Adaptive Batch Size Selection in Active Learning for Regression

Dr Anthony L Faulds, PhD, Microsoft, Redmond, WA, 98052, anthony.l.faulds@gmail.com,
https://orcid.org/0000-0002-7209-2472

***Abstract*—Training supervised machine learning models requires labeled examples. A judicious choice of examples is helpful when there is a significant cost associated with assigning labels. This article improves upon a promising extant method – Batch-mode Expected Model Change Maximization (B-EMCM) method – for selecting examples to be labeled for regression problems. Specifically, it develops and evaluates alternate strategies for adaptively selecting batch size in B-EMCM.**

By determining the cumulative error that occurs from the estimation of the stochastic gradient descent, a stop criteria for each iteration of the batch can be specified to ensure that selected candidates are the most beneficial to model learning. This new methodology is compared to B-EMCM via mean absolute error and root mean square error over ten iterations benchmarked against machine learning data sets.

Using multiple data sets and metrics across all methods, one variation of AB-EMCM, the max bound of the accumulated error (AB-EMCM Max), showed the best results for an adaptive batch approach. It achieved better root mean squared error (RMSE) and mean absolute error (MAE) than the other adaptive and non-adaptive batch methods while reaching the result in nearly the same number of iterations as the non-adaptive batch methods.

***Index Terms*—Active learning (AL), batch mode, expected model change, linear regression, nonlinear regression.**

I. Introduction

Supervised machine learning for classification problems is used for real world applications. There are many useful techniques to train and to predict from labeled data. In an academic setting it is easy to devise supervised learning problems in which 100% of the labels are known. In real-world applications all labels are not known. It is costly and time consuming to derive the correct labels, often requiring human intervention. The problem is exacerbated if the data set is very large or if the labels require specialized skills, like determining bad behaviour in social media according to company policies or predicting health conditions that often require a nurse's or doctor's diagnosis.

In these problem areas where not all labels are known for training, semi-supervised learning is applied in different ways that span from passive learning, using predicted scores on unlabeled data in training, to active learning, finding the best unlabeled items to apply a human label. Basu used a form of active learning in clustering [1] [2].

Settles and Schwenker have a review of different forms of active and passive learning from unlabeled data [3] [4]. While active learning for categorical has been an active and highly developed area [5] [6] [7], Cai [8] points out that there is little work in active learning in regression.

Augmenting data with side information, like keywords on images, has been effective in associating unlabeled to labeled data [9] [10] for use in traditional supervised learning. Another approach is models trained on a subset of data to calculate predicted labels for unlabeled data and use those labels for retraining [11] [12] [13] [14]. In the realm of active learning, approaches for classification demonstrate how to pick the best items to label [15].

One approach that has proven to work well is the selection of unlabeled data to be labeled, active learning. This approach has been studied extensively for classification problems [16] [17] [18], but has limited coverage for regression problems [8].

In recent research [8], the concept of Expected Model Change Maximization (EMCM) was introduced, demonstrating how unlabeled data can be chosen using stochastic gradient descent (SGD). This method defines the chosen candidates to be labeled as ones that cause the largest change in linear regression generalized error with respect to the model parameters. They showed that this method of selection not only worked well with linear regression models, but also generalized to gradient boosted tree models.

Then Cai [8] took the research one step further introducing batch-mode EMCM (B-EMCM). Since it is not practical in the real world to label one item at a time and retrain, either because training takes too long or because in some instances there is access to many labelers who can label in parallel, the research was extended to select a batch of unlabeled data that had the largest effect on SGD while minimizing correlation between candidates. Essentially this algorithm selects the candidates with little to no duplicate information that will have the largest change in the model trained parameters.

The process of selecting candidates in batch is similar to the single item approach, but instead of labeling and learning from the newly labeled data, the process uses each accepted candidate to estimate a new model's generalized error. Adding a subsequent candidate takes into account previous candidates and an estimation of what is learned from them.

Cai [8] assumed a fixed batch selection size, k, and recognized through further research that selecting an adaptive batch size may increase the accuracy in selecting candidates that most affect generalized error. Since each element in a batch is selected based on the accumulated estimate of the change in model parameters, this estimate has error associated with it and, therefore, the accumulation of change has an accumulating error that grows with each addition to the batch. To address the accumulated error, this research examines several different approaches to selecting an adaptive batch size.

For EMCM, the selection of each element is based on the maximum stochastic gradient of a linear regression model. As each new element is introduced to the labeled data, the equations are reevaluated using the new element and its label. B-EMCM takes the base concept of EMCM and estimates the stochastic gradient of the k^{th} additional element. For the batch method, the labels are unknown until the batch is complete and annotated by humans based on stochastic gradient of the first $k-1$ elements. The purpose of this is to allow k elements to be identified without retraining or evaluating the changes of SGD linear regression model while taking into account the value of the k^{th} element relative to the first $k-1$ elements. Cai [8] identified an issue where "the estimate accuracy in the model change may decrease with the increase in the size of the batch, resulting in error accumulation." They go on to propose "one possible solution to this problem is to adaptively determine the batch size taking." The true label is unknown so this method must use some estimation of the true label in calculating the change in parameters. Calculating just one element of the batch introduces error. Calculating k elements of a batch introduces an accumulative and compounding error. Since there is diminishing accuracy of the stochastic gradient, then there is diminishing accuracy that the k^{th} element will yield the maximum model parameter change. Therefore, if an adaptive batch size can be determined, then the chance of selecting non-optimal elements is reduced. This works in two ways: 1) reducing the batch size when the accumulated error is big, which in turn reduces wasted effort in labeling suboptimal elements, and 2) increasing batch size when the accumulated error is small, to allow more labeling between retraining which can reduce the number of batch cycles. If the accumulated error is very small, the batch size can be increased to label more data in between each model retraining.

II. Related Work

A. Active Learning for Classification

Active learning algorithms have evolved to address problems with partially labeled data. The research by Campigotto [19] was used to solve many interesting problems or to readdress older problems with newer, less computationally intensive solutions. de Fortuny and Martens [20] used active learning to create interpretable models that explore and build out human understandable rules.

Uncertainty sampling has been shown as a more effective method than random sampling [16] [17]. For this, candidates are selected by least confidence. Query by Committee (QBC) shows the continued advance of active learning techniques [18]. Using multiple separate models trained on different sampling of the primary training data set, the result is multiple models, each with slightly different results. In classification problems, the candidates with the most disagreement across the models is the next item to be labeled. One form besides label disagreement is vote entropy. It uses the probability instead of just the predicted label. Similarly for regression, variance across model predictions is one of the more popular ways to pick up disagreement.

B. Active Learning for Regression

Although many problems can be mapped into categorical space, having the detailed information that numerical prediction can provide is useful. One challenging aspect of regression problems is that many of the active learning algorithms are tree-based and therefore non-parametric. Taking the first derivative, or calculating descent, can yield non-continuous results leading to results that do not converge to a solution.

Given continuous outputs, Freund [18] remarked that QBC could be used for regression. Yu [21] provided a method of passive sampling with which the feature space of the candidates is used to select the best candidate to label. This method was more efficient as it did not require the calculating or refreshing of the model. Cai [22] presented an approach to regression problems that labels the example leading to the largest model change.

C. Batch vs Sequential

Most of the research in active learning centers around sequential labeling. In practice, it can be very slow to annotate one candidate at a time and is more cost efficient to label in batches. Sequential labeling is more optimal than batch because information about the chosen item can immediately be used to select the new candidate. Practical considerations make it desirable to annotate candidates in batches. Because batch cannot outperform sequential, the unconstrained optimal solution for batch is to select batch sizes of a single item.

D. EMCM Extended Work

Wu extends the EMCM method by defining essential criteria of informativeness, representativeness and diversity. Recognizing that B-EMCM only addresses informativeness, Wu created Enhanced Batch-Mode for QBC and EMCM that selects a better initial set of labeled data and removes outliers with every iteration [23] [24] [25] [26]. This idea of accounting for representativeness and diversity has been applied to greedy sampling [27]. These methods still approach the problem with fixed batch sizes.

Zhang did related work to their B-EMCM work but concentrated on stopping criteria for batch methods using logistic regression and support vector machines [28]. The work demonstrates the value of adding stopping criteria to batch algorithms. Farkis shows the importance of stopping criteria for a combination of active learning and semi-supervised learning methods [29] [30].

EMCM and its variant B-EMCM has relevance to today's problems and is used in practical applications. OneStopTuner is an example where B-EMCM is used to autotune JVM flags for optimal Apache Spark runs [31]. ChemML is an example of a library built to handle specific problems of prediction in chemical and material domain that implements the EMCM method [32]. EMCM method has been combined with regression trees to reduce the number of field experiments needed for production A/B test rollout [33].

doi:10.20944/preprints202201.0441.v1

III. Adaptive Batch Expected Model Change Maximization

Batch mode algorithms, like EMCM, have been optimized further by adaptive batch sizing. Chakraborty [34] introduced a batch mode active learning (BMAL) framework that combined batch size and candidate selection into a single algorithm. By varying the batch size, the error can be reduced in fewer iterations than the non-adaptive batch method while maintaining the same computational complexity. This indicates some of the direction of this research to explore adaptive batch for B-EMCM.

