Online Feature Selection for Regression

Fransisca Susan

Abstract

We consider sparse regression in an online setting for problems where high-dimensional data arrive sequentially. We compare two recent methods: truncated stochastic gradient descent (TSGD) and online least square with thresholding (OLST), and propose a hybrid method, OLST-then-TSGD. We implement these algorithms and compare their performances with each other and two offline sparse regression algorithms on synthetic and Wikiface images datasets. TSGD runs the fastest but gives the worst error and true support recovery, while OLST runs the slowest and requires much computational space but gives the most accurate prediction and true support recovery on both datasets. OLST-then-TSGD maintains a small regret using less computational time and space.

1 Introduction

Motivation for online learning: An offline regression problem considers a sequence of i.i.d. observations from an unknown distribution: $\{(\boldsymbol{x}_t, y_t)\}_{t=1,\dots,n}$, $\boldsymbol{x}_t \in \mathbb{R}^p, y_t \in \mathbb{R}, \forall 1 \leq i \leq n$, and aims to find $\boldsymbol{\beta}$ that minimizes loss, $L(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=1}^{n} l(\boldsymbol{\beta}; (\boldsymbol{x}_i, y_i))$, where $l(\cdot; (\boldsymbol{x}_i, y_i)) : \mathbb{R}^p \to \mathbb{R}$ is a loss function. When the data comes sequentially and is not available by the time of prediction, or when the size of the datasets are so large (such as in bioinformatics and computer vision), online learning comes in play by constructing models sequentially, one data point at a time. In this case, the algorithm makes prediction $\boldsymbol{\beta}_t$ of the weight at period t based on past history while attempting to maximize the accuracy of the sequence of predictions. A comprehensive survey of the online learning and optimization literature can be found in [2] and [3].

Motivation for feature selection: Feature selection (FS) is the process of selecting a subset of relevant features and constructing a model using only the relevant features. In nature, a lot of high-dimensional data has a low-dimensional structure. Since it is usually expensive to acquire a full set of high dimensional data, the growing availability of such high-dimensional datasets motivates lots of work in variable ranking and feature selection. Furthermore, doing feature selection - removing irrelevant and redundant features - can improve a model's performance by increasing generalization capabilities, alleviating the curse of dimensionality, speeding up the learning process, and enhancing interpretability.

1.1 Problem statement

In this paper, we consider feature selection for online regression. Let $\{(\boldsymbol{x}_t, y_t)\}_{t=1,2,\dots,n}$ be a sequence of i.i.d. observations where each $\boldsymbol{x}_t \in \mathbb{R}^p$ and $y_t \in \mathbb{R}$. We assume that p is a large number and for computational efficiency we need to select a sparse weight for regression at each period. Specifically, at period t, the algorithm chooses a weight $\boldsymbol{\beta}_t \in \mathbb{R}^p$, which will be used to compute the prediction on $\boldsymbol{x}_t$ using $y_{t,pred} = \boldsymbol{x}_t^T \boldsymbol{\beta}_t$; however, we restraint the $\boldsymbol{\beta}_t$ to have at most k non-zero elements: $||\boldsymbol{\beta}_t||_0 \leq k$, meaning that only k features of $\boldsymbol{x}_t$ will be used for prediction. We want to find $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \dots, \boldsymbol{\beta}_n$ that maximize the accuracy of the sequence of predictions and minimize the regret,

$$R_n = \frac{1}{n}\sum_{i=1}^{n} l(\boldsymbol{\beta}_i; (\boldsymbol{x}_i, y_i)) - \min_{\boldsymbol{\beta}} \frac{1}{n}\sum_{i=1}^{n} l(\boldsymbol{\beta}; (\boldsymbol{x}_i, y_i)), \tag{1}$$

where $l(\cdot; (\boldsymbol{x}_i, y_i)) : \mathbb{R}^p \to \mathbb{R}$ is a loss function. This regret measures what is lost compared to the offline optimization algorithm loss, which we get by looking at all the data. The speed of convergence

of online algorithms is usually defined as how fast the regret R_n goes to 0. Throughout this paper, we use p to denote the dimension of our data, k to denote the number of important features (at most k features of each $\boldsymbol{x}_t$ can be used for prediction), and n to denote the total number of observations.

