Journal of Machine Learning Research 15 (2014) 2607-2627 Submitted 1/13; Revised 3/14; Published 7/14

Efficient and Accurate Methods for Updating Generalized Linear Models with Multiple Feature Additions

Amit Dhurandhar ADHURAN@US.IBM.COM
Marek Petrik MPETRIK@US.IBM.COM
IBM TJ Watson,
1101 Kitchawan Road, Yorktown Heights, NY 10598 USA

Editor: Charles Elkan

Abstract

In this paper, we propose an approach for learning regression models efficiently in an environment where multiple features and data-points are added incrementally in a multi-step process. At each step, any finite number of features maybe added and hence, the setting is not amenable to low rank updates. We show that our approach is not only efficient and optimal for ordinary least squares, weighted least squares, generalized least squares and ridge regression, but also more generally for generalized linear models and lasso regression that use iterated re-weighted least squares for maximum likelihood estimation. Our approach instantiated to linear settings has close relations to the partitioned matrix inversion mechanism based on Schur's complement. For arbitrary regression methods, even a relaxation of the approach is no worse than using the model from the previous step or using a model that learns on the additional features and optimizes the residual of the model at the previous step. Such problems are commonplace in complex manufacturing operations consisting of hundreds of steps, where multiple measurements are taken at each step to monitor the quality of the final product. Accurately predicting if the finished product will meet specifications at each or, at least, important intermediate steps can be extremely useful in enhancing productivity. We further validate our claims through experiments on synthetic and real industrial data sets.

Keywords: linear regression, logistic regressions, lasso, group lasso, feature selection, manufacturing

1. Introduction

Complex manufacturing is a multi-billion dollar industry, which encompasses diverse domains ranging from semiconductor to pharmaceutical to consumer and snack products. Each of these industries undertake complex operations that consist of hundreds of steps from the beginning to the very end when the final product is produced. In each of these steps, multiple measurements are made to monitor the process. Thus, every product has an incremental history of measurements accumulated over time. The hope is that the final finished product will meet the necessary specifications. However, in each of these industries, there are millions (to even billions) of dollars of losses every year because of out-of-spec products. It would thus be extremely useful if one could efficiently utilize the accrued intermediate measurements or features to accurately predict the quality of the final product. For example, in the semiconductor industry, wafers—which are a collection of chips—travel together in groups called lots through hundreds of processing steps with thousands of measurements being accrued as the wafers reach the end. At this point, the quality of the wafers is determined by either checking their speed or power. It can be extremely useful if one could, at least at critical intermediate steps, provide an accurate estimate of the final speed so as to possibly take corrective actions or avoid further processing.

In particular, the data flows through K steps as groups of data-points or a batch with the target becoming available almost simultaneously for each member of the group at the very end. We want to efficiently build a model at each step based on this data so as to estimate the performance of forthcoming batches at these steps. There are multiple such batches that flow through the processing steps and hence, we also want to update our model at each step based on recently acquired data. Therefore, in our problem, both features and data-points are added as batches move through the process and as new batches are created. The notation used throughout the paper is given in Table 1 and an illustration of our problem setting is shown in Figure 1.

The methods we propose for incremental updates with new feature additions are by no means constrained only to manufacturing. They also are applicable to feature selection algorithms such as least angle regression (Efron et al., 2004), the homotopy method for lasso (Tibshirani and Taylor, 2011), or orthogonal matching pursuit (Hastie et al., 2009). These feature selection methods must efficiently update the model whenever a new feature is added. The most common approach is to keep a QR factorization of the regression matrix. This factorization can be updated with every new feature added using either Givens rotations or Householder reflections, which are both computationally efficient and numerically stable (Golub and Loan, 1996).

The problem we study differs from the typical feature selection setting in two important aspects. First, while most feature selection methods add one feature at a time, our method is more suitable when many features are to be added at a time. Second, the QR factorization can only be used with linear regressors, while our method also works with generalized linear models. As a result, our methods can also be applied in more sophisticated settings, such as to efficiently update regression models in non-linear group variable selection techniques (Lozano et al., 2009, 2011).

Our method when instantiated to the linear regression case, where we minimize a least squares objective, is closely related to the partitioned matrix inverse mechanism based on Schur's complement (Boyd and Vandenberghe, 2004). Although both absolve the need for computing matrix inverses over the entire currently available feature set, our procedure has an intuitive explanation. Moreover, our method is a meta-technique applicable to any base regression method, and as you will see later, is an essential component in optimally updating generalized linear models such as logistic regression, which as we know are non-linear.

In the rest of the paper, we first formally define our problem. We then propose a novel algorithm, which we show to be optimal for generalized least squares and ridge regression. We discuss its relation to the partitioned matrix inversion mechanism. We provide an extension of our algorithm, which is efficient and optimal for generalized linear models. We provide a relaxation of our algorithm, which can be used to efficiently update other regression techniques and whose performance is no worse than using the model from the previous step or using a model that learns on the additional features and optimizes the residual of the model at the previous step.

Term/Symbol	Description
Step/Epoch	The instant when new features become available.
Batch	A set of data points that move through all the steps.
d_i	Number of features accumulated till (and including) step i.
X_i^b,x_i^b	Input data from batch b with d_i and $(d_i - d_{i-1})$ features respectively.
M_i^b,m_i^b	Model built on the first d_i features with all the data till batch b and only data in batch b respectively.
Y^b	Outputs corresponding to batch b.
$\hat{Y}_q^p$,R_q^p	Matrix of predictions and the matrix of residuals respectively, obtained by regressing inputs q on columns of p.

Table 1: The notation used in the paper. In much of the paper we drop the batch b from the superscript since, we propose methods for efficient updates to models with feature additions that are applicable to any batch.

$$\begin{array}{c|ccccc|c}
 & 1 & \xrightarrow{\text{Step}} & i & \longrightarrow & K & \\
\hline
1 & M_1^1 & \cdots & M_i^1 & \cdots & M_K^1 & N_1 \\
\text{Batch} \downarrow & \vdots & & \vdots & & \vdots & \\
\hline
b & M_1^b & \cdots & M_i^b & \cdots & M_K^b & N_b \\
 & m_1^b & \cdots & m_i^b & \cdots & m_K^b &
\end{array}$$

Figure 1: Process where features and data-points are added, with new models being learned at each step.

2. Problem Statement

Before describing the problem statement, we will introduce some notation. Let K denote the number of steps in the process. Let d_i denote the number of features $F_i = \{f_1, ..., f_{d_i}\}$ present at step i. It is important to note that $F_i \supset F_{i-1}, \forall i \in \{2, ..., K\}$ which implies $d_i > d_{i-1}, \forall i \in \{2, ..., K\}$. There are multiple batches that flow through the K steps and we denote the size of one such batch b, by N_b. Based on a particular learning algorithm, we denote the model learned at step i trained on data from batch b, by m_i^b (local model). Let M_i^b (global model) denote the model based on all the data accumulated till batch b in step i. For efficiency reasons, though we may not learn from scratch in step i using all the available data, M_i^b is obtained by potentially learning over a sample of size $\sum_{j=1}^{b} N_j$. Thus,

m_i^b is a local model learned over just recent data, while M_i^b is the model we will use to predict the outcomes of the data points in the $(b+1)^{th}$ batch when they reach step i. Note that for batch 1, the local and global models are the same, i.e., $m_i^1 = M_i^1, \forall i \in \{1, ..., K\}$. A pictorial representation of the process with notation is shown in Figure 1.