A. EMCM Method

The goal of EMCM is to select the unlabeled item that reduces the error between predicted values $f(x_i)$ and true values y_i. The first step is to begin with L defined as the loss function over the labeled training set D. This is defined as general loss, ϵ, which is a generic definition of the loss function for supervised machine learning algorithms.

$$\epsilon = \frac{1}{2}\sum_{i=1}^{n} L(f(x_i), y_i)^2 \tag{1}$$

SGD is defined by Equation 2 where α is the learning rate and Θ is a vector of model parameters.

$$\Theta_{new} \leftarrow \Theta - \alpha\frac{\delta L_{xi}(\Theta)}{\delta\Theta}, 1, 2, ..., n \tag{2}$$

Using SGD, the change in parameters, C_Θ, based on a chosen candidate, x^+, can be approximated by the single value of this sequence.

$$C_\Theta(x^+) = \Delta\Theta \approx \alpha\frac{\delta L_{x^+}(\theta)}{\delta\theta} \tag{3}$$

The best candidate, x^+, is chosen from the unlabeled data set, U, where there is the largest change in the model.

$$x^* = \underset{x \epsilon U}{\operatorname{argmax}} \|C_\theta(x)\| \tag{4}$$

This is an estimation based on the current parameters, but the actual change in parameters is based on all the labeled data and several iterations of this calculation. The assumption to make this valid is the change with respect to x^+ is much greater at the current Θ then all other labeled data, $x\epsilon D$.

The next step is to simplify the system and start with linear regression, where x is a vector of the features $x_0 = 1$.

$$f(x;\Theta) = \sum_{i=0}^{p}\Theta_i x_i = \Theta^T x \tag{5}$$

EMCM method determines which unlabeled item will cause the largest reduction in generalized error, with respect to change in parameters. For linear regression the squared error loss is used for the generalized error. ϵ_D is the error over the training set, y_i is the true value of x_i.

$$\epsilon_D = \frac{1}{2}\sum_{i=1}^{n}(f(x_i) - y_i)^2 \tag{6}$$

Adding an element from the unlabeled data set to the labeled data set results in a squared error loss:

$$\epsilon_{D^+} = \frac{1}{2}\sum_{i=1}^{n}(f(x_i) - y_i)^2 + \frac{1}{2}(f(x^+) - y^+)^2 \tag{7}$$

Therefore, the largest change in linear regression parameters using SGD is:

$$\begin{aligned} \frac{\delta L_{x^+}(\theta)}{\delta\theta} &= (f(x^+) - y^+)\frac{\delta f(x^+)}{\delta\theta} \\ &= (f(x^+) - y^+)\frac{\delta\theta^T x^+}{\delta\theta} \\ &= (f(x^+) - y^+)x^+ \end{aligned} \tag{8}$$

Since y is not explicitly known, the purpose of the system is to select the next candidate to determine y; then an ensemble of bootstraps $B(Z) = f_1, f_2, ..., f_z$ is used to estimate $y^+ = y_1, y_2, ..., y_z$. The candidate that results in the largest change in parameters becomes:

$$x^* = \underset{x \epsilon U}{\operatorname{argmax}} \frac{1}{Z}\sum_{z=1}^{Z}\|f(x) - y_z(x))x\| \tag{9}$$

where Z are models trained with bootstrap data from D, averaged to estimate the possible label for each x value. This is the single item EMCM step finding the next unlabeled item to label.

IV. B-EMCM Method

This article extends the B_k-EMCM model to develop AB-EMCM. By including an adaptive batch, each step maximizes the derivative of the loss function. Bounds placed on the loss function, or on the batch size, stop it from going to one unlabeled item. This would default the solution to the original EMCM model. Despite this interesting result, it defeats the purpose of building a practical system for use with real world problems.

The B-EMCM algorithm changes this logic into a batch process. After the first item is selected, the linear regression model changes. That change is estimated by Equation 2:

$$f_*(x) = \theta_*^T x \approx \left(\theta - \alpha\frac{\delta L_{x^*}(\theta)}{\delta\theta}\right)^T x \tag{10}$$

The derivation of the second item's derivative of change of parameters is:

$$\begin{aligned} \frac{\delta L_{x_2}(\theta|(x_*, y_*))}{\delta\theta_*} &= (f_*(x_2) - y_2)x_2 \\ &\approx ((\theta - \alpha\frac{\delta L_{x_*}(\theta)}{\delta\theta})x_2 - y_2)x_2 \\ &= (\theta^T x_2 - y_2 - \alpha(\frac{\delta L_{x_*}(\theta)}{\delta\theta})^T x_2)x_2 \\ &= (f(x_2) - y_2)x_2 - \alpha(\frac{\delta L_{x_*}(\theta)}{\delta\theta})^T x_2 x_2 \\ &= \frac{\delta L_{x_2^+}(\theta)}{\delta\theta} - \alpha(\frac{\delta L_{x_*}(\theta)}{\delta\theta})^T x_2 x_2 \end{aligned} \tag{11}$$

This demonstrates that the derivative of the change of parameters at the second candidate x_2 is obtained with the current model $f(x)$ without retraining, meaning new parameters of Θ need not to be calculated with each selection in the batch. Each subsequent selection is selected based on:

$$C(x_j^+, b_{j-1}^*) = \alpha \sum_{i=1}^{j-1} \left(\frac{\delta L_{x_i^*}(\theta_* | b_{i-1}^*)}{\delta \theta_*} \right)^T x_j^+ x_j^+ \qquad (12)$$

$$\frac{\delta L_{x_j^+}(\theta_* | b_{j-1}^*)}{\delta \theta_*} \approx (f(x_j^+) - y_j^+) x_j^+ - C(x_j^+, b_{j-1}^*) \qquad (13)$$
$$j = 1, 2, ..., k, 1 \leq i \leq j-1$$

$$x_j^* = \underset{x \epsilon U \backslash b_{j-1}^*}{\operatorname{argmax}} \frac{1}{Z} \sum_{z=1}^{Z} ||f(x) - y_z(x))x - C(x, b_{j-1})|| \qquad (14)$$
$$j = 1, 2, ..., k, 1 \leq i \leq j-1$$

Algorithm 1 B-EMCM for Linear Regression

```
Input: $D = (x_i, y_i)_{i=1}^n$ Initial labeled data set, $U$ - Unlabeled
    data set, $B(Z)$ - Ensemble of linear regression models
    based on bootstrap $D$, $k$ - Size of batch
Initialize $b = \phi$
Calculate ensemble bootstrap $B(Z)$ based on labeled data
while $|b| < k$ do
    for each $x$ in $U$ do
        $y_1, y_2, ..., y_z \leftarrow B(Z)$
        Calculate the derivative using Eq. 13
        Eq. 14 to calculate estimated model change
    end for
    Select $x^*$ having the greatest model change
    $U \leftarrow \backslash x^*, b \leftarrow b \cup x^*$
end while
Output: $b = \{x_1^*, x_2^*, ..., x_k^*\}$
```

A. AB-EMCM for Linear Regression

Batch mode algorithms, like EMCM, have been optimized further by adaptive batch sizing. Chakraborty [34] introduced a batch mode active learning (BMAL) framework that combined batch size and candidate selection into a single algorithm. By varying the batch size, the error can be reduced faster than the non-adaptive batch method while maintaining the same computational complexity. This indicates the direction of this research to explore adaptive batch for B-EMCM.

The process of selecting items in the batch is similar to the single item approach, but the effect of adding each unlabeled item to the labeled set is unknown. Cai [8] built an estimation of the change to calculate the effect of each subsequent addition. This change considered some of the item-to-item correlation, so the batch did not include similar candidates.

They compared three new techniques to the existing one. The first was the ideal of batch size equal to one (N_1-EMCM), which defaulted to the basic EMCM that they introduced in the previous research; no batching, just single element retraining. The second they compared to the naively derived N_k-EMCM, which follows N_1-EMCM, but instead of selecting one element for a batch, it selected k elements. Unfortunately, this approach did not take into account correlation or information overlap of elements inside the batch. The third was the B-EMCM method. This was similar to N_k-EMCM, yet it included an estimation of how each item would affect the model. Therefore, it attempted to remove items that are correlated or could have added the same information to the model. This research showed the effectiveness of the batch approach that did not require recalculation of the model parameters in order to obtain each member of the batch. The batch approach outperformed other algorithms like random selection, Greedy selection, QBC, bias variance, full variance and EMCM selecting the top k per loop.

The previous research used a static value for k. Similar to other algorithms that have a batch, iteration, or step size, calculating an optimal dynamic batch size is shown to handle a larger set of problems without prior knowledge.