The structure of this paper is as follows. In section 2 and 3, we present two recent online feature selection algorithms for regression, together with a hybrid algorithm in section 3.2. In section 4, we describe some challenges encountered in implementation and our solution to implement the methods. In section 5, we compare the performance of the three online algorithms with each other and two offline feature selection for regression algorithms: OMP and Lasso, on synthetic and real datasets. We conclude in section 6.

2 Algorithm 1: online least squares with thresholding (OLST)

2.1 Running averages

We first define our running averages. Suppose that $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_t, y_t)$ are the data we have seen so far by the end of round t, let

$$\boldsymbol{\mu}_{x,t} = \frac{1}{t}\sum_{i=1}^{t}\boldsymbol{x}_i, \quad \mu_{y,t} = \frac{1}{t}\sum_{i=1}^{t} y_i, \quad \boldsymbol{S}_{xx,t} = \frac{1}{t}\sum_{i=1}^{t}\boldsymbol{x}_i\boldsymbol{x}_i^T, \quad \boldsymbol{S}_{xy,t} = \frac{1}{t}\sum_{i=1}^{t} y_i\boldsymbol{x}_i \tag{2}$$

be our running averages. These running averages cover all necessary sample information for model estimation, can be updated one example at a time (perfect for online learning), and do not increase in dimension as t increases. We need to normalize our running averages since in feature selection, the scale of a variable should not influence its importance. To simplify our notation, let $\boldsymbol{X} \in \mathbb{R}^{t\times p}$ and $\boldsymbol{y} \in \mathbb{R}^t$ be the data matrix and the label vector respectively. Let $\sigma_{x_j,t}$ be the sample standard deviation of the j-th coordinate, $\sigma_{x_j,t} = \sqrt{(\boldsymbol{S}_{xx,t})_j - (\boldsymbol{\mu}_{x,t})_j^2}$, where $(\boldsymbol{S}_{xx,t})_j$ is the j-th diagonal entry of $\boldsymbol{S}_{xx,t}$. Let $\boldsymbol{\Pi}_t = \text{diag}(\sigma_{x_1,t}, \sigma_{x_2,t}, \ldots, \sigma_{x_p,t})^{-1}$, $\mathbf{1}_t = [1, \ldots, 1]^T \in \mathbb{R}^t$, $\tilde{\boldsymbol{X}}$ be the normalized $\boldsymbol{X}$, and $\tilde{\boldsymbol{y}}$ be the normalized $\boldsymbol{y}$. After standardization, we have $\tilde{\boldsymbol{X}} = (\boldsymbol{X} - \mathbf{1}_t\boldsymbol{\mu}_{x,t}^T)\boldsymbol{\Pi}$ and $\tilde{\boldsymbol{y}} = (\boldsymbol{y} - \mu_{y,t}\mathbf{1}_t)$. Substituting these to the running averages, we get

$$\boldsymbol{S}_{\tilde{x}\tilde{x},t} = \boldsymbol{\Pi}_t(\boldsymbol{S}_{xx,t} - \boldsymbol{\mu}_{x,t}\boldsymbol{\mu}_{x,t}^T)\boldsymbol{\Pi}_t, \quad \boldsymbol{S}_{\tilde{x}\tilde{y},t} = \boldsymbol{\Pi}_t\boldsymbol{S}_{xy,t} - \mu_{y,t}\boldsymbol{\Pi}_t\boldsymbol{\mu}_{x,t}, \tag{3}$$

but for convenience, from this point on we use $\boldsymbol{S}_{xx,t}$ and $\boldsymbol{S}_{xy,t}$ to denote the normalized running averages. We first present a statement that motivates the algorithm.