Let X_i^b ($N_b \times d_i$ matrix) and x_i^b ($N_b \times (d_i - d_{i-1})$ matrix) denote all the input data available[1] in step i batch b and the input data for only the additional features in step i batch b, i.e., $\{f_{d_{i-1}+1}, ..., f_{d_i}\}$ respectively. Let Y^b denote the final outcomes or target in batch b. Let $\hat{Y}_q^p$ and R_q^p denote the matrix of predictions and the matrix of residuals respectively, obtained by regressing inputs q on columns of p based on a chosen regression technique. Thus, if p is a matrix, each of its columns is considered a target and q is regressed separately on each of them. Consequently, $\hat{Y}_{X_i^b}^Y$ and $R_{X_i^b}^Y$ are the predictions and residuals of the model m_i^b respectively. Let I_j denote an identity matrix of rank j.

Given our dual goal of efficiently learning with i) new features and ii) with new data points being added, we address each of these issues in isolation. With this, we have the following two problems that we need to tackle:

I During any batch b of the K step process, given a model at step $i \in \{1, ..., K-1\}$ learned over d_i features $\{f_1, ..., f_{d_i}\}$ and a sample size N_b, how do we update this model at step $i+1$ with d_{i+1} features $\{f_1, ..., f_{d_i}, ..., f_{d_{i+1}}\}$ such that the resultant model is (a) no less accurate than the model at step i? (b) no less accurate than the composite model which consists of the model learned using only the additional features $\{f_{d_i+1}, ..., f_{d_{i+1}}\}$ on the residuals of the model at step i? and (c) more efficient to learn than learning from scratch over all the features available at step $i+1$ i.e., $\{f_1, ..., f_{d_{i+1}}\}$?

II At any step i in the K step process, given the model M_i^b and the model m_i^{b+1}, how do we efficiently obtain the model M_i^{b+1}?

Though not completely solved, there has been extensive work to handle Question II (Blum, 1996; Smale and Yao, 2005; Bottou and Cun, 2003). We thus focus our attention on question I. Given this and for simplicity of notation, from here on, we do not refer to the batch any more; that is, we drop b from the notation, since the methods we describe are applicable to any batch in the process.

3. Methodology

In this section, we first suggest a meta-algorithm to successfully tackle question I. We then show that not only can this algorithm be realized efficiently for ordinary least squares, weighted least squares, generalized least squares and ridge regression, but it is also optimal for these techniques. We also show that the algorithm can be used as a core function to efficiently and optimally solve iterated re-weighted least squares procedures, which are used to find the maximum likelihood estimates for Generalized Linear Models (McCullagh and Nelder, 1990) and sometimes even Lasso (Tibshirani, 1994). For arbitrary regression algorithms, it can be shown that even a relaxation of our technique results in a positive response to questions I(a), I(b) and I(c). We refer to the optimal model in any step that

1. This includes the constant term.

Algorithm 1 Proposed meta-algorithm—which can be embedded as a core function in other algorithms—to update model built in step i during the next step $i+1$. ζ_i are other regression algorithm specific parameters.

```
Input: $m_i$, $R^Y_{X_i}$, $X_{i+1}$, $Y$ and $\zeta_i$.
Output: $m_{i+1}$, $R^Y_{X_{i+1}}$ and $\zeta_{i+1}$
Compute $R^{x_{i+1}}_{X_i}$ {Use $X_i$ to predict columns of $x_{i+1}$ and compute the residual matrix.}
Regress $R^{x_{i+1}}_{X_i}$ on $R^Y_{X_i} \to \bar{m}_{i+1}$ {Regressing the residuals outputs the model on the right.}
Regress $X_i$ on $Y - \hat{Y}^{R^Y_{X_i}}_{R^{x_{i+1}}_{X_i}} \to \hat{m}_i$ {Regressing the input of step $i$ with the previous residual yields the model on the right.}
$m_{i+1} = (\hat{m}_i; \bar{m}_{i+1})$ {Composing these models, i.e., for example concatenating their parameter vectors, gives the final model.}
if $i + 1 == K$ then
    {If $i+1$ is the last step.}
    Return $m_{i+1}$
else
    Return $m_{i+1}$, $R^Y_{X_{i+1}}$ and $\zeta_{i+1}$
end if
```

is learned from scratch by method Standard. We refer to the model that learns on the **a**dditional **f**eatures and **o**ptimizes the **r**esidue of a model at the previous step by method AFOR.

Algorithm 1 can be described as follows: First, we find the portion of the target that was not modeled by the regression algorithm (residuals) in the previous step. We then try to find the additional information that the new features in the current step contain. Using this additional information we fit to the residuals in the previous step. The residuals from this model are subtracted from the original target and the features in the previous step are fit to this modified target. The final model is a composition of these two latter models.

Loosely speaking, the intuition behind Algorithm 1 is to find what additional benefit the new features bring us in predicting the portion of the target that was not modeled by the previously available set of features, and after the removal of their effect, we focus on modeling this new modified target using the previously available set of features. This procedure thus uses the old and new set of features to model, as much as possible, the parts of the target that are not explained by the other, resulting in significantly reduced redundancy in the final model.

We now see how effectively this method can be applied to different regression algorithms. A comparison of the different approaches using different regression methods is shown in Table 2. The computational complexity of each algorithm depends on the method used to invert the matrices. The most suitable method depends on the size and sparsity of the matrix (Golub and Loan, 1996; Saad, 1997). The fastest known algorithms have a time complexity of $O(d^{2.3})$ but are generally not useful in practice because of a large multiplicative constant (Coppersmith and Winograd, 1990).

	OLS	WLS	GLS	Ridge	IRLS (for GLM and Lasso)
	Efficiency, Optimality				
Our Method	$(d_{i+1} - d_i) \times (d_{i+1} - d_i)$ inverse, Optimal				Smaller problem size, Optimal
Standard	$d_{i+1} \times d_{i+1}$ inverse, Optimal				Optimal
AFOR	$(d_{i+1} - d_i) \times (d_{i+1} - d_i)$ inverse, Sub-optimal				Sub-optimal

Table 2: Comparison of three meta-approaches across different regression algorithms in step $i + 1$. Standard denotes learning from scratch.

3.1 Generalized Least Squares and Ridge Regression

In least squares regression (Hastie et al., 2009), we minimize a quadratic loss function wherein each datapoint may be weighted or unweighted (i.e., equi-weighted). Based on the weighting scheme, we have the following three variants with each successive variant being a generalization of the ones before. Ordinary Least Squares (OLS) is the unweighted variant. Weighted Least Squares (WLS) is the variant, where the weight matrix is diagonal. Generalized Least Squares (GLS) is the most general variant, wherein we have a full weight matrix W. Ridge regression on the other hand is regularized OLS with a quadratic penalty (Hoerl and Kennard, 1970).

We now instantiate Algorithm 1 for GLS with a quadratic penalty. We could refer to this method as Generalized Ridge (GR) regression. Eliminating the penalty term or setting W to the identity matrix in GR would give us instantiations for GLS and ridge regression respectively.

