vectors, the boundary is not strict. Figure 2(b) shows the percentage of estimates that varied larger than the set tolerance. The ratio is relatively large for very small tolerances because the covered parameter space is equally small. The percentage of boundary violations sinks with increasing tol, but rises again for those approaches that do not adapt the boundary.

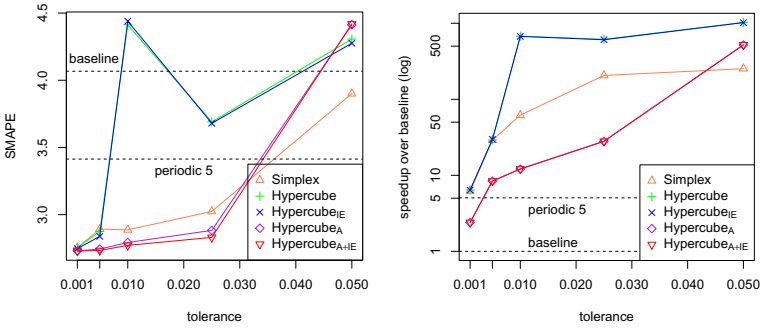
We see from Figure 2(c) where $tol = 0.05$ that the speedup increases as expected with the length of the overall simulated time series. The reason is that the benefit for each skipped update is larger for larger time series. Speedup also increases with increasing tolerance (Figure 2(d), $length = 500$), whereby we see an speedup of about 50 for the Simplex and about 40 for the static Hypercube estimators at $tol = 0.05$. Looking at Figures 2(a) and 2(d) conjointly, we see that for a given tolerance, e.g. $tol = 0.05$, the variants of on-demand estimation in comparison to periodic estimation can offer a smaller error, e.g. adapted Hypercube vs. `periodic 25`, a greater speedup, e.g. Simplex vs. `periodic 10`, or both e.g. adapted Hypercube with internal estimation vs. `periodic 10`.

Final edited form was published in "Scientific and Statistical Database Management: 23rd International Conference. Portland 2011", S. 537–545, ISBN 978-3-642-22351-8
http://dx.doi.org/10.1007/978-3-642-22351-8_36



(a) SMAPE vs. tolerance. (b) Speedup vs. tolerance.

Fig. 3. Experimental Results on M3 Competition Data



(a) SMAPE vs. tolerance. (b) Speedup vs. tolerance.

Fig. 4. Experimental Results on Energy Demand Data

Real Data. We used the time series from the M3 competition to test the robustness of our approach under adverse conditions. These series are very short with a minimum length of only 28 elements. Hence, there is little that can be gained from omitting estimation and the estimates on such short time series may vary strongly with each update. However, there are still inserts that do not change the parameters significantly and on-demand estimation identifies these updates. As can be seen form Figure 3(b), even on this adverse data set we are never slower than the baseline and can even achieve speedups, e.g. a speedup of about 6 for the Hypercube approach with $tol = 0.05$.

We evaluated accuracy using a series of one-step-ahead forecasts $\hat{x}_{t+1}$ from the models and measuring the error to the real value x_{t+1} using the *symmetric mean absolute percentage error* $\text{SMAPE} = 100 * t^{-1} \sum_{i=1}^{t} (|x_t - \hat{x}_t|)(|x_t| + |\hat{x}_t|)^{-1}$ (range: 0% to 100%). We can see from Figure 3(a) that on-demand and periodic estimation yield a slightly worse but acceptable SMAPE than the baseline. The variation between approaches is small, but for larger tolerances the speedup

Final edited form was published in "Scientific and Statistical Database Management: 23rd International Conference. Portland 2011", S. 537–545, ISBN 978-3-642-22351-8
http://dx.doi.org/10.1007/978-3-642-22351-8_36

of on-demand estimation is higher than that of periodic estimation with only slightly greater error.

Figure 4(a) shows the SMAPE of the energy dataset. Note that the SMAPE for nearly all of our approaches is lower than the SMAPE of the baseline. This can occur in real world data, where skipping an update might decrease the forecast error. We can also see an outlier for the two static Hypercube approaches at tolerance $tol = 0.01$ which does not show up in the adaptive approaches that perform better. The speedup on this data set is huge for all our approaches e.g. about 250 for the Simplex at $tol = 0.025$ (Figure 4(b), logarithmic scale). Periodic estimation has a smaller speedup and greater error.

4 Conclusion

We proposed on-demand estimation, a maintenance strategy that uses expensive estimation only when necessary and we presented several on-demand estimators. With these estimators we achieved a considerable speedup over the baseline approach on synthetic and real data with only little impact on predictive accuracy. Our approaches also outperform periodic reestimation by offering either a smaller error or a greater speedup.

References

1. Agarwal, D., Chen, D., ji Lin, L., Shanmugasundaram, J., Vee, E.: Forecasting high-dimensional data. In: SIGMOD, pp. 1003–1012 (2010)
2. Box, G., Jenkins, G., Reinsel, G.: Time Series Analysis: Forecasting and Control, 4th edn. Wiley, Chichester (2008)
3. Duan, S., Babu, S.: Processing Forecasting Queries. In: VLDB, pp. 711–722 (2007)
4. Fischer, U., Rosenthal, F., Boehm, M., Lehner, W.: Indexing forecast models for matching and maintenance. In: IDEAS, pp. 26–31 (2010)
5. Ge, T., Zdonik, S.B.: A skip-list approach for efficiently processing forecasting queries. PVLDB 1(1), 984–995 (2008)
6. Gooijera, J.G.D., Hyndman, R.J.: 25 years of time series forecasting. International Journal of Forecasting 22(3), 443–473 (2006)
7. Grid, N.: Demand data, `http://www.nationalgrid.com/uk/Electricity/Data/Demand+Data`
8. Makridakis, S., Hibon, M.: The M3-Competition: results, conclusions and implications. International Journal of Forecasting 16(4), 451–476 (2000)
9. Microsoft: PredictTimeSeries - Microsoft SQL Server Books (2008), `http://msdn.microsoft.com/en-us/library/ms132167.aspx`
10. Tashman, L.J.: Out-of-sample tests of forecasting accuracy: an analysis and review. International Journal of Forecasting 16(4), 437–450 (2000)
11. Weisstein, E.W.: Simplex, `http://mathworld.wolfram.com/Simplex.html`