Notice that the following penalized regression problem: $\min_{\beta} \frac{1}{2n}||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}||^2 + \boldsymbol{P}(\boldsymbol{\beta}; \lambda)$, where $\boldsymbol{\beta}$ is the support vector and $\boldsymbol{P}(\boldsymbol{\beta}; \lambda)$ is a penalty function, is equivalent to the running averages-based online optimization problem: $\min_{\beta} \frac{1}{2}\boldsymbol{\beta}^T\boldsymbol{S}_{xx}\boldsymbol{\beta} - \boldsymbol{\beta}^T\boldsymbol{S}_{xy} + \boldsymbol{P}(\boldsymbol{\beta}; \lambda)$. This statement can be easily proved by rewriting the L2-norm and substituting the running averages. It implies that the offline problem is equivalent to running averages-based optimization.

2.2 Online least squares (OLS)

Ordinary least squares gives the solution for the equation $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{X}^T\boldsymbol{y}$, when $\boldsymbol{X}^T\boldsymbol{X}$ is invertible. If we replace $\boldsymbol{X}^T\boldsymbol{X}$ and $\boldsymbol{X}^T\boldsymbol{y}$ with running averages when $\boldsymbol{X}$ contains t data points, we have $\boldsymbol{S}_{xx,t}\boldsymbol{\beta} = \boldsymbol{S}_{xy,t}$, which is the online least squares; hence, OLS is equivalent to its offline counterpart.

2.3 Online least squares with thresholding (OLST)

The OLST [7] algorithm wants to find $\{\boldsymbol{\beta}_t\}_{t=1,\ldots,T}$, each having at most k non-zero coefficients that minimize the sum of squared errors, i.e. minimizing $\frac{1}{2n}\sum_{t=1}^{n}||y_t - \boldsymbol{x}_t^T\boldsymbol{\beta}_t||^2$. It does so by doing the following three steps: 1) finding the online least square estimate $\hat{\boldsymbol{\beta}}$ from running averages, 2) keeping only k non-zero features with the largest absolute magnitudes $|\beta_j|, j = 1, 2, \ldots, p$, while setting the rests to 0, and 3) fitting another online least

square coefficient on the selected features. We summarize the OLST algorithm as follows:

Algorithm 1: Online Least Squares with Thresholding (OLST)

Input: Streaming data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, sparsity level k
Output: Regression parameters $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, \ldots, \boldsymbol{\beta}_n$ for each round, where $||\boldsymbol{\beta}_t||_0 \leq k$ for $1 \leq t \leq n$.

Initialize $\boldsymbol{\beta}_0 = \mathbf{0}$;
for $t = 1, 2, \ldots, n$ **do**
  Output $\boldsymbol{\beta}_{t-1}$, get $(\boldsymbol{x}_t, y_t)$ as feedback;
  Compute $\boldsymbol{S}_{xx,t}, \boldsymbol{S}_{xy,t}$ incorporating $(\boldsymbol{x}_t, y_t)$;
  Find $\hat{\boldsymbol{\beta}}$ by solving $\boldsymbol{S}_{xx,t}\boldsymbol{\beta} = \boldsymbol{S}_{xy,t}$;
  Keep only k features with the largest $|\hat{\beta}_j|$;
  Set $\boldsymbol{\beta}_t \leftarrow$ OLS estimator on the selected features, solving $\boldsymbol{S}_{xx,t}\boldsymbol{\beta} = \boldsymbol{S}_{xy,t}$, where $\beta_j = 0$ if j is not selected;
end

2.4 Complexity and convergence

The time complexity of OLST is $O(p^2)$ to update the running averages at each period and $O(p^2)$ to compute the OLS estimator using the Sherman-Woodbury formula for rank 1 update of the inverse of $\boldsymbol{S}_{xx}$, which was previously mentioned in lecture. The space complexity of OLST is $O(p^2)$ to store all the running averages $(\boldsymbol{\mu}_{x,t}, \mu_{y,t}, \boldsymbol{S}_{xx,t}, \boldsymbol{S}_{xy,t})$, the inverse of $\boldsymbol{S}_{xx,t}$ for OLS updates, and the $\boldsymbol{\beta}_t$ coefficients. The OLST algorithm has $O(1/n)$ and $O(\log^2(n)/n)$ convergence for the coefficients and regret respectively, which was proven in great detail in theorem 4 and 11 in [7].