The objective function minimized in step i for GR is

$$Q_{gr} = (Y - X_i\beta_i)^T W(Y - X_i\beta_i) + \lambda {\beta_i}^T \beta_i,$$

where β_i is the parameter vector we wish to estimate[2] and $\lambda > 0$ is the regularization parameter. The optimal β_i is given by $m_i = ({X_i}^T W X_i + \lambda I_{d_i})^{-1} {X_i}^T W Y$, where I_{d_i} is a $d_i \times d_i$ identity matrix. To estimate β_{i+1} we use Algorithm 1 as follows:

- $R^{x_{i+1}}_{X_i} = (I_N - H_i)x_{i+1}$. Here $\zeta_i = H_i = X_i({X_i}^T W X_i + \lambda I_{d_i})^{-1} {X_i}^T W$.
- $\bar{m}_{i+1} = ({x_{i+1}}^T W(I_N - H_i)x_{i+1} + \lambda I_{(d_{i+1}-d_i)})^{-1} {x_{i+1}}^T (I_N - H_i)Y$. This is a consequence of the fact that H_i is idempotent.
- $\hat{m}_i = ({X_i}^T W X_i + \lambda I_{d_i})^{-1} {X_i}^T W(Y - x_{i+1}\bar{m}_{i+1}) = m_i - \alpha_i \bar{m}_{i+1}$, where $\alpha_i = ({X_i}^T W X_i + \lambda I_{d_i})^{-1} {X_i}^T W x_{i+1}$.
- $m_{i+1} = (\hat{m}_i; \bar{m}_{i+1})$. Now m_{i+1} can be immediately used to obtain predictions for new batches whose target is currently unknown and have reached step $i + 1$.
- If $i + 1 < K$, then return m_{i+1}, $R^Y_{X_{i+1}}$ and H_{i+1}. Here H_{i+1}—which also determines $({X_{i+1}}^T W X_{i+1} + \lambda I_{d_{i+1}})^{-1}$—can be computed from $({X_i}^T W X_i + \lambda I_{d_i})^{-1}$ and

2. For simplicity of notation assume the data has zero intercept or the constant term is included in X_i.

$(x_{i+1}{}^T W(I_N - H_i)x_{i+1} + \lambda I_{(d_{i+1}-d_i)})^{-1}$ using standard partitioned matrix inversion property with only matrix multiplications but no more inversions (Boyd and Vandenberghe, 2004).

Let us now see why this method is more efficient than performing regression from scratch in step $i+1$. If we performed regression from scratch we would have had to invert a $d_{i+1} \times d_{i+1}$ matrix. In our case however, we have to invert only a $(d_{i+1} - d_i) \times (d_{i+1} - d_i)$ matrix in step $i+1$. The reason for this is that $(X_i{}^T W X_i + \lambda I_{d_i})^{-1}$, which is a $d_i \times d_i$ matrix, is already available from step i. Moreover, our method is also more efficient at obtaining the current step model than applying the partitioned matrix inversion property based on Schur's complement directly, because although no more inversions are required by it there are many more matrix multiplications that have to be performed.

For instance, based on Schur's complement and the partitioned matrix inversion mechanism we would compute the optimal estimate for β_{i+1} denoted by β_{i+1}^{opt} as follows

$$\begin{aligned}\beta_{i+1}^{opt} &= \left[\begin{array}{cc} X_i{}^T W X_i + \lambda I_{d_i} & X_i{}^T W x_{i+1} \\ x_{i+1}^T W X_i & x_{i+1}^T W x_{i+1} + \lambda I_{(d_{i+1}-d_i)} \end{array}\right]^{-1} \left(\begin{array}{c} X_i{}^T \\ x_{i+1}^T \end{array}\right) Y \\ &= \left[\begin{array}{cc} A^{-1} + A^{-1} U C^{-1} V A^{-1} & -A^{-1} U C^{-1} \\ -C^{-1} V A^{-1} & C^{-1} \end{array}\right] \left(\begin{array}{c} X_i{}^T \\ x_{i+1}^T \end{array}\right) Y \end{aligned}, \qquad (1)$$

where $A = X_i{}^T W X_i + \lambda I_{d_i}$, $U = X_i{}^T W x_{i+1}$, $V = x_{i+1}^T W X_i$ and $C = x_{i+1}^T W(I_N - H_i)x_{i+1} + \lambda I_{(d_{i+1}-d_i)}$. Here C is the Schur's complement.

It turns out that our estimate $m_{i+1} = \beta_{i+1}^{opt}$ and is thus optimal for GR regression as seen below.

Theorem 1 *If we use GR regression in our process, then in any step $i+1$, where $i \in 1, ..., K-1$, m_{i+1} is the optimal GR estimator.*

Proof Equation (1) can be expanded and rewritten as follows

$$\begin{aligned}\beta_{i+1}^{opt} &= \left[\begin{array}{cc} A^{-1} + A^{-1} X_i{}^T W x_{i+1} C^{-1} x_{i+1}^T W X_i A^{-1} & -A^{-1} X_i{}^T W x_{i+1} C^{-1} \\ -C^{-1} x_{i+1}^T W X_i A^{-1} & C^{-1} \end{array}\right] \left(\begin{array}{c} X_i{}^T \\ x_{i+1}^T \end{array}\right) Y \\ &= \left(\begin{array}{c} A^{-1} X_i{}^T W(Y - A^{-1} X_i{}^T W x_{i+1}{}^T C^{-1} x_{i+1}{}^T (I_N - H_i) Y) \\ C^{-1} x_{i+1}{}^T (I_N - H_i) Y \end{array}\right) \\ &= \left(\begin{array}{c} (X_i{}^T W X_i + \lambda I_{d_i})^{-1} X_i{}^T W(Y - x_{i+1}\bar{m}_{i+1}) \\ (x_{i+1}{}^T W(I_N - H_i)x_{i+1} + \lambda I_{(d_{i+1}-d_i)})^{-1} x_{i+1}{}^T (I_N - H_i) Y \end{array}\right) \\ &= \left(\begin{array}{c} \hat{m}_i \\ \bar{m}_{i+1} \end{array}\right) \\ &= m_{i+1} \end{aligned}.$$

■

Remark 1 *The result in Theorem 1 implies that using Algorithm 1, we get efficient and optimal estimates for OLS, WLS, GLS and ridge regression in step $i+1$.*