Algorithm 2 conveys the steps for the new adaptive B-EMCM (AB-EMCM). The items of change are both Step 3 and Step 10. During each loop the stop criteria was evaluated and, if a threshold were reached, the loop immediately ended and the batch contained all items added up to that point.

Algorithm 2 AB-EMCM for Linear Regression

```
Input: $D = \{(x_i, y_i)_{i=1}^n\}$ Initial labeled data set, $U$ - Unlabeled
    data set, $B(Z)$ - Ensemble of linear regression models
    based on bootstrap D
1:  Initialize $b = \phi$
2:  Calculate ensemble bootstrap $B(Z)$ based on labeled data
3:  while not STOP CRITERIA do
4:      for each $x$ in $U$ do
5:          $\{y_1, y_2, ..., y_z\} \leftarrow B(Z)$
6:          Calculate the derivative using Eq. 13
7:          Eq. 14 to calculate estimated model change
8:      end for
9:      Select $x^*$ having the greatest model change
10:     CALCULATE STOP CRITERIA
11: end while
12: Output: $b = \{x_1^*, x_2^*, ..., x_k^*\}$
```

V. Stop Criteria for Batch Size

The methods explored to evaluate the effectiveness of adaptive batch size included selecting batches that were potentially better than the B-EMCM method. Algorithm 2, Line 3 was replaced with specific stop criteria: instead of using a fixed k, the stopping points for a single batch were: linear increase (Linear+), linear decrease (Linear-), max bound (Max), relative change (Rel), and estimated vs actual (EVA).

Algorithm 2 conveys the steps for the new adaptive B-EMCM (AB-EMCM). The items of change are both Step 3 and Step 10. During each loop the stop criteria was evaluated and, if a threshold were reached, the loop immediately ended and the batch contained all items added up to that point.

A. Linear Increase/Decrease Stop Criteria

For a simplified version of stop criteria, an algorithm was applied that worked for most problems. Intuitively, the expectation was with few labeled data, the first data points would change the model parameters more drastically than the ones in the final batch. This method began with a lower number of labeled data added per iteration and linearly increased the count. The expectation was that the first iteration would take a more cautious step, adding fewer examples and acting closer to the single EMCM algorithm; with each iteration taking bigger steps and acting more like the B-EMCM algorithm. Cai [8] showed that N_1-EMCM, which is the single item EMCM method, performed better than B-EMCM in all tests, as an estimation was used in calculating which item to add to the batch, and the error in the estimation increased with the size of the batch. This stop criteria started with a training size of 1.8%, which is under 3% used for the static batch size algorithms, and increased by 0.24% with each iteration; the 10th iteration finished with a total of 4.2% labeled data.

To demonstrate the differences, the same algorithm was used, yet decreasing the batch size with each iteration. This version of the stop criteria began 4.2% above the 3% of the batch algorithms and decreased by 0.24% each iteration to have the 10th iteration finish with 1.8% labeled data. These two simple versions were used as a baseline for adaptive batch and as a comparison for the other versions that reacted to either each iteration or the introduction of increasing error. These two were bound to ten iterations for comparison to the static batch algorithms, and still included adding in 30% labeled data over those ten iterations. The other algorithms, described below, were constrained by including 30% labeled data over the iterations. They took more, or less, than the ten iterations to attain the final stop.

B. Max Bound Stop Criteria

For max bound, when the accumulated error exceeded a set threshold, the current iteration was terminated and the batch was complete. Equation 4 shows that the best candidate was based on the one that had the largest change in the parameter vector. That calculation was based on Equation 3 that introduced an estimation of the parameter change. This stop criteria required the calculation of the sum of the total possible change. The concept was to make each batch change the model in equal amounts. If there were some candidates that created larger changes, only a few were used. If the candidates created a small amount of change, more were included in the batch. This change was an estimate. If the accumulated size was large, it was reasonable to assume that it increased the error in the change calculation proportionally. Therefore, creating batches where the error was similar made for better selection.

Calculating a maximum bound for the error introduced by the assumption that the change in the parameter vector for adding a single element using SGD was approximately equal to the change in parameter vector based on one iteration of the single element. Equation 2 shows that one step in SGD was the iteration of Equation 3 overall of the training data. Assuming the parameters were in a local minimum based on the training data, this loop should have resulted in zero change and adding one more item to the training data caused the first step in the iteration of SGD to be equal to Equation 3. Yet several iterations of the algorithm often resulted in a different change in parameters, which was caused by the method being stochastic.

The actual value of θ when x^+ was added shown in Algorithm 3.

Algorithm 3 Comparison of $\delta\theta$ estimate and actual.

```
One iteration of SGD
Sample list of training data $L^+ = L \cup x^+$
$\theta = \theta_0$
**for** each $x$ in sample($L^+$) **do**
    $\theta = \theta - \alpha \frac{\delta L_{x^+}(\Theta)}{\delta \Theta}$
**end for**
$\delta = \theta - \theta_0$
Estimated value of theta for B-EMCM
$\theta = \theta_0$
$\theta = \theta - \alpha \frac{\delta L_{x^+}(\Theta)}{\delta \Theta}$
$\delta = \theta - \theta_0$
```

When added to the labeled data set, one item in the unlabeled data set took one to two iterations to find a local minimum of the squared error, while another took several more iterations and resulted in a larger overall change to the parameters. If a maximum estimated error was derived, this was used as a stop criterion to reduce accumulated error in the estimation and guarantee selected candidates were accurately chosen. If the estimated error grew too large, the B-EMCM equations were no longer accurate for finding the best candidate.

Equation 15 was used to calculate the max bound. x_n was the candidate selected in each iteration and d_n was the expected change in each iteration of selection. When the bound, D exceeded a prescribed value, the batch iteration ended.

$$\begin{aligned} d_1 &= \max_{x \epsilon U} \frac{1}{Z} \sum_{z=1}^{Z} \|(f(x) - y_z(x))x\| \\ x_1 &= \operatorname*{argmax}_{x \epsilon U} \frac{1}{Z} \sum_{z=1}^{Z} \|(f(x) - y_z(x))x\| \\ U &\leftarrow U \setminus x_1 \\ D &= \sum_{n=1}^{N} d_n \end{aligned} \tag{15}$$

C. Relative Change Stop Criteria

The concept of this stop criteria was to batch changes together that were relatively the same level. For most models, as more candidates were selected and labeled, there were diminishing returns to label subsequent candidates. In a batch, the first element was selected, then the second, to the j^{th} element. For this stop criteria the ratio of the change of the first element over the change of the j^{th} element was calculated.

If this ratio exceeded a specified threshold, the batch was complete.

$$R_j = \frac{C(x_1^*)}{C(x_j^*)} = \frac{d_1}{d_j} \tag{16}$$

If the ratio of model change reached a certain threshold, the error in the first element was significantly larger than the change of the x^* value. Significance, in this case, was determined by defining the ratio. Therefore, one iteration of the batch process was complete. Each batch introduced items that had similar relative influence on the model. When the change in parameters went above a prescribed threshold defined for R_j, then the new items were an order of magnitude lower in error. For this research the threshold for R_j was set at 10. This method included items of similar change to the model parameters. For an item that had a change that was ten to one hundred times larger than another change, the error of that estimate was approximately at the same scale as the change of the second item. Therefore, items with a similar scale of changes are grouped and the batch iteration is terminated before moving on to the next tier which is an order of magnitude smaller.

The detailed equation for this method is shown in Equation 17:

$$\frac{(f(x_1) - y_1)x_1}{(f(x_j) - y_j)x_j - \alpha \sum_{i=1}^{j-1}(f(x_i) - y_i)x_i x_j x_j} \tag{17}$$

When this value exceeded a prescribed ratio, then the batch was complete for that iteration.

D. Estimated vs Actual Stop Criteria

As Cai [8] suggested in further research, a stop criteria could be formed that uses estimate of the change in θ and actual change in θ after each batch iteration to determine the batch size of the next step. A large difference indicated that the estimation was far from actual, and thus the batch size had to be reduced. A small difference indicates that the estimation matched the actual change and the batch size was increased.

This method began with an initial batch size, k_1. The value of k was adjusted by comparing the estimated change in θ, the model parameters, with the actual change of the model parameters, as described in Algorithm 3. If this value was zero, than the estimate was the same as the actual change, in which case the batch size was increased, k_{n+1}. If the value was significantly positive or negative, it indicated the estimate in change was incorrect, which invalidated the B-EMCM candidate selection logic and assumptions. Therefore, the batch size, k_{n+1}, was decreased in order to increase the chance of the estimate being equal to the actual parameter change. To keep the first iterations of this algorithm simple, this approach increased the batch size by 25% if the difference was within a prescribed amount. If the difference exceeded the prescribed amount, then the batch size decreased by 25%.

