CS 764 Project Final Report
Topic: Effect of Storage Format on Stochastic Gradient Descent and Stochastic Coordinate descent

Yusong Yang, Qun Zou

Abstract:
There is a huge interest in understanding how common machine learning algorithms interact with the way in which the data is laid out. In this project, a simple in memory storage manager is built to evaluate the effect of storage format on common machine learning algorithm stochastic gradient descent and stochastic coordinate descent. It is found that the row-wise storage format is preferred by stochastic gradient descent and the column-wise storage format is preferred by stochastic coordinate descent. In our evaluation, the time consumed by stochastic gradient descent increases linearly to the size of datasets and the time consumed by stochastic coordinate descent increases much faster than the increasing rate