Algorithm 2 Method to update model built using IRLS in step i during the next step $i+1$, using Algorithm 1.

```
Input: $m_i$ or $\beta_i$, $R_{X_i}^{z_i^{t_i}}$, $X_{i+1}$, $Y$, $z_i^{t_i}$, $H_i^{t_i}$, $W_i^{t_i}$, $\epsilon_{\mathcal{D}} \in (0,1]$ and $\epsilon_\beta \geq 0$.
Output: $m_{i+1}$ or $\beta_{i+1}$, $R_{X_{i+1}}^{z_{i+1}^{t_{i+1}}}$ and $H_{i+1}^{t_{i+1}}$
Run Algorithm 1 with inputs $m_i$, $R_{X_i}^{z_i^{t_i}}$, $X_{i+1}$, $z_i^{t_i}$, $H_i^{t_i}$ and $W_i^{t_i}$
if $P[\mathcal{D}_{i+1} > -2(\mathcal{L}(m_{i+1}|X_{i+1}, Y) - \mathcal{L})] \leq \epsilon_{\mathcal{D}}$ then
  {Checking deviance, where $\mathcal{L}$ is the max possible log-likelihood value.}
  $z_{i+1}^{t_{i+1}} = z_{i+1}^{t_i}$, $W_{i+1}^{t_{i+1}} = W_i^{t_i}$
else if $max_{j\in\{1,...,d_i\}}|m_{i+1}(j) - m_i(j)| \leq \epsilon_\beta$ then
  Let $\hat{m}_{i+1} = (m_{i+1}(1), ..., m_{i+1}(d_i))^T$
  Run IRLS only on $x_{i+1}$ from this point onward with $\hat{m}_{i+1}$ fixed
  Let $\bar{m}_{i+1}$ be the optimal solution of the above IRLS run
  $m_{i+1} = (\hat{m}_{i+1}; \bar{m}_{i+1})$
else
  Run IRLS using output from Algorithm 1 i.e., $m_{i+1}$, $R_{X_{i+1}}^{z_{i+1}^{t_i}}$ and $H_{i+1}^{t_i}$
end if
Return $m_{i+1}$, $R_{X_{i+1}}^{z_{i+1}^{t_{i+1}}}$ and $H_{i+1}^{t_{i+1}}$
```

$$\begin{tikzcd}
W_1^1, z_1^1, X_i \arrow[d, "\text{standard after } t_i \text{ iterations}"] & W_1^1, z_1^1, X_{i+1} \arrow[d, "\text{Taylor approx. after } t_i \text{ iterations}"] \\
W_i^{t_i}, z_i^{t_i}, X_i \arrow[r, "\text{Algorithm 1}"'] & W_i^{t_i}, z_i^{t_i}, X_{i+1}
\end{tikzcd}$$

Figure 2: The figure shows where Algorithm 1 can be used to efficiently transform the optimal solution in step i to a solution in step $i+1$.

3.2 Generalized Linear Models and Lasso using IRLS

Generalized Linear Models (GLMs) assume that the target is generated from certain specific distributions belonging to the exponential family. Normal, Poisson, binomial, gamma and exponential are some of the distributions that are considered. The target is related to a linear combination of the inputs through a linear or non-linear function called the link function. Formally, if we learned a GLM in step i, we would have the following relationship

$$E[Y] = \hat{Y}_{X_i}^{Y} = g^{-1}(X_i\beta_i),$$

where $g(.)$ is the link function. For a normal distribution $g(.)$ is identity and the resultant regression method is just OLS. For a binomial $g(.)$ is the logit function and if values of the target lie in $[0,1]$, then the resultant regression method is logistic regression.

Lasso (Tibshirani, 1994) is L1 regularized OLS. It is usually used to solve under-determined systems of equations, where we have more features than data points. The idea here is to avoid over-fitting and choose features that are truly predictive. Formally, a lasso in step i would minimize the following objective

$$Q_L = (Y - X_i\beta_i)^T(Y - X_i\beta_i) + \lambda|\beta_i|^1.$$

3.2.1 Iteratively Re-weighted Least Squares

Both of the above classes of regression methods can be solved using iterative procedures. Iteratively Re-weighted Least Squares (IRLS) is a popular technique used to find the maximum likelihood estimates (MLE) for GLMs. Although there are other preferred methods to solve lasso, IRLS is still an effective method in this context. IRLS usually uses Newton Raphson updates, where the updated predictions at each iteration determine the weight matrix and the target in the next iteration. In particular, if $W_i^{t_j}$ and $z_i^{t_j}$ are the weight matrix and target in step i, and in the t_j^{th} iteration respectively, then the weight matrix is a $N \times N$ diagonal matrix whose diagonal elements correspond to the reciprocal of the variances computed from predictions in the previous iteration $(t_j - 1)$. $z_i^{t_j}$ on the other hand is given by $X_i\beta_i^{t_j-1} + W_i^{t_j}(Y - \hat{Y}_{X_i}^{z_i^{t_j-1}})$.

On the left hand portion of Figure 2, we see that in step i if we run IRLS, we get the optimal solution after t_i iterations. This optimal solution corresponds to an optimal weight matrix $W_i^{t_i}$ and an optimal target $z_i^{t_j}$. Using this weight matrix and target we can efficiently obtain the corresponding WLS solution in step $i+1$ using Algorithm 1. This solution relates to performing IRLS in step $i+1$, where at each iteration we approximate (zeroth order) the predictions around $R_{X_i}^{x_{i+1}}x_{i+1}$ using Taylor expansion

$$\hat{Y}_{X_{i+1}}^{z_{i+1}^{t_j}} = \hat{Y}_{X_i}^{z_i^{t_j}} + R_{X_i}^{x_{i+1}}x_{i+1}s(X_{i+1}, z_{i+1}^{t_j}),$$

where $s(.)$ is a function denoting higher order terms. Hence, at each iteration if we take the zeroth order approximation, then the WLS problem solved has the same weight matrix and target as that solved in step i at the same iteration. This is depicted on the right hand side of Figure 2.

3.2.2 Understanding Algorithm 2

In Algorithm 2, we first use Algorithm 1 to find the optimal solution in the current step $i+1$ of the WLS problem at iteration t_i in step i. The weight matrix and target in this WLS problem correspond to the optimal IRLS solution in step i. We then check to see if the probability of deviance $\mathcal{D}_{i+1}$ (McCullagh and Nelder, 1990) being greater than twice the difference between the max possible log-likelihood value (i.e., if $\hat{Y}_{X_{i+1}}^{Y} = Y$) and the log-likelihood of the current solution is less than a small constant $\epsilon_{\mathcal{D}}$. Deviance is a statistic that measures goodness of fit. For any step i, $\mathcal{D}_i = -2(\mathcal{L}(m_i|X_i, Y) - \mathcal{L})$, where m_i denotes a corresponding optimal model. Asymptotically, $\mathcal{D}_i$ has a χ^2 distribution with $N - d_i$ degrees of freedom. Using this fact we can compute the required probability in Algorithm 2 and check for the specified condition. A small value of the probability indicates that we are already near an optimal solution and hence, we have a satisfactory solution.

If this condition is not satisfied, we then check to see what is the maximum change in the d_i parameter values going from the optimal IRLS solution in step i to our current solution in step $i+1$. If the maximum change is small, i.e., $\leq \epsilon_\beta$, we fix these parameters and only run IRLS on the remaining set. A small change in the previous step parameters indicates that the new set of features in step $i+1$ are practically orthogonal to step i features and hence, the previous step parameters will change little as we approach the optimal solution in the current step.

If neither of the above conditions are satisfied we simply run IRLS, starting at the current solution.

3.2.3 Analysis

It is easy to see that if the first condition in Algorithm 2 is satisfied, then we have reached our desired solution and hence, this is definitely more efficient than starting from the beginning. The question is, are we doing better in the other two cases, namely, when only the second condition is satisfied or when neither condition is satisfied? The latter scenario is probably worse since, we are solving at each iteration a WLS problem with d_{i+1} features rather than $d_{i+1} - d_i$ features. We cannot claim for certain that these two scenarios are more efficient than learning afresh in step $i+1$, but the following two results along with experiments on real industrial data lead us to strongly believe that this is the case.