Algorithm 4 One loop of estimated vs actual

```
Require: $\gamma \in (0,1)$ - batch scaling, $\Delta\theta_t$ - Threshold for parameter change difference
$k = k_1$ (an initial batch size)
$\delta\theta_e = 0$
for loop $k$ do
    $\delta\theta_e$ += $max\frac{1}{Z}$ $\sum_{z=1}^{Z}(f(x_j) - y_j)x_j - \alpha \sum_{i=1}^{j-1}(f(x_i) - y_i)x_i)x_j x_j$
    $\theta_b$ = current model parameters
    Retrain model with selected batch
    $\theta_{b+1}$ = model parameters after training
    $\Delta\theta = |\theta_{b+1} - \theta_b - \delta\theta_e|$
end for
if $\Delta\theta > \Delta\theta_t$ then
    $k = k/\gamma$
else
    $k = k\gamma$
end if
```

Data set	Num Examples	Num Features	Source
Concrete	1,030	8	UCI
CPS	537	8	StatLib
Forest	517	10	UCI
Housing	506	13	UCI
PM10	500	7	StatLib
Redwine	1,599	11	UCI
Whitewine	4,898	11	UCI

TABLE I
STATISTICS ABOUT EVALUATION DATA SETS

VI. EXPERIMENTS

A. Data Sets and Experimental Setups

The algorithms were validated using data sets from UCI and StatLib. These cover a range of regression data types and have been used by the active learning community to validate new models and processes. Table I shows the data sets by name, along with the data set size and number of features.

B. Comparison Methods and Evaluation Metrics

Starting with the B-EMCM approach used by Cai [8], this research replicated it to verify the claims, then a baseline model was built for variation comparisons. This research explored an approach for adaptively selecting batch sizes for the EMCM algorithm applied to linear and non-linear regression. It provided a selection of possible adaptive selection algorithms and defined which was best by using standard data sets measuring its MAE and RMSE and the number of iterations that attained the smallest error.

$$MAE = \frac{\sum_{i=1}^{n}|y_i - x_i|}{n} \tag{18}$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - x_i)^2}{n}} \tag{19}$$

As discovered in Chai [35], results based on both MAE and RMSE added similar but varied information related to average, variance and general distribution of the results. Therefore, this research used both the analysis and comparison of the results.

To make the methods more comparable, each run across different methods started with the same labeled, unlabeled and test data sets. Different data sets were used across different runs to study whether the method was robust under different starting conditions. Each method ran ten times with different sets, yet both the first and second methods had the same starting conditions in their respective first runs, as well as in their second runs.

The two measures used to evaluate the benefits of this system were MAE and RMSE. These were evaluated over each iteration of the batch method and calculated on the test set. These values reduced the MAE or RMSE more quickly in early iterations; yet comparing after the tenth iteration, it remained superior.

Because this was a continuation of other work intending to compare methods, this research used the same data sets as Cai [22] [8]. These data sets were studied thoroughly and provided a valid set of variation showing the ability of the algorithm. They have been used extensively in regression analysis as shown by Dong [36].

Several methodologies were examined for selecting batches per iteration. A pitfall of the optimization was strictly minimizing MAE. When this was done, the optimal batch size of $k = 1$ resulted. This was shown in [8] work and was intuitive because $k = 1$ batch size used exact calculation of change in the model parameters with each label chosen, as opposed to batch which estimated each subsequent step.

To prepare this data for linear models, several transformations were applied. Each set had categorical and numerical variables. For numerical variables, the data was normalized using Equation 20.

$$x = \frac{(x - x_{min})}{(x_{max} - x_{min})} \tag{20}$$

For categorical variables, if there were only two categories, it was converted to 1 and 0, where each represents one of the categories. If there were more than two categories, the values were hot encoded. The method for hot encoding was to create n new variables, where n was the number of categories. Each variable represented one of the categories and was set to 1 while the rest were set to 0. For instance, take the variable occupation from the current population survey (CPS) data set which has values 1 for *Management*, 2 for *Sales*, 3 for *Clerical*, 4 for *Service*, 5 for *Professional*, and 6 for *Other*.

Direct comparison of batch method and adaptive batch method was not possible. Using the earlier work [8], the unit of the x-axis for the different runs was the number of iterations. If an iteration number was used, adaptive batch does not mean the same number of training samples were used per iteration. It could have included all unlabeled data in the first iteration, which would not have allowed further iterations. It could have also only included one additional training data item, which was the non-batch EMCM. The new method could end up labeling more or less than the other methods.

Therefore, the preferred comparison was using the percent of training data compared to the total data set size. All methods started with 10% labeled data. The batch methods calculated ten iterations, where each iteration used 3% additional labeled data, finishing with a total of 40% labeled data. The comparison graphs showed percent labeled data versus RMSE and MAE to indicate what each method looked like starting from 10% to 40% labeled data.

This comparison was not yet completed, as the adaptive batch could have taken more (or fewer) iterations to reach 40% labeled data. The final comparison that was used calculated the number of iterations necessary to reach 40% labeled data. It was not desirable to have an adaptive batch default to 1 item per batch, nor was it useful if the adaptive batch includes all training data in the first iteration. The most desirable solution was one in which the number of iterations was reduced, and the new solution had smaller RMSE and MAE compared to batch at each iteration and each percent of labeled data. It was expected that no solution provided all three items. There were advantages and disadvantages of each, to be discussed in the following section.

C. *Error-Based Experimental Results*

The different stop criteria have been simulated over several runs. The results of the different methods were compared to each other and to several baseline algorithms. Each data set was examined using RMSE and MAE. The methods were then compared by the number of iterations used to reach 40% labeled data. Finally, runtime was examined and all four measures were used to determine the best algorithm.

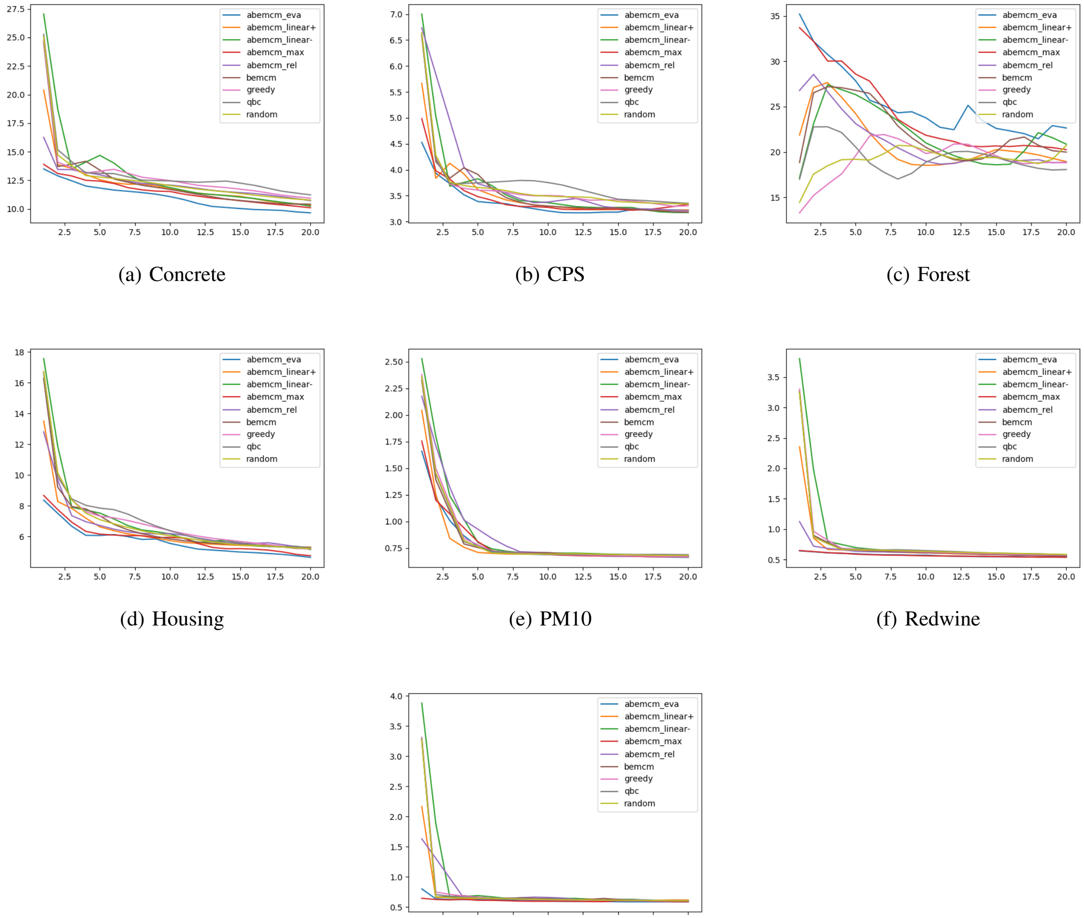
The algorithms used for comparison included: Random, Greedy, QBC, B-EMCM, AB-EMCM Linear +, AB-EMCM Linear -, AB-EMCM Max, AB-EMCM Rel, and AB-EMCM Eva. Random, Greedy and QBC were calculated as batches so their results would be compared to the proposed batch algorithm.