3 Algorithm 2: truncated stochastic gradient descent (TSGD)

The TSGD [1] algorithm adds a truncation step to the SGD algorithm on the streaming data to find weights $\boldsymbol{\beta}_t$ at each round having at most k non-zero coefficients, i.e., the updated weight after getting the t-th data point is

$$\boldsymbol{\beta}_t = \texttt{Truncate}(\boldsymbol{\beta}_{t-1} + \eta(y_t - \boldsymbol{x}_t^T\boldsymbol{\beta}_{t-1})\boldsymbol{x}_t, k),$$

where $\texttt{Truncate}(\boldsymbol{\beta}, k)$ keeps the k largest $|\beta_j|$ and sets the rests to 0.

3.1 Complexity and convergence

The time and space complexities of TSGD are both $O(p)$, since it takes $O(p)$ time to compute the updated weight and truncate it at each round, and the algorithm only needs to keep the p-dimensional weights from the previous and current rounds.

The true support convergence of TSGD relies heavily on a good initialization, let $\boldsymbol{\beta}_{init}$ be the initialized weight with support set $\mathcal{A}_{init} = \{j : (\boldsymbol{\beta}_{init})_j \neq 0\}$. Suppose that $\boldsymbol{\beta}^*$ is the true support of the underlying data, i.e. $y = \boldsymbol{x}^T\boldsymbol{\beta}^* + \epsilon$, and S is the support set of $\boldsymbol{\beta}^*$, $S = \{j : \boldsymbol{\beta}_j^* \neq 0\}$. Moreover, let α_X be the smallest eigenvalue of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]$. Then, under some assumptions (look at [1]) that includes a good initialization $S \subseteq \mathcal{A}_{init}$, Sun et al. [7] proves that setting the learning rate $\eta = \alpha_X^{-1} \log n/n$ for n streaming examples implies an $O(\log n/n)$ convergence of the weight, $\mathbb{E}\left[||\boldsymbol{\beta}_n - \boldsymbol{\beta}^*||^2 | S \subseteq \mathcal{A}_{init}\right] \lesssim \frac{\sigma^2}{\alpha_X} \cdot \frac{k \log n}{n}$.

3.2 Algorithm 3: OLST-then-TSGD - giving a good initialization weight for TSGD

Since a good initialization is crucial for TSGD to converge to the true support, to take advantage of the convergence and regret guarantee from OLST while maintaining the fast running time of TSGD, a natural extension is a hybrid between the two methods: finding a good initialization by running OLST in the first C rounds then switching to TSGD for a faster compute time and less space requirement; here, C is our switch parameter.

This method enjoys both a good guarantee and a good time complexity. The time and memory complexity is amortized $O\left(\frac{C}{n}p^2 + \frac{n-C}{n}p\right)$, which goes to $O(p)$ as n goes to infinity for a fixed C. Furthermore, picking a large enough C will imply the true support convergence from OLST (with

high probability) while initializing TSGD with a weight whose support is close to the true support implies the convergence guarantee from TSGD (with high probability), hence OLST-then-TSGD enjoys a good guarantee while keeping the time and memory complexity low in average.

Another approach used to get a good initialization for TSGD is by running an offline algorithm on a mini-batch data first, assuming that a mini-batch data is readily available before the next set of data comes in a streaming fashion. This situation appears in some scientific studies, such as in genomics, where some preliminary results are readily available before running further studies on streaming data.

4 Implementation and challenges

We compare the performance of three online algorithms: TSGD, OLST, OLST-then-TSGD, and two offline algorithms: orthogonal matching pursuit (OMP) [9] and Lasso [8] on synthetic and real datasets.

OMP takes the number of the non-zero coefficients, k, as an input to the algorithm, while Lasso takes α, the multiplier of the regularization factor as its input but not k, hence we tune the values of α to get our desired k. We use Scikit learn [4] library for OMP and Lasso as our benchmark, especially in computing the regret of our online algorithms.