As the log-likelihood of the predictions increases after each iteration in step i, the variances and hence, the weight matrix approach the true weight matrix. Hence, the optimal weight matrix in step i represents the variances much more accurately than starting with the default, which is uniform weights.

Remark 2 *Using the optimal weight matrix and target of step i as a starting point for step $i+1$ has a higher log-likelihood solution than starting with the defaults, i.e., uniform weight matrix and the original target (Pregibon, 1981; Wang, 1987; McCullagh and Nelder, 1990).*

Based on Remark 2, one might ask, does starting from a higher likelihood point guarantee us faster convergence or fewer iterations? Loosely speaking, in the proposition that follows, we show that better initialization leads to no worse a solution after the same number of iterations.

Proposition 1 *When using IRLS, if two feasible points β_1 and β_2 are in the neighborhood of the same local optimum γ of a log-likelihood function where the Hessians exist and the first partial derivatives are non-zero, with $\mathcal{L}(\beta_1|X,Y) \geq \mathcal{L}(\beta_2|X,Y)$, then a tight upper bound on the distance of the solutions starting at each of these points to γ after t iterations has the following relationship, $\eta_t(\beta_1) \leq \eta_t(\beta_2)$, where $\eta_t(.)$ is a function that takes as input the starting point and outputs the required upper bound on the distance at iteration t.*

Proof Let γ be the root of the log-likelihood function $\mathcal{L}(.)$. The log-likelihood is always conditioned on inputs and outputs but for notational conciseness we do not repeatedly write this here. Let β_j^t denote the solution of IRLS at iteration t starting from the initial point β_j. Thus, by Taylor expansion starting from β_1 at iteration t we have

$$\mathcal{L}(\gamma) = \mathcal{L}(\beta_1^t) + \bigtriangledown\mathcal{L}(\beta_1^t)^T(\gamma - \beta_1^t) + \frac{1}{2}(\gamma - \beta_1^t)^T F(\beta_1^t)(\gamma - \beta_1^t),$$

where $\bigtriangledown$ denotes the gradient and $F(.)$ denotes the hessian. Since, γ is the root of the above function and after pre-multiplying by the inverse of the Jacobian at β_1^t we have

$$\begin{aligned}\gamma - \beta_1^t + J^{-1}(\beta_1^t)\mathcal{L}(\beta_1^t) &= -\frac{1}{2}J^{-1}(\beta_1^t)(\gamma-\beta_j^t)^T F(\beta_1^t)(\gamma-\beta_1^t)\\ \gamma - \beta_1^{t+1} &= -\frac{1}{2}J^{-1}(\beta_1^t)(\gamma-\beta_j^t)^T F(\beta_1^t)(\gamma-\beta_1^t)\end{aligned},$$

since, $\beta_1^{t+1} = \beta_1^t - J^{-1}(\beta_1^t)\mathcal{L}(\beta_1^t)$. Now if we take norm on both sides we get

$$|\gamma - \beta_1^{t+1}| = \frac{1}{2}|J^{-1}(\beta_1^t)F(\beta_1^t)||\gamma-\beta_1^t|^2.$$

Let $\Delta_1^t = |\gamma - \beta_1^t|$. Thus

$$\Delta_1^{t+1} = \frac{1}{2}|J^{-1}(\beta_1^t)F(\beta_1^t)|{\Delta_1^t}^2.$$

Let $\mathcal{N}_j$ denote the neighborhood around γ such that $\mathcal{N}_j = \{\beta|\mathcal{L}(\beta) \geq \mathcal{L}(\beta_j)\}$. Let $u_j = sup_{\beta\in\mathcal{N}_j}\frac{1}{2}|J^{-1}(\beta)F(\beta)|$. We thus have

$$\begin{aligned}\Delta_1^{t+1} &\leq u_1{\Delta_1^t}^2\\ &\leq u_1^t{\Delta_1^1}^2 = \eta_{t+1}(\beta_1)\end{aligned},$$

where $\eta_{t+1}(\beta_1) = u_1^t{\Delta_1^1}^2$. Similarly, if we started at β_2 we would have

$$\Delta_2^{t+1} \leq \eta_{t+1}(\beta_2).$$

Now if $\mathcal{L}(\beta_1) \geq \mathcal{L}(\beta_2)$, then $\mathcal{N}_1 \subset \mathcal{N}_2$ and hence, $u_1 \leq u_2$. Moreover, $\Delta_1^1 \leq \Delta_2^1$. Based on these two facts we would have

$$\eta_{t+1}(\beta_1) \leq \eta_{t+1}(\beta_2).$$

■

The assumptions in Proposition 1 about local Hessians and the Jacobian matrix are standard assumptions used in proving convergence of Newton-Raphson's method (Luenberger, 1984). These are not additional assumptions that we make.

Based on Remark 2 and Proposition 1, we can say that starting from the optimal weight matrix and target of the previous step i will lead to fewer iterations in the current step $i+1$, which means faster convergence.

Note that our suggested technique is also more efficient than doing a warm start in the current step using the optimal weight matrix and target from the previous step. This is because even in the worst case where none of the conditions are satisfied, finding the solution of the corresponding WLS problem in the current step is, as discussed before, more efficient using Algorithm 1 rather than learning from scratch.

3.3 Other Regression Techniques

In the previous sections, we showed that using Algorithm 1 just by itself or using it as a core function in other algorithms can lead to optimally and efficiently solving regression problems that use a rich class of regression techniques. This naturally implies that questions I(a), I(b) and I(c) in Section 2 have been answered positively for these cases. In this section, we try to answer these questions for arbitrary linear or non-linear regression algorithms.

If we again use Algorithm 1 for an arbitrary regression algorithm, it may very well provide accurate predictions but is likely to be inefficient. The most expensive computation in Algorithm 1 would be computing the residual of fitting X_i onto each column of x_{i+1}. This could potentially lead to running the chosen regression algorithm $d_{i+1} - d_i$ times. We want to save on these computations and therefore, we relax Algorithm 1, where we simply fit x_{i+1} to $R^Y_{X_i}$ rather than fitting $R^{x_{i+1}}_{X_i}$ to $R^Y_{X_i}$. Let us refer to this new version of Algorithm 1 by Algorithm $1^{(r)}$. Let us now see if Algorithm $1^{(r)}$ also results in a positive response to questions I(a), I(b) and I(c).