D. *Correlation-Based Experimental Results*

The training method for the machine learning model was statistical based on the selection of training data and the order in which they were used with each iteration. The RMSE was averaged over twenty runs to determine trends in the different approaches. As new training data was introduced to the model, the RMSE calculated on the validation set was sometimes higher than the previous training cycle. What occurred was the new training data pushed the model out of a local minimum into its optimization to reduce error in the training set. This was an expected result.

Figure 1a, shows the different methods as applied to Concrete. The models that stood out below 20% labeled data were AB-EMCM Linear+ and AB-EMCM Max. Both had low RMSE values in the first few iterations. As the data approached 40% labeled data, AB-EMCM Linear+ and AB-EMCM Max maintained the lowest RMSE, but B-EMCM began to make up that difference, offsetting its early higher RMSE. By 40%, AB-EMCM Linear+ had the best RMSE, with B-EMCM and AB-EMCM Linear- closely behind.

Figure 1b, the results for the methods on CPS, AB-EMCM Linear+ and AB-EMCM Max had an early advantage with



(a) Concrete

(b) CPS

(c) Forest

(d) Housing

(e) PM10

(f) Redwine

(g) Whitewine

Fig. 1. Averaged RMSE for each batch method.

a lower RMSE. AB-EMCM Max lost this advantage and became worse in the final steps of the iterations. Several other algorithms had an issue after the second or third iteration in which RMSE increased before decreasing again, including B-EMCM, AB-EMCM Linear-, AB-EMCM Rel, and AB-EMCM EVA. This was caused by the algorithms finding local minimum for the training error function and then new training pushed it out of that local minimum.

For the Forest data, Figure 1c, interesting results happened. The algorithms all converged fairly quickly. AB-EMCM Max had the highest RMSE and reached the lowest by 40% labeled data. For this data set, it was much more difficult to draw a solid conclusion on the results.

Figure 1d and Figure 1f, Housing and Redwine, yielded similar results to Figure 1a, Concrete. Again, AB-EMCM Linear+ and AB-EMCM Max reduced RMSE quickly within the first set of labeled data. AB-EMCM Max stayed well below all algorithms for all labeled data. It was clearly the best for these two data sets.

Figure 1e showed PM10 with slightly different results. AB-EMCM Linear+ reduced RMSE at a much quicker rate than all the algorithms with less data. It outperformed all algorithms at almost every percentage of labeled data. One interesting note is that AB-EMCM Max had been performing very well at low labeled percentages, yet for this data set it was not the lowest RMSE at the beginning nor end of the iteration; only for a short time around 25% labeled. AB-EMCM EVA performed poorly here. The expected vs actual value was too large. It caused very few items to be chosen. The model started to reduce the error for the training set, but the RMSE of the validation set grew large. Basically, AB-EMCM EVA algorithm caused the model to overfit in the beginning iterations until it received enough diverse training candidates to generalize the model better.

Whitewine, Figure 1g, was the only set where AB-EMCM Linear+ did not perform better than the rest. AB-EMCM Max, on the other hand, gained and maintained a lower RMSE for the entire run. While training on the whitewine data set, most of the algorithms had noisy RMSE values; they did not reduce RMSE every iteration. Almost every algorithm either found local minimums or overfit and broke out of the local minimums once new training data was introduced.

It should be noted the B-EMCM performed well in all of these runs. B-EMCM was regularly the third or fourth best algorithm related to RMSE on the data sets. For most of the data sets B-EMCM was slower to converge. It had a higher RMSE on the first five iterations, but after the fifth, it started to yield the same as AB-EMCM Max and the other algorithms.

One pattern that can be seen in all of these is that many of the algorithms converged to the same result by the 40% labeled data. As the pool of candidates reduces, the batch and adaptive batch algorithms tend to select the same candidates. Therefore, as the active learning model received more data to train, the algorithms' RMSE and MAE tended toward each other. This was expected in the absolute case. For 40%, all available data has been labeled. The reasons why all of the algorithms do not result in the same model or same RMSE is the stochastic nature of the algorithm and getting stuck in the training data error local minimum.

AB-EMCM Linear+ and AB-EMCM Linear- performed as expected. These methods started with a lower batch count than the static batch methods and increased batch size linearly with each iteration; or started higher and decreased batch size. These methods both iterated the same amount of times as the static batch methods and covered the same amount of labeled data. Intuitively, starting with a smaller batch and increasing would work better than starting a larger batch and decreasing. The model changed more drastically in the first iterations and its RMSE was much higher. Therefore, the error, or the ability for the model to predict, was poor. Also, the assumptions made in B-EMCM included a rounding error in Equation 3 that grew as the batch increased in size. As training samples were added, the parameters of the model changed more dramatically on the first iterations than it did on the last iterations.

AB-EMCM Max performed well on all data sets. Its selection followed the pattern of AB-EMCM Linear+; starting with smaller sets of batches and increasing the batch size over the iteration. Where it deviated from AB-EMCM Linear+ was when AB-EMCM Max was reactive to the data and prediction itself. AB-EMCM Linear+ increased batch sizes at a steady rate regardless of the data. AB-EMCM Max increased only as the difference of the expected y value and actual y values of the model decreased. AB-EMCM Max, for some runs, increased batch size and then decreased batch size if too many of the candidates caused excessively new or unlearned information to be introduced. AB-EMCM Max outperformed in the first iterations because it took smaller batch sizes. It continued to outperform because the model learned more without overfitting each step of the iteration.

AB-EMCM EVA did not perform well in any of the runs. It took too many small steps, which proved useful in the first few iterations, yet caused too many similar examples to be chosen as candidates. This resulted in the model getting many examples of things similar to what it already learned, so it learned at a slower rate.

AB-EMCM Rel performed very similarly to B-EMCM relative to RMSE. The cutoff of ten iterations was used to indicate when the error changed by one order of magnitude and to end that iteration. This made for similarly sized items to be in the same batch. For some of the data sets, this was an advantage for the first few iterations, but after that, this technique became worse. It caused the batches to become too large. To increase the effectiveness of this algorithm related to RMSE, the cutoff had to decay over the iterations. This is left for further research.

It appears from this first analysis that AB-EMCM Max was reliably the best method to reduce RMSE in the first few iterations over the other algorithms; it maintained the RMSE while still reducing RMSE as new candidates were introduced. If the algorithm stopped prior to the 10th iteration, it was considered the best in relative to almost all of the data sets. It typically had the lowest RMSE at each iteration. It also asymptotically decreased for most data sets which showed the stability and ability to not calculate local minimums as it learned the data set. That was resolved by the reactive varying of batch sizes.

E. Comparison of MAE

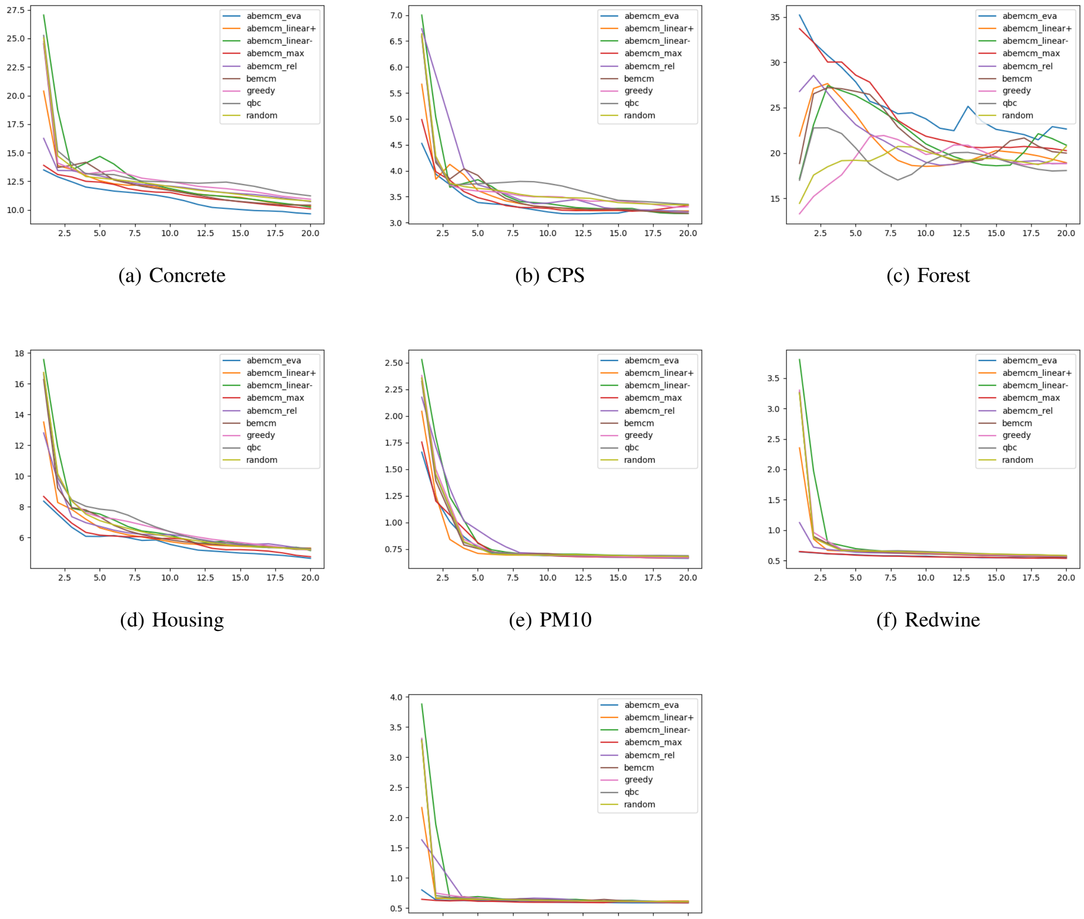
For this analysis and in the previous work, RMSE and MAE are two important metrics to review. While they both provide similar information on the error of the model on the validation set, they have differences that are important to understand. As a comparison it is similar to the way that mean and median are two different statistical values that contain some similar information individually, but together they represent a more complete picture. Since RMSE squares values, large individual error values skew the RMSE. Therefore RMSE is ideal when few large outlier errors are present. RMSE penalizes on the variance of errors which can exaggerate the error. As demonstrated in Chai [35], even if half of the data has no error, RMSE can produce the same result as data with all errors, just with a low error variance.