For the online algorithms, the only hyperparameters we need to tune are the learning rate η for TSGD and the TSGD part of the hybrid algorithm, and the switch parameter C for OLST-then-TSGD. We code all the online approaches ourselves with some tweaks. First, when implementing TSGD, we assume that we do not know the length of the horizon and use the doubling trick to determine the learning rate, meaning that we start with some $\eta = \log(a)/a$ for some $a > 0$, then at round $t_1 = \min(t : t > a)$, we update a to be $2t_1$ and η to be $\log(2t_1)/(2t_1)$, and so on. Second, when implementing OLST, we find that the weight vectors always blow up (having huge norms) on the first OLS step at each round, resulting in bad performance; we fix the issue by replacing OLS on line 5 of the algorithm with a ridge regression estimator. Finally, we compare the performance of several C values by running the OLST-then-TSGD algorithm on our generated synthetic dataset 100 times using various values of C, and looking at the average RMSE.

For the Wikiface dataset, since we do not know the sparsity level (k) of the true parameter, we pick the optimal k using 5-fold stratified cross-validation, where we randomly split the dataset into five balanced subsets and pick the highest R^2 on average.

5 Experiments, results, and discussion

In this section, we evaluate and compare the performance of five algorithms: TSGD, OLST, OLST-then-TSGD, OMP, and Lasso algorithm. First, we compare their performance on prediction and feature selection (picking the true support) on synthetic data. Then, we compare their regret terms on the synthetic data. Finally, we demonstrate their performance on a real regression dataset, Wikiface, by comparing their average R^2 values. All experiments are run with 2.5 GHz Intel Core i7 processor and 16GB of RAM.

5.1 Metrics

To evaluate the performance of the algorithms on synthetic datasets, we look at four metrics: 1) root mean square error (RMSE), defined as $RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$, where $\hat{y}_i$ and y_i denote the predicted and observed values respectively, 2) time, and 3) true predictor detection rate (DR), which is defined as $DR = \frac{1}{n}\sum_{t=1}^{n}\frac{|S_{\boldsymbol{\beta}_t} \cap S_{\boldsymbol{\beta}^*}|}{|S_{\boldsymbol{\beta}^*}|}$. For the Wikiface dataset, since RMSE is heavily influenced by the y distribution and we do not know the true support of the real dataset, the metrics that we use are time and Rsquared value, defined as $R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$, where $\bar{y}$ denotes the mean of the observed data.

5.2 Synthetic datasets experiments

We generate a data matrix $\boldsymbol{X} \in \mathbb{R}_{n \times p}$, whose rows are independent realizations of a p-dimensional centered distribution, $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^T$, where only k dimensions are relevant for regression. We generate X to be uniformly correlated predictors: $\boldsymbol{x}_i \sim \mathcal{N}(0, \Sigma)$, where $\Sigma_{i,i} = 1, \forall i$, and $\Sigma_{i,j} = \rho, \forall i \neq j$ for $\rho = 0.5$. Given $\boldsymbol{X}$, we generate the dependent variable $\boldsymbol{y}$ as follows

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}^* + \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}_n),$$

where $\boldsymbol{\beta}^*$ is a p-dimensional predictor with k non-zero values. We consider the following predictors: 1) **weak signal**: $\boldsymbol{\beta}^* = \beta \cdot (\mathbf{1}_k, \mathbf{0}_{p-k})^T$, where $\beta = 0.01$, 2) **strong signal**: $\boldsymbol{\beta}^* = (\mathbf{1}_k, \mathbf{0}_{p-k})^T$, and 3) **varied signal**: $\boldsymbol{\beta}^* = \left(\frac{1}{k}, \frac{2}{k}, \ldots, \frac{k}{k}, \mathbf{0}_{p-k}\right)^T$. The simulation is based on the following parameters: $p = 1000, k \in \{10, 100\}$, and chosen values of n in the interval $[10^3, 10^6]$; we run the simulation 100 times for each set of parameters and report the averages in the next section.

5.2.1 Synthetic datasets results

We summarize the results in Table 2 in the appendix. We replicate the experiments 100 times and present the average results. We discuss a few important points.