Let us denote the error or residual of the optimal model built in step i (previous step) by δ_{prev}. If AFOR optimizes the residual of this model—which leads to the best AFOR model—based on x_{i+1}, then we denote its error by δ^{bst}_{AFOR}. Let the error of our relaxed method be denoted by $\delta_{1^{(r)}}$. With this, we have the following result,

Proposition 2 *Algorithm $1^{(r)}$ in step $i+1$ is no less accurate than the model in step i and AFOR in step $i+1$, i.e., $\delta_{1^{(r)}} \leq \delta^{bst}_{AFOR} \leq \delta_{prev}$.*

Proof We have $\delta_{prev} = R^Y_{X_i}$ and $\delta^{bst}_{AFOR} = R^{R^Y_{X_i}}_{x_{i+1}}$. Thus, δ^{bst}_{AFOR} can be written as, $\delta^{bst}_{AFOR} = R^{\delta_{prev}}_{x_{i+1}}$. Since in AFOR we optimize the residual of the model at the previous step, we have

$$\delta_{prev} \geq \delta^{bst}_{AFOR}.$$

In our relaxed method, we first fit x_{i+1} to $R^Y_{X_i}$. We then fit X_i to $Y^{(r)} = Y - \hat{Y}^{R^Y_{X_i}}_{x_{i+1}}$ and whose error maybe denoted by $\delta_{1^{(r)}} = R^{Y^{(r)}}_{X_i}$. Thus, $\delta_{1^{(r)}}$ can be written as, $\delta_{1^{(r)}} = R^{\delta^{bst}_{AFOR}}_{X_i}$ and since we optimize the residual of AFOR we have

$$\delta^{bst}_{AFOR} \geq \delta_{1^{(r)}}.$$

■

Proposition 2 implies a positive response to questions I(a) and I(b). It is also easy to see that Algorithm $1^{(r)}$ is more efficient than learning over the whole space and hence, we also have a positive response for I(c). If we further relax Algorithm 1 to exclude fitting X_i to $Y^{(r)}$, then our method is reduced to AFOR.

4. Addressing Question II

In the previous sections we provided techniques to update an existing model based on a new set of available features. In this section, we want to tackle the complimentary problem of updating a model in a particular step based on a new batch.

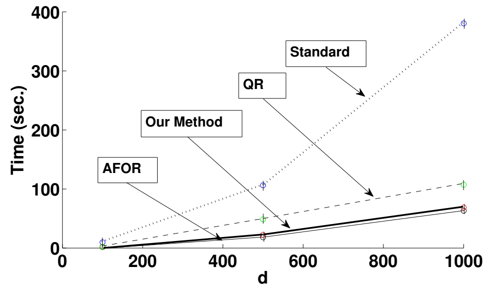

Figure 3: Total training time for OLS summed over the three steps for the different methods with different number of features d added at each step.

There are numerous works in online learning (Blum, 1996; Smale and Yao, 2005; Bottou and Cun, 2003), where existing models are updated using stochastic techniques in time proportional to the newly added information. A common update procedure for many parametric models is to perform additive updates. Thus, in step i and based on the data up until batch b, if we have a model M_i and we learn a model based on batch $b+1$, m_i^{b+1}, then the model based on data up until batch $b+1$ would be given by

$$M_i^{b+1} = M_i^b + \nu(m_i^{b+1} - M_i^b),$$

where $\nu^b \in [0,1]$ is the learning rate when batch $b+1$ becomes available. It is reasonable to make ν^b a function of N_{b+1} and $S = \sum_{j=1}^{b} N_j$, where the learning rate is higher when m_i^{b+1} is obtained by learning over a relatively larger data set. For example, $\nu^b = \frac{N_{b+1}}{S_{b+1}}$. Other possible updates are multiplicative updates (Arora et al., 2005), where the original model M_i^b can be updated based on its error in relation to the error of m_i^{b+1}.

5. Experiments

In this section, we empirically validate our claims through synthetic and real data experiments.

5.1 Synthetic Experiments

We consider the setting where we have three feature subsets or steps (i.e., $K = 3$) each of d dimensions. We generate data from a $3d+1$ dimensional Gaussian, where the first d dimensions make up the first representation X_1, the next d make up the next representation X_2, while the remaining but the last make up the third representation X_3. The final dimension corresponds to the target. We set the variances of each of the variables to 1. Since we want to model a realistic scenario, where the correlation between the target and the features is low and non-uniform, we set the correlation of the target with the features in the a) first representation to 0.1, b) second representation to 0.2 and c) third representation to 0.3. The other off-diagonal entries in the correlation matrix are set to 0.5. With this

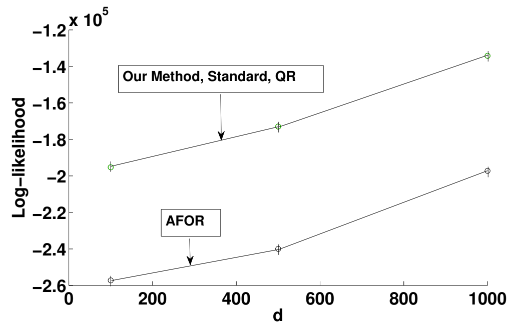

Figure 4: Average log-likelihood of the final model for the different methods with different number of features d added at each step for OLS.

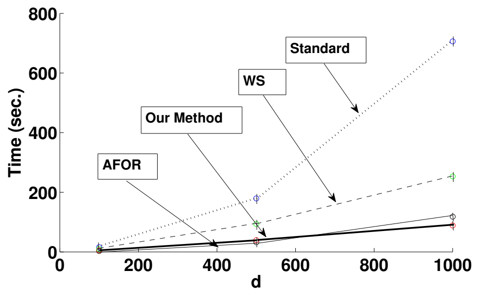

Figure 5: Total training time summed over the three steps for the different methods with different number of features d added at each step for logistic regression.

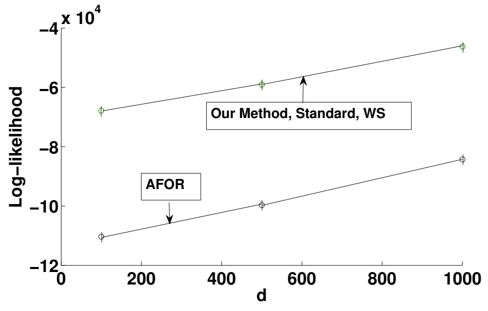

Figure 6: Average log-likelihood of the final model for the different methods with different number of features d added at each step for logistic regression.

correlation matrix and the mean set to zero we generate 100 data sets of 10000 points for each of the three values of d, namely: a) $d = 100$, b) $d = 500$ and c) $d = 1000$.

We compare the performance of our algorithms with a) learning from scratch i.e., standard, b) AFOR and c) QR decomposition based updates using Givens rotations (Stewart, 2001; Golub and Loan, 1996) or warm starting (QR and WS), for two regression methods namely, OLS regression and logistic regression ($\epsilon_\beta = \epsilon_\mathcal{D} = 0.1$). QR decomposition (Stewart, 2001) is only efficient in the OLS case since for logistic regression we have to perform the decomposition—not simply update—at every step as the optimal weight matrix and target keep changing. Thus, we use QR only for OLS, while warm starts are only applicable for logistic regression. For logistic regression, we discretize the target where we insert a value of 1 if the target has value ≥ 0, otherwise we insert a 0. We perform 10-fold cross validation and report the total training time summed over the three steps and the average test likelihood of the final model for each value of d with a 95% confidence interval.

In Figure 3, we see that the standard method takes significantly more time than the other methods. Our method is almost as fast as AFOR which is very promising and much faster than QR based updates. The performance gain of our method compared to the other optimal methods improves as d increases. In Figure 4, we see that AFOR is significantly worse in terms of accuracy than the other methods which are all optimal. In Figure 5, we see a similar trend as in the OLS case. However, the speedup of our method is greater and for the case with the highest d, our method is even faster than AFOR. A possible reason for this is that in one or more steps the updating using Algorithm 1 is much better than starting from the default even with just the added features. Here again, as seen in Figure 6, AFOR has much lower accuracy than the other methods that are optimal.