Figure 2 shows all of the same algorithms and data sets as Figure 1, but using MAE to analyze the results. The results are similar to those of Figure 1 . The AB-EMCM Max algorithm reduces RMSE in the first percent of labeled data, and then continues to keep RMSE lower than the other algorithms. AB-EMCM Linear+ performs at a similar level to AB-EMCM at times. B-EMCM performs at an average compared to the other algorithms employed. AB-EMCM EVA performs the worst, as it did with RMSE.

There are some noticeable differences between RMSE and MAE as it relates to these algorithms. In several of the data sets, including most notably Concrete Figure 1a and Figure 2a, the algorithms AB-EMCM EVA and AB-EMCM Linear- had iterations that increased the RMSE. These errors were exaggerated more in the MAE results. This indicates that the variance of errors was small, but the actual error was large for this model. QBC for Figure 1b and 2b, CPS, also showed a large RMSE regression when analyzing using MAE. There are just a few cases where the results differed. AB-EMCM Max showed to be the best algorithm for most data sets when compared to MAE.

F. Iterations Comparison

The new metric that was introduced in this research was the difference between iterations and amount of labeled data. The batch jobs began with 10% labeled data, added 3% for 10 iterations. For an adaptive batch, it was not reasonable



(a) Concrete

(b) CPS

(c) Forest

(d) Housing

(e) PM10

(f) Redwine

(g) Whitewine

Fig. 2. Averaged MAE for each batch method.

to analyze only ten iterations. The 10th iteration could have included all possible labeled data, or none at all. Therefore, for each method, it was important to examine both the number of iterations and amount of data per iteration. Algorithms that optimized batch sizing were of most interest. Defaulting to include most of the training data in the first batch or one element per batch was undesirable. Although interesting to note, it did not fit the practical goals of developing an algorithm useful for annotation that is nontrivial.

Table 3 shows the average number of iterations taken to reach 40% labeled data.

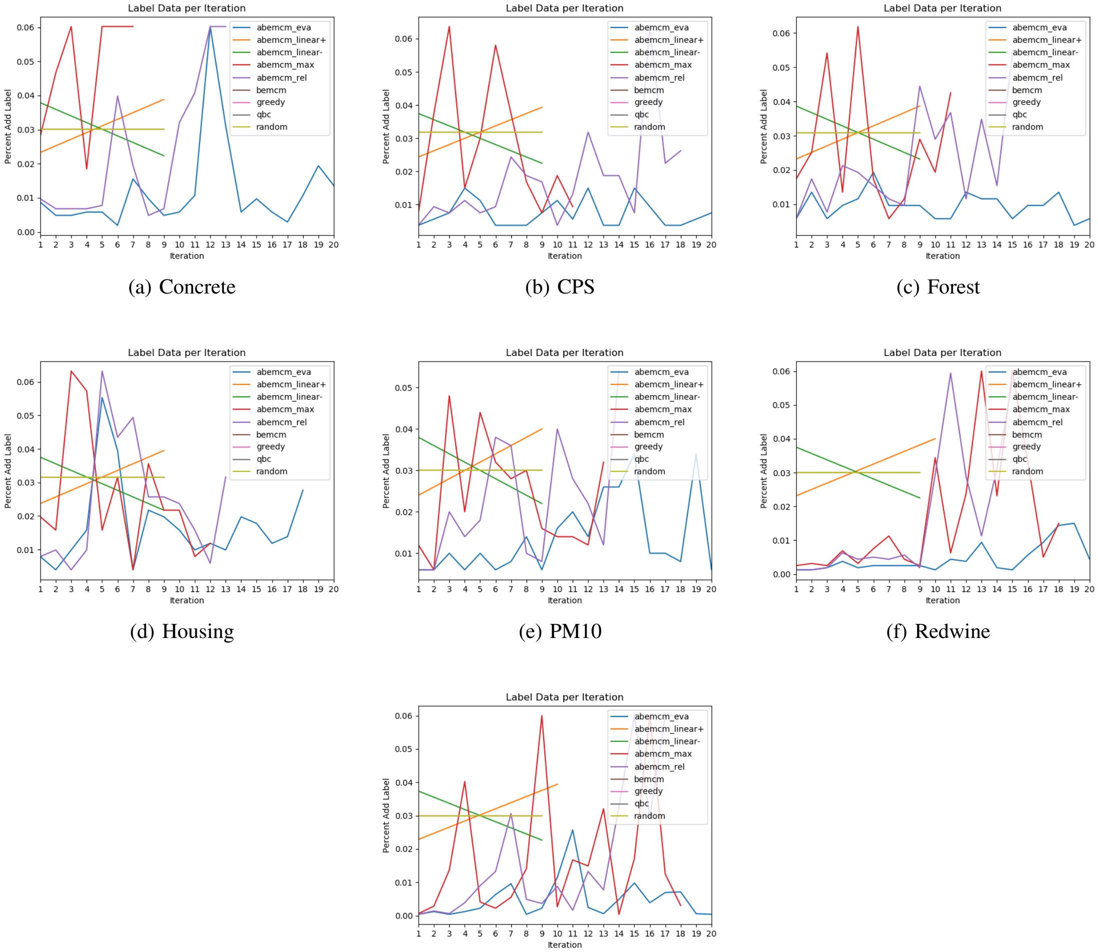
The static batch methods all took ten iterations to go from 10% labeled data to 40% labeled data, using 3% each iteration. AB-EMCM Linear+ and AB-EMCM Linear- were specifically designed as ten iterations, yet having varying batch size. AB-EMCM EVA took approximately two to six times the number of iterations as the static methods. This was accomplished by setting the threshold of the result of Equation 17 to control the acceptable error. For this research, AB-EMCM Max and AB-EMCM Rel thresholds were chosen by running tests for the algorithm and determined which yielded the best end of run RMSE, without exceeding 50 iterations. AB-EMCM Max and AB-EMCM Rel were less than two times the number of iterations as the static methods. An interesting note was that AB-EMCM Max took less iterations for Concrete, and in the RMSE and MAE graphs, outperformed all algorithms except AB-EMCM Linear+. AB-EMCM EVA had the highest number of iterations. Also note as larger data sets are introduced, like Whitewine, the number of iterations grew more quickly than linear. The AB-EMCM EVA algorithm had difficulty scaling to large problems.

Figure 3 a-g shows each algorithm and compares the iteration step with the number of candidates included in that iteration. Random, QBC, Greedy, and B-EMCM all overlapped, taking ten iterations, each step using 3% of the candidates to label, although not every graph shows exactly 3% for these static methods. The data size of each data set varied, and in order to take 3% of an integer, for some of the data sets, it required rounding up to the next integer. Therefore, these graphs show the actual percent of candidates labeled with each iteration.

AB-EMCM Linear+ shows a constant increase, adding about 2.4% labeled data at the start, and ending with adding 3.8% labeled data. AB-EMCM Linear- shows a similar in-

	Random	Greedy	QBC	B-EMCM	Linear+	Linear-	AB-EMCM	Rel	EVA
							Max		
Concrete	10	10	10	10	10	10	8	14	24
CPS	10	10	10	10	10	10	12	19	37
Forest	10	10	10	10	10	10	12	16	30
Housing	10	10	10	10	10	10	13	14	19
PM10	10	10	10	10	10	10	14	14	23
Redwine	10	10	10	10	10	10	19	17	33
Whitewine	10	10	10	10	10	10	19	19	37

TABLE II
ITERATION COUNT PER ALGORITHM AND DATA SET WHEN 40% DATA IS LABELED



(a) Concrete

(b) CPS

(c) Forest

(d) Housing

(e) PM10

(f) Redwine

(g) Whitewine

Fig. 3. Average number of items in each batch.

crease, but started by adding 3.8% labeled data and decreased until adding 2.4% labeled data.