Time and memory complexity: Compared to the offline algorithms, OMP and Lasso, all the online algorithms: TSGD, OLST, and OLST-then-TSGD enjoy lower memory complexity. When the dataset is large, ex: $n = 10^6$, OMP and Lasso surpass the memory limit, while the online methods run smoothly; recall that the memory complexity of our online algorithms is at most $O(p^2)$, which is smaller than $O(np)$ (the lower bound of the memory complexity of the offline algorithms) when $n >> p$. Furthermore, TSGD runs the fastest, followed by OLST-then-TSGD (depending on the C value as expected), then OLST; this is consistent with the theoretical time and space complexity of each algorithm. Interestingly, the running time of the hybrid algorithm is closer to TSGD's running time than OLST, especially for big n.

True support recovery: OLST and OMP are the only algorithms that perform consistently well in feature recovery. When the signal is weak, OLST needs a larger sample size n to recover the true support, yet it beats Lasso in performance as n gets bigger. In general, this is true for algorithms with a good detection rate (OLST, Lasso, and OMP), a strong signal is easier to recover than a weak signal, while our varied signal is in between, which can be seen from the RMSE and true support detection rate values. For TSGD, however, the result is inconclusive; we hypothesize that this is because of TSGD's dependence on a good initialization. For OLST-then-TSGD, the feature detection rate is higher when OLST is run for a longer time. i.e., $C = 500$ than when $C = 50$, this is reasonable because running OLST for a longer time gives us a better initialization vector for the TSGD step. However, we suspect that sometimes $C = 500$ is not enough to get a vector whose support is close to the true support; this can be seen for the following case: ($p = 1000, k = 100$, weak signal), where the detection rate for $C = 500$ is still relatively low.

Prediction: Most methods do well in prediction except for OMP, Lasso, TSGD, and the hybrid algorithm with $C = 50$, when the signal is strong. In contrast, OLST and the hybrid algorithm with $C = 500$ perform well regardless of the strength of the signal, which can be seen from low RMSE values.

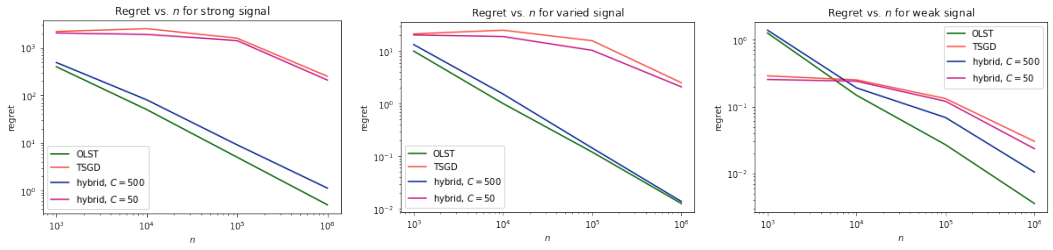

Figure 1: Regret vs. n for TSGD, OLST, and OLST-then-TSGD algorithm from our experiments on synthetic dataset, averaged over 100 runs. Left: strong signal, middle: varied signal, right: weak signal, see section 5.2 for details.

Regret analysis: In figure 1, we show the regret curve of our online algorithms using OMP as the offline loss benchmark. The three subplots represent strong signal (left), varied signal (middle), and weak signal (right). We get these regrets for the following set of parameters: $n \in \{10^3, 10^4, 10^5, 10^6\}, p = 1000$, and $k = 10$. The slopes of the curve show the convergence rates. We can see that the convergence rate of TSGD, OLST, and OLST-then-TSGD all are close to $O(1/n)$, although for TSGD and OLST-then-TSGD with $C = 50$, the curves start with a plateau before going down. Moreover, we can see that for weak signal, the regret of OLST-then-TSGD with $C = 500$ is far from the regret of OLST; this is because when the signal is weak, OLST takes longer

to recover the true support, hence when we switch from OLST to TSGD at $t = 500$, the weight at that point might still be far from the true support.