5.2 Real Data Experiments

We evaluate our methods on two real industrial data sets obtained from diverse domains. The first experiment is on a chip production data set obtained from the semiconductor industry. In this empirical study we compare the different updating strategies mentioned in this paper to learn a regression model. In the second experiment, we consider a finance data set, where we want to find which sources of information (or feature sets) are predictive of the final revenue. In this case, we run LogitGOMP (Lozano et al., 2011) to select the feature sets. When a feature set is selected, we update the current model using the different updating strategies. We then compare the runs of LogitGOMP based on these different strategies. Through this experiment we show that our method can also be effectively used to speed up sophisticated group variable selection techniques in real world settings. In both the experiments we add another straw man, which learns only on the features available at the particular step. We refer to this method, which learns only on the **a**dditional **f**eatures as AF.

5.3 Regression in Chip Manufacturing

We first provide some background of the chip manufacturing process and describe the general setup. We then discuss the major takeaways from the conducted experiments.

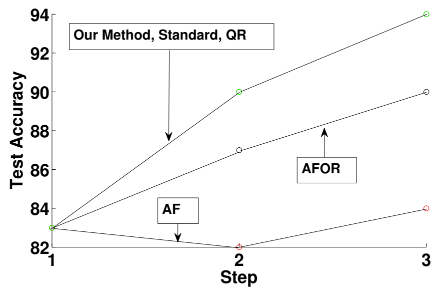

Figure 7: Test set accuracy of the methods in classifying wafers as in spec or out of spec at the respective steps for OLS.

77

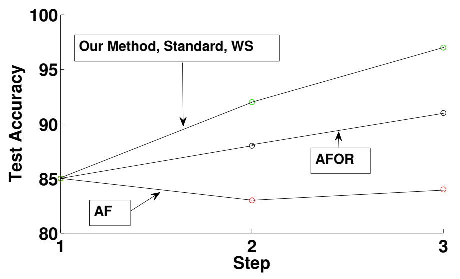

Figure 8: Test set accuracy of the methods in classifying wafers as in spec or out of spec at the respective steps for logistic regression.

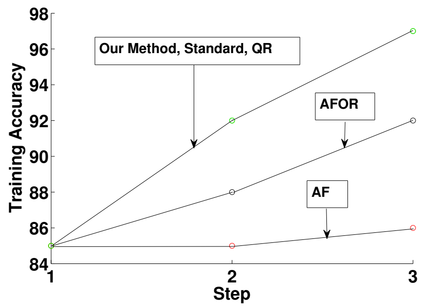

Figure 9: Training set accuracy of the different methods in classifying wafers as in spec or out of spec at the respective steps for OLS.

	Step 2			Step 3		
	Time	$\mathcal{L}_{tr}(.)$	$\mathcal{L}_{tst}(.)$	Time	$\mathcal{L}_{tr}(.)$	$\mathcal{L}_{tst}(.)$
Our Method	**20.6**	**-1991.8**	**-2156.2**	**17.8**	**-1742.1**	**-1895.5**
Standard	100.1	**-1991.8**	**-2156.2**	145.9	**-1742.1**	**-1895.5**
AFOR	**20.4**	-2693.4	-3482.5	**17.5**	-2176.7	-3285.2
AF	**20.2**	-3793.7	-4755.5	**17.4**	-3893.7	-4712.3
QR	72.3	**-1991.8**	**-2156.2**	67.5	**-1742.1**	**-1895.5**

Table 3: Average time (in sec.) the least squares methods take to train in each of the steps, the average training log-likelihood value $\mathcal{L}_{tr}(.)$ in the respective steps and the average test log-likelihood value $\mathcal{L}_{tst}(.)$ in the respective steps.

	Step 2			Step 3		
	Time	$\mathcal{L}_{tr}(.)$	$\mathcal{L}_{tst}(.)$	Time	$\mathcal{L}_{tr}(.)$	$\mathcal{L}_{tst}(.)$
Our Method	216.2	**-0.5**	**-1.7**	385.7	**-0.1**	**-1.2**
Standard	2085.8	**-0.5**	**-1.7**	3960.9	**-0.1**	**-1.2**
AFOR	**206.2**	-8.9	-12.8	**374.7**	-7.3	-11.4
AF	**206.2**	-20.9	-22.8	**374.4**	-27.3	-31.4
WS	556.7	**-0.5**	**-1.7**	791.2	**-0.1**	**-1.2**

Table 4: Average time (in sec.) the logistic regression approaches take to train in each of the steps, the average training log-likelihood value $\mathcal{L}_{tr}(.)$ in the respective steps and the average test log-likelihood value $\mathcal{L}_{tst}(.)$ in the respective steps.

5.3.1 Setup

We consider a real semiconductor process of microprocessor or chip production. In our data, a single data point is a wafer, which is a group of chips, and measurements which correspond to input features that are made on this wafer throughout its production. The target that is used to evaluate the quality of the wafer, in this case, is the speed of the wafer, which is the median of the speeds of its chips. Speed is usually used to diagnose the health of a wafer and needs to be within specifications. A slow wafer is undesirable for obvious reasons, but a fast wafer is also bad since it consumes too much power and can lead to overheating. The wafer speed prediction problem is quite challenging since the measurements are noisy and the speeds vary considerably from wafer to wafer relative to the spec. The wafers indicated as out of spec are usually discarded to save time and money in downstream processing. In some cases, they are re-routed for corrective action. In some isolated cases, wafers that are not too far out of spec may even be processed further and queued for low-end products. It takes a few months to produce a wafer as it goes through many complex high precision steps. Though the overall processing time maybe in the order of months, as each process takes a significant amount of time, the wafers move from one step to the next within a few minutes. It is thus important that we quickly estimate the

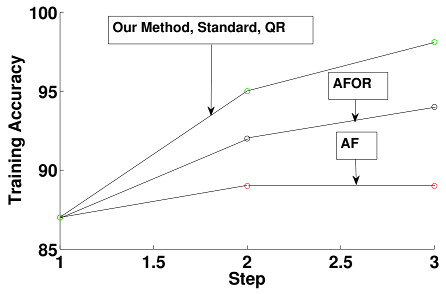

Figure 10: Training set accuracy of the methods in classifying wafers as in spec or out of spec at the respective steps for logistic regression.

speed after measurements from the finished step become available if we are to take any of the previously mentioned remedial actions.

We consider three critical steps in the manufacturing process with our data spanning over three batches. The first is the wafer polishing step referred to as chemical-mechanical planarization. Here the wafer is smoothed with removal of unnecessary material. During this step, various pressures—viz. condition head pressure, head zone pressures, etc.—and torques are measured indicating the amount of force the wafer is subjected to. The second step we consider is the etching step, where any remaining abnormalities in the photo-resistance on the wafer are removed by plasma ashing. Here the quantity and temperature of the plasma are controlled among other things and corresponding temperatures, pressures and concentrations are measured. The third and final step we consider is the rapid thermal processing step. In this step the electrical properties such as the material dielectric are altered. To alter the electrical properties, the wafer is subjected to sudden temperature ramps at tightly controlled pressures and chemical flows. Hence, here ramp up temperatures, ramp up rates, cool-down rates and various pressures and flows are measured.

By the time the wafer reaches the first step, 2287 measurements are taken. By the second step we have 3317 measurements. Finally we have 4284 measurements by the third step. Each of the three batches have 8926 wafers. In each of the steps, we train over the first batch and test over the second. We then train over the second batch and test over the third. We report the average training time and the average train and test log-likelihoods[3] for each step in Tables 3 and 4.