The reactive algorithms are when these graphs get more interesting. AB-EMCM EVA started selecting below 3%, but immediately increased to 6% with each step. This algorithm was not selective and tried to include most of the training data in each iteration. With too large of an addition to the training set, the EMCM algorithm lost its validity as related to the estimated parameter in Equation 2. AB-EMCM Max, in every case, started with a small batch size and almost immediately increased to take the biggest batch sizes possible without negatively affecting RMSE. It adjusted with each step, increasing and decreasing the batch size. This was very different from AB-EMCM Linear+, which was always increasing. Finally AB-EMCM Rel started below 3% in all cases and stayed low for the entire iteration, causing this algorithm to take twice as many iterations as the static to reach 40% labeled data.

In Figure 3, the AB-EMCM Max significantly outperformed the other algorithms in Housing (d), Redwine (f), and Whitewine (g). It performed similarly to the second and third best algorithm for Concrete (a), CPS (b), and PM10 (e). Batch size was examined across all iterations in order to seek a pattern for when the algorithm performed well and when it performed poorly. For Concrete and CPS, AB-EMCM Max started low and oscillated between adding about 2% to 6%. Concrete settled at adding 6% for the last few iterations, but CPS added under 2% for the last few iterations. For PM10, the batch sizes were not quite as large, yet every iteration jumped between 2.5% and 5%. The last few iterations added 1.5% until jumping to 3% for the last iteration.

In contrast, the data sets on which the model performed well tended to stay around 3% for several iterations or more before changing. Housing started at 2%, jumped to 6% and then settled below 3.3% for the remaining iterations. Redwine started with very few candidates per iteration and grew as data was introduced. These large fluctuations at the final iterations indicated new data or outliers to that training were introduced, which changed the model significantly. The algorithm was aggressive when adding new data and then conservative when the model changed too much. Whitewine showed a progression of conservative batch sizes and then large batch sizes when the errors were low enough to introduce more data. The data sets for AB-EMCM Max best results were when it gradually changed batch sizes.

Another important aspect is that most of the adaptive algorithms took over ten iterations; they took smaller steps which can increase the effectiveness of the algorithm. However, the entire advantage was not due to small steps or more iterations. Updating B-EMCM to twenty iterations with 1.5% added each time increased RMSE effectiveness slightly. At the extremes, it has been demonstrated that a single element batch resulted in the best RMSE. Mainly selecting more items to join the batch resulted in error accumulation. The single element, or non-batch EMCM algorithm, only had the single error introduced by the estimation in Equation 2. The batch and adaptive batch algorithms had accumulated errors that were created by estimating the model result based on each candidate added to the batch.

G. Time Performance

Each of the different runs were timed to compare relative speed. The calculations were completed on a Windows Surface with 1.9GHz processor and 8GB of RAM. These numbers were not used to determine absolute runtime as better CPUs can reduce runtime, yet the purpose is to understand the relative runtime between the different algorithms. Table III shows the runtimes of the different algorithms amongst the data sets. They were compared to each other and analyzed to determine how the algorithm scaled up with the varying size of the data in each of the data sets.

Table III shows that Random was by far the fastest and simplest to calculate. Greedy, B-EMCM, AB-EMCM Linear+ and AB-EMCM Linear-, which are all fairly lightweight algorithms, are all the same order of magnitude and took about ten times longer than Random. Greedy required a minimal amount of calculation for each iteration. B-EMCM required just one equation per candidate. AB-EMCM Linear+ and AB-EMCM Linear- are an extension of B-EMCM with a changing batch size that required very little computation to recalculate. AB-EMCM Max and AB-EMCM Rel are also extensions of B-EMCM with more complicated calculations than AB-EMCM Linear+, therefore, the runtime was slightly longer still. QBC had one of the longest as it required model training of the committee models. AB-EMCM EVA took the longest, yet did not have significantly more calculations than the other AB-EMCM introduced. AB-EMCM generally took smaller steps which caused more iterations. Each iteration had some variable time calculations based on additional candidates and some fixed time calculations based on the current training set. More iterations lead to more model training from the training set. From this, Random was the best by overall time, B-EMCM was the best of the batch solutions. If restricted to just the adaptive batch solutions, AB-EMCM Linear+ and AB-EMCM Linear- have the best runtime.

Another aspect that was examined was the runtimes and how they relate to the size of each data set. This explored the scalability of each algorithm. Table I shows the size of the data sets in number of examples and number of features. Table IV shows the relative runtime of each algorithm and data set relative to PM10, the smallest data set. Therefore, the ratio in QBC-Concrete was the runtime of QBC-Concrete divided by the runtime of QBC-PM10. Table I and Table IV were examined together comparing how the algorithms scaled with data size.

CPS, Forest, Housing, and PM10 have similar data size and their runtimes across all algorithms relative to PM10 are nearly 1.0. This shows the algorithms all start from a similar base runtime. When that is compared to AB-EMCM Rel and AB-EMCM EVA, the runtimes across these data sets are close to 2.0. These algorithms produce a higher baseline relative to PM10 and are more sensitive to small increases in data size. Where the ratio started to diverge is Concrete. That data was 2.0 times larger than PM10. Several of the relative runtimes are 2.0 times longer, but AB-EMCM Rel, AB-EMCM EVA and QBC showed an exponential growth in runtime. Redwine, whose relative size is 3.2 times larger, had 5.207, 5.465 and 7.559 increase in runtime. All the other algorithms started to show better than linear growth of runtime. Whitewine is the most pronounced. A ten times increase in data size led to a 13.272, 32.019, and 72.735 times increase in runtime. The best AB-EMCM algorithms relative to runtime were Linear+, Linear-, and Max. AB-EMCM Max exhibited better than linear behavior as the data set increased.

The static batch algorithms have equal batch size and equal number of iterations. When moving to an adaptive batch, the algorithm can choose varying batch sizes, which result in a varying number of iterations to reach the 40% labeled data size. The runtime was partially due to complexity of the algorithm and partly due to batch size choice as the B-EMCM algorithm itself has overhead each iteration. More iterations lead to longer runtimes.

VII. Conclusion and Future Work

Active learning is an essential part of practical problems. With active learning, the resources used to label real life examples is not trivial. Selecting the candidates that reduce the estimated error has been shown to be the best for regression problems using the EMCM.

							AB-EMCM		
	Random	Greedy	QBC	B-EMCM	Linear+	Linear-	Max	Rel	EVA
Concrete	0.080	1.130	3.100	0.870	0.970	0.803	4.400	4.500	9.700
CPS	0.030	0.663	1.010	0.470	0.500	0.500	2.050	3.100	4.775
Forest	0.045	0.650	0.903	0.600	0.575	0.498	1.860	1.950	2.585
PM10	0.035	0.795	1.020	0.450	0.590	0.475	1.785	1.162	2.482
Housing	0.043	0.800	1.065	0.540	0.690	0.533	2.900	2.390	4.560
Redwine	0.130	1.995	7.710	2.020	3.655	2.795	5.363	6.050	13.565
Whitewine	0.273	6.518	74.190	6.050	6.527	6.090	12.065	15.4225	79.472

Note: in seconds

TABLE III
AVERAGE RUNTIMES PER ALGORITHM AND PER DATA SET

							AB-EMCM		
	Random	Greedy	QBC	B-EMCM	Linear+	Linear-	Max	Rel	EVA
Concrete	2.286	1.421	3.039	1.933	1.644	1.691	2.465	3.873	3.908
CPS	0.857	0.834	0.990	1.044	0.847	1.053	1.148	2.668	1.924
Forest	1.286	0.818	0.885	1.333	0.975	1.048	1.042	1.678	1.041
PM10	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Housing 1.229	1.006	1.044	1.200	1.169	1.122	1.625	2.057	1.837	
Redwine 3.714	2.509	7.559	4.489	6.195	5.884	3.004	5.207	5.465	
Whitewine 7.800	8.199	72.735	13.444	11.063	12.821	6.759	13.272	32.019	

TABLE IV
RATIO OF RUNTIME OF EACH DATA SET TO PM10

Summarizing the different metrics together, AB-EMCM Linear+ and AB-EMCM Max showed the best results when looking at RMSE and MAE. When comparing the number of iterations, all the static batch algorithms, along with both AB-EMCM Linear+ and Linear- finished in ten iterations. Of the adaptive batch, AB-EMCM Max was closest to ten iterations, and less than ten in the case of Concrete. Finally, in the analysis of runtime, both AB-EMCM Linear+ and Linear- and B-EMCM performed the best; they were the fastest although AB-EMCM Max was not significantly higher in runtime. AB-EMCM Max also showed better than linear growth as the data size grew: this indicates the ability of the algorithm to scale. Considering all these aspects, AB-EMCM Max is the most well rounded algorithm, the most important attribute being its ability to reduce RMSE to its lowest, with the fastest runtimes during the iterations. As well, it had the fewest number of iterations to achieve 40% labeled data. fastest runtime. In direct comparison to B-EMCM, AB-EMCM Max performed better in almost every respect. It is a substantial addition to B-EMCM to adaptively grow and shrink batch to accommodate accumulated error.