5.3 Real dataset experiments

We use Wikiface [5] as our real dataset. Wikiface contains 53040 face images of celebrities from Wikipedia and their age when the pictures were taken. We compare our algorithms on age prediction task from images. Each image is cropped, resized to 224 x 224 pixels, fed to a pre-trained CNN model, VGG-16 [6], and transformed into a 4096-dimensional feature vector, which is the input to our regression problem.

5.3.1 Wikiface results

Metric	OMP	Lasso	TSGD	OLST	OLST-then-TSGD	
					$C = 50$	$C = 500$
R^2	0.483	0.501	0.389	0.538	0.396	0.522
Time (s)	6688.576	2564.637	3.490	565.241	4.474	7.194

Table 1: The performances of various algorithms on the age prediction task on Wikiface dataset

In our simulation, we divide the data into five balanced subsets and do a 5-fold stratified cross-validation to decide on various hyperparameters and the sparsity level k, we run the algorithms 100 times and take the average performances for each fold; we pick k that produces the highest R^2 average on the test set. We discover that for all algorithms, $k \approx 25$ gives the highest R^2, suggesting that the true support of the age regression problem might only have 25 non-zero features. We report our results in Table 1.

Again, the OLST performs the best but is also the slowest among the online algorithms we present in this paper. However, OLST-then-TSGD with $C = 500$ performs very similarly to OLST while taking way less time to run; hence, this might be the best algorithm for this case considering the trade-off.

6 Conclusion

We have presented two recent methods on feature selection for online regression, TSGD and OLST, and suggested a hybrid method, OLST-then-TSGD. In our simulations, we show that OLST has the lowest error and the highest true support detection, but is the slowest among the three algorithms, while TSGD is the fastest, but it has a low true feature detection rate. Our proposed method, OLST-then-TSGD, maintains a high true support detection rate while requiring less time and memory, especially with an appropriate switch parameter C, as shown from experiments on synthetic and Wikiface datasets.

References

[1] Jianqing Fan, Wenyan Gong, Chris Junchi Li, and Qiang Sun. Statistical sparse online regression: A diffusion approximation perspective. In *International Conference on Artificial Intelligence and Statistics*, pages 1017–1026, 2018.

[2] Elad Hazan et al. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016.

[3] Steven CH Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. Online learning: A comprehensive survey. *arXiv preprint arXiv:1802.02871*, 2018.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] Rasmus Rothe, Radu Timofte, and Luc Van Gool. Dex: Deep expectation of apparent age from a single image. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 10–15, 2015.

[6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[7] Lizhe Sun, Yangzi Guo, and Adrian Barbu. A novel framework for online supervised learning with feature selection. *arXiv preprint arXiv:1803.11521*, 2018.

[8] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[9] Joel A Tropp and Anna C Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on information theory*, 53(12):4655–4666, 2007.