In this experiment, we compare the performance of our algorithms with a) learning from scratch, i.e., standard, b) AFOR, c) AF and d) QR decomposition based updates (Stewart, 2001) or warm starting (QR and WS), for two regression methods, namely, OLS regression and logistic regression ($\epsilon_{\beta} = \epsilon_{\mathcal{D}} = 0.1$) as in the synthetic case. The metric we use to evaluate the approaches is the time it takes for the local models to find the solution and the corresponding log-likelihood value at the solution. The higher the log-likelihood and the less time it takes to get to it, the better the method. For logistic regression, we discretize

3. We do not report the confidence intervals since, the variances are insignificant.

Approaches	Time	$\mathcal{L}_{tr}(.)$	$\mathcal{L}_{tst}(.)$
Our Method	**2.2** ± 0.1	**-561.3** ± 1.1	**-782.2** ± 1.3
Standard	10.4 ± 0.2	**-561.3** ± 1.1	**-782.2** ± 1.3
AFOR	2.9 ± 0.1	-605.2 ± 1.3	-841.6 ± 1.9
AF	2.8 ± 0.1	-1605.2 ± 1.8	-1841.6 ± 2.1
WS	7.8 ± 0.2	**-561.3** ± 1.1	**-782.2** ± 1.3

Table 5: Average running time (in seconds) of LogitGOMP, the average training log-likelihood value $\mathcal{L}_{tr}(.)$ and the average test log-likelihood value $\mathcal{L}_{tst}(.)$ with a 95% confidence interval.

the target based on specifications to denote either within spec by 1 or out of spec by 0. In all the three batches, roughly 20% of the wafers are out of spec.

5.3.2 Observations

In Tables 3 and 4, we see the results for step 2 and step 3 across the three batches. We do not show step 1 since we have to learn from scratch for all the models. The results show that our methods are optimal and significantly more efficient than the standard method. Our methods are also more efficient than QR and WS. AF and AFOR are efficient since they learn only on the additional features but are sub-optimal. Our methods are almost as efficient as AF and AFOR, which is very encouraging.

In Figures 7, 8, 9 and 10 we observe the test and training set accuracy in classifying wafers as within spec or out of spec based on the predictions obtained from OLS and logistic regression. The OLS predictions of the wafer speed are easily categorized into the two classes by considering the acceptable speed range given in the specification. In all the four figures we see that our method, which is optimal, performs significantly better than AFOR and AF as new features become available, even though it has comparable running time to both of them, as is seen in Tables 3 and 4. The poor performance of AF and AFOR indicates that the features across the various steps are correlated and are required to build an accurate predictive model.

5.4 Group Feature Selection in Finance

We consider a financial data set which is composed of three sources of information. Each of the three sources spans over two years and the data we have is at a weekly level. Thus, the number of data points is 104. The first source contains aging information of different deals, that is, how much time the different deals were in different financial stages before they were either won or lost. There are 105 features in this group. The second source is competitive information about different size bids made and how many of those were won. There are 21 features that characterize this group. The third source has information about important product launches over this two year period. There are five important product lines leading to five features. We want to find which of these groups have a significant effect in determining the revenue. Given that we run LogitGOMP, which learns a logistic regression model, we normalize the revenue, our target, using the sum over the two year period. We divide the data into 8 quarters and perform 8-fold cross-validation as predicting an entire quarter is of more interest than predicting randomly scattered weeks.

Table 5 shows that our method ($\epsilon_\beta = \epsilon_\mathcal{D} = 0.1$) is almost 5 times faster than the standard method and about 3.5 times faster than WS. Interestingly, it is also faster than AF and AFOR. The reason for this is that, in the second step, condition two in Algorithm 2 is satisfied and we have the optimal weight matrix and target from the previous update step in LogitGOMP, which is a better starting point than starting from the default during updates.

6. Discussion

It is important to note that the problem addressed in this paper is somewhat complimentary to stagewise learning. Stagewise learning mainly addresses the problem of feature selection, where coefficients are estimated for one feature at a time. Forward selection (Weisberg, 1980) and least angle regression (Efron et al., 2004) are examples of stagewise learning techniques. On the other hand, ours is a meta-learning technique—not limited to any particular regression algorithm—that considers all new features simultaneously with the final goal of learning a predictive model accurately and efficiently.

Our methods are also more general than the incremental feature learning method based on autoencoders to update an existing model(Zhou et al., 2012). This method makes the strong assumption of bounded input and output and the number of features considered at each step is a free parameter. The method is somewhat similar to AFOR as only new features are used to optimize the residual of the model learned over old features. Moreover, our methods are also more general than QR decomposition (Stewart, 2001), which is mainly used for efficient updates in OLS problems.

In the future, it would be interesting to update an existing model simultaneously based on added features and data points rather than doing it sequentially. One may be able to merge the ideas in this paper with the vast online learning literature. However, the main challenge in developing such a method would be guaranteeing accuracy while maintaining efficiency.

Acknowledgments

We would like to thank Katherine Dhurandhar for proofreading the paper.

References

S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta algorithm and applications. *Theory of Computing*, 8:121–164, 2005.

A. Blum. On-line algorithms in machine learning. In *Workshop on On-Line Algorithms, Dagstuhl*, pages 306–325. Springer, 1996.

L. Bottou and Y. Cun. Large scale online learning. In *Neural Information Processing Systems (NIPS)*, pages 77–84, 2003.

S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, NY, USA, 2004.

D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of Statistics*, 32:407–499, 2004.

G. H. Golub and C. F. Van Loan. *Matrix Computations.* John Hopkins University Press, 3rd edition, 1996.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2nd edition, 2009.

A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

A. Lozano, G. Swirszcz, and N. Abe. Grouped orthogonal matching pursuit for variable selection and prediction. In *Neural Information Processing Systems*, 2009.

A. Lozano, G. Swirszcz, and N. Abe. Group orthogonal matching pursuit for logistic regression. In *Artificial Intelligence and Statistics*, pages 452–460, 2011.

D. Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley; 2nd edition, 1984.

P. McCullagh and J. Nelder. *Generalized Linear Models, Second Edition.* Chapman and Hall, 1990.

D. Pregibon. Logistic regression diagnostics. *Annals of Statistics*, 9:704–724, 1981.

Y. Saad. Analysis of augmented Krylov subspace methods. *SIAM Journal on Matrix Analysis and Applications*, 18(2):435–449, April 1997.

S. Smale and Y. Yao. Online learning algorithms. *Found. Comp. Math*, 6:145–170, 2005.

G. Stewart. *Matrix Algorithms.* SIAM, 2001.

R. J. Tibshirani. Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

R. J. Tibshirani and J. Taylor. The solution path of the generalized LASSO. *The Annals of Statistics*, 39(3):1335–1371, 2011.

P. Wang. Residual plots for detecting nonlinearity in generalized linear models. *Technometrics*, 29(4):435–438, 1987.

S. Weisberg. *Applied Linear Regression.* New York: Wiley, 1980.

G. Zhou, K. Sohn, and H. Lee. Online incremental feature learning with denoising autoencoder. In *Artificial Intelligence and Statistics*, pages 1453–1461, 2012.