The method for max bound stop criteria could be explored further by adding a reduction factor in the accumulated error. Since the error became smaller with each iteration, reducing the acceptable accumulated threshold to consistently increase accuracy in the example selection.

Another item that is open for further research is determining stop criteria in a more systematic manner. Several algorithms, including the best one as shown in this research, AB-EMCM Max, required the determination of a threshold which is relative in value to the error and to predicted values of the model. The best value for the threshold was determined by test runs calculating the best output related to RMSE. For practical problems, it would be better if this value were a percentage, or an absolute value, that could be selected before a run. Learning rate in machine learning is a good example of a value that is set and determines the ability of a model to learn and converge, but traditionally has a value that can be recommended prior to a run.

When analyzing batch sizes per iterations, it was noted that gradual changes in batch size led to better results relative to RMSE of the validation set. For further research it may be advantageous to build damping into the stop criteria equation so that batch sizes do not change so radically from one iteration to the next. When the errors are low and the batch size is increased drastically; this could stop too many candidates being added which can cause the error to increase drastically in proportion to the batch size. Large errors cause the next batch size to be too small and effectively useless.

The research of active learning history was presented leading to the latest work of EMCM and B-EMCM. Using B-EMCM as a starting point, adaptive batch and stop criteria were introduced to increase the effectiveness of B-EMCM. Several different stop criteria were introduced including all AB-EMCM: Linear+, Linear-, Max, Rel, and EVA. Each of these methods explored a different error as candidates are added to the batch. These methods were used on the Statlib and Y data sets, and compared against B-EMCM and several non-batch methods. Analyzing RMSE, MAE, runtime, and number of iterations, resulted in AB-EMCM Max being the best adaptive batch method to use.

REFERENCES

[1] S. Basu, M. Bilenko, and R. J. Mooney, "A probabilistic framework for semi-supervised clustering," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 59–68.

[2] S. Basu, I. Davidson, and K. Wagstaff, *Constrained clustering: Advances in algorithms, theory, and applications*. CRC Press, 2008.

[3] F. Schwenker and E. Trentin, "Pattern classification and clustering: A review of partially supervised learning approaches," *Pattern Recognition Letters*, vol. 37, pp. 4–14, 2014.

[4] B. Settles, "Active learning literature survey," 2009.

[5] J. Friedman, T. Hastie, R. Tibshirani *et al.*, *The elements of statistical learning*. Springer series in statistics New York, 2001, vol. 1, no. 10.

[6] J. H. Friedman, “Greedy function approximation: a gradient boosting machine,” *Annals of statistics*, pp. 1189–1232, 2001.

[7] J. A. Hertz, *Introduction to the theory of neural computation*. CRC Press, 2018.

[8] W. Cai, M. Zhang, and Y. Zhang, “Batch mode active learning for regression with expected model change,” *IEEE transactions on neural networks and learning systems*, vol. 28, no. 7, pp. 1668–1681, 2016.

[9] M. Guillaumin, J. Verbeek, and C. Schmid, “Multimodal semi-supervised learning for image classification,” in *2010 IEEE Computer society conference on computer vision and pattern recognition*. IEEE, 2010, pp. 902–909.

[10] G. Nagy and G. Shelton, “Self-corrective character recognition system,” *IEEE Transactions on Information Theory*, vol. 12, no. 2, pp. 215–222, 1966.

[11] A. Blum and T. Mitchell, “Combining labeled and unlabeled data with co-training,” in *Proceedings of the eleventh annual conference on Computational learning theory*, 1998, pp. 92–100.

[12] A. Verikas, A. Gelzinis, and K. Malmqvist, “Using unlabelled data to train a multilayer perceptron,” *Neural Processing Letters*, vol. 14, no. 3, pp. 179–201, 2001.

[13] O. Chapelle, P. Shivaswamy, S. Vadrevu, K. Weinberger, Y. Zhang, and B. Tseng, “Boosted multi-task learning,” *Machine learning*, vol. 85, no. 1, pp. 149–173, 2011.

[14] K. Nigam and R. Ghani, “Analyzing the effectiveness and applicability of co-training,” in *Proceedings of the ninth international conference on Information and knowledge management*, 2000, pp. 86–93.

[15] I. Žliobaitė, A. Bifet, B. Pfahringer, and G. Holmes, “Active learning with drifting streaming data,” *IEEE transactions on neural networks and learning systems*, vol. 25, no. 1, pp. 27–39, 2013.

[16] D. D. Lewis and W. A. Gale, “A sequential algorithm for training text classifiers,” in *SIGIR’94*. Springer, 1994, pp. 3–12.

[17] B. Settles and M. Craven, “An analysis of active learning strategies for sequence labeling tasks,” in *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, 2008, pp. 1070–1079.

[18] Y. Freund, H. S. Seung, E. Shamir, and N. Tishby, “Selective sampling using the query by committee algorithm,” *Machine learning*, vol. 28, no. 2, pp. 133–168, 1997.

[19] P. Campigotto, A. Passerini, and R. Battiti, “Active learning of pareto fronts,” *IEEE transactions on neural networks and learning systems*, vol. 25, no. 3, pp. 506–519, 2013.

[20] E. J. de Fortuny and D. Martens, “Active learning-based pedagogical rule extraction,” *IEEE transactions on neural networks and learning systems*, vol. 26, no. 11, pp. 2664–2677, 2015.

[21] H. Yu and S. Kim, “Passive sampling for regression,” in *2010 IEEE International Conference on Data Mining*. IEEE, 2010, pp. 1151–1156.

[22] W. Cai, Y. Zhang, and J. Zhou, “Maximizing expected model change for active learning in regression,” in *2013 IEEE 13th International Conference on Data Mining*. IEEE, 2013, pp. 51–60.

[23] D. Wu, “Pool-based sequential active learning for regression,” *IEEE transactions on neural networks and learning systems*, vol. 30, no. 5, pp. 1348–1359, 2018.

[24] Z. Liu and D. Wu, “Integrating informativeness, representativeness and diversity in pool-based sequential active learning for regression,” in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–7.

[25] Z. Liu, X. Jiang, H. Luo, W. Fang, J. Liu, and D. Wu, “Pool-based unsupervised active learning for regression using iterative representativeness-diversity maximization (irdm),” *Pattern Recognition Letters*, vol. 142, pp. 11–19, 2021.

[26] S. H. Park and S. B. Kim, “Robust expected model change for active learning in regression,” *Applied Intelligence*, vol. 50, no. 2, pp. 296–313, 2020.

[27] D. Wu, C.-T. Lin, and J. Huang, “Active learning for regression using greedy sampling,” *Information Sciences*, vol. 474, pp. 90–105, 2019.

[28] Y. Zhang, W. Cai, W. Wang, and Y. Zhang, “Stopping criterion for active learning with model stability,” *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 9, no. 2, pp. 1–26, 2017.

[29] N. Fazakis, V. G. Kanas, C. K. Aridas, S. Karlos, and S. Kotsiantis, “Combination of active learning and semi-supervised learning under a self-training scheme,” *Entropy*, vol. 21, no. 10, p. 988, 2019.

[30] N. Fazakis, G. Kostopoulos, S. Karlos, S. Kotsiantis, and K. Sgarbas, “An active learning ensemble method for regression tasks,” *Intelligent Data Analysis*, vol. 24, no. 3, pp. 607–623, 2020.

[31] P. B. Bindal, D. Singhal, A. Subramanyam, V. Kumar *et al.*, “Onestop-tuner: An end to end architecture for jvm tuning of spark applications,” *arXiv preprint arXiv:2009.06374*, 2020.

[32] M. Haghighatlari, G. Vishwakarma, D. Altarawy, R. Subramanian, B. U. Kota, A. Sonpal, S. Setlur, and J. Hachmann, “Chemml: A machine learning and informatics program package for the analysis, mining, and modeling of chemical and materials data,” *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 10, no. 4, p. e1458, 2020.

[33] Z. Puha, M. Kaptein, and A. Lemmens, “Batch mode active learning for individual treatment effect estimation,” in *2020 International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2020, pp. 859–866.

[34] S. Chakraborty, V. Balasubramanian, and S. Panchanathan, “Adaptive batch mode active learning,” *IEEE transactions on neural networks and learning systems*, vol. 26, no. 8, pp. 1747–1760, 2014.

[35] T. Chai and R. R. Draxler, “Root mean square error (rmse) or mean absolute error (mae),” *Geoscientific Model Development Discussions*, vol. 7, no. 1, pp. 1525–1534, 2014.

[36] G. Dong and V. Taslimitehrani, “Pattern-aided regression modeling and prediction model analysis,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 9, pp. 2452–2465, 2015.