7 Appendix

	True Support Detection Rate (%)						RMSE						Time(s)					
n	OMP	Lasso	TSGD	OLST	OLST-then-TSGD $C=50$	OLST-then-TSGD $C=500$	OMP	Lasso	TSGD	OLST	OLST-then-TSGD $C=50$	OLST-then-TSGD $C=500$	OMP	Lasso	TSGD	OLST	OLST-then-TSGD $C=50$	OLST-then-TSGD $C=500$
$p=1000, k=10$, strong signal, $\beta^*=(1,\dots,1,0,\dots,0)$																		
10^3	100.00	48.20	15.56	89.55	19.97	74.31	7.67	5.43	8.314	1.147	7.515	1.869	7.360	4.385	0.006	2.053	0.112	1.029
$3\cdot10^3$	100.00	73.16	15.56	100.00	20.38	82.97	5.91	5.02	6.051	1.018	5.510	1.544	32.062	23.130	0.020	6.081	0.174	1.037
10^4	100.00	80.03	15.56	100.00	20.05	90.48	4.01	3.45	3.788	1.003	3.484	1.303	51.925	43.128	0.034	21.458	0.244	1.094
$p=1000, k=100$, strong signal, $\beta^*=(1,\dots,1,0,\dots,0)$																		
10^3	100.00	44.12	18.55	82.19	21.66	71.22	22.01	16.73	34.072	7.123	31.346	9.989	6.438	5.220	0.005	2.248	0.133	1.015
$3\cdot10^3$	100.00	46.50	18.55	100.00	24.01	90.08	15.43	10.84	24.950	2.041	22.521	4.349	31.426	24.710	0.018	5.998	0.157	1.124
10^4	100.00	72.82	18.55	100.00	26.24	92.80	12.31	9.01	19.926	1.020	17.972	3.026	52.408	45.384	0.032	22.037	0.274	1.226
$p=1000, k=10$, weak signal, $\beta^*=(0.1,\dots,0.1,0,\dots,0)$																		
10^3	100.00	7.02	5.50	13.30	6.49	11.27	1.150	1.108	1.056	1.035	1.053	1.040	9.015	8.278	0.006	2.442	0.130	1.140
10^4	100.00	14.93	5.50	23.51	6.56	20.75	1.104	1.003	1.019	1.004	1.013	1.003	86.490	62.738	0.083	21.247	0.333	1.222
10^5	100.00	40.51	5.50	81.44	7.38	43.19	1.086	0.999	1.004	0.999	1.003	0.999	1165.689	489.248	0.699	109.778	0.831	1.324
$3\cdot10^5$	100.00	50.05	5.50	99.27	7.66	56.78	0.999	0.996	0.999	0.996	0.996	0.995	3962.462	1296.416	1.984	352.582	2.364	2.427
$p=1000, k=100$, weak signal, $\beta^*=(0.1,\dots,0.1,0,\dots,0)$																		
10^3	100.00	14.03	9.09	11.10	10.50	11.39	2.319	1.246	1.102	1.096	1.101	1.099	9.144	6.578	0.003	2.864	0.174	1.115
10^4	100.00	30.26	9.09	21.86	11.02	14.71	2.055	1.209	1.023	1.052	1.024	1.050	87.616	60.839	0.064	21.483	0.246	1.122
10^5	100.00	78.93	9.09	80.58	11.73	19.32	1.934	1.106	1.019	1.030	1.021	1.034	1141.712	497.866	0.721	126.351	1.282	1.433
$3\cdot10^5$	100.00	96.88	9.09	98.72	12.87	28.89	1.320	0.999	1.006	0.999	0.004	1.003	3785.814	1384.512	2.134	351.096	2.578	2.471
10^6	-	-	9.09	100.00	13.04	29.99	-	-	1.004	0.998	0.999	1.001	-	-	6.594	1008.134	15.261	6.750
$p=1000, k=10$, varied signal, $\beta^*=(0.1,0.2,\dots,1,0,\dots,0)$																		
10^3	100.00	39.10	15.49	74.32	16.18	65.67	5.547	3.766	5.446	1.132	5.018	1.583	7.323	4.920	0.005	2.122	0.122	0.900
10^4	100.00	65.60	15.49	88.18	17.78	79.25	3.214	2.566	2.700	1.017	2.538	1.194	58.047	47.929	0.030	21.403	0.246	0.967
10^5	100.00	84.46	15.49	90.68	23.19	81.21	2.491	1.663	1.806	1.011	1.718	1.090	545.671	458.803	0.673	89.363	1.309	1.055
$p=1000, k=100$, varied signal, $\beta^*=(0.01,0.02,\dots,1,0,\dots,0)$																		
10^3	100.00	32.90	9.42	32.87	9.75	10.66	10.206	7.355	14.249	3.466	13.084	4.525	8.196	6.103	0.004	2.648	0.161	1.314
10^4	100.00	50.49	9.42	40.53	11.01	17.32	6.086	4.248	8.529	1.038	7.756	1.852	75.293	55.430	0.053	21.677	0.287	2.009
10^5	100.00	82.28	9.42	85.26	11.04	22.19	4.422	3.076	4.559	1.007	4.197	1.366	925.540	482.457	0.581	159.258	0.821	5.348
10^6	-	-	9.42	100.00	13.15	25.06	-	-	3.707	0.985	3.425	1.283	-	-	5.994	1413.637	12.912	4.840

Table 2: Comparison between online and offline sparse regression algorithms on synthetic dataset, averaged over 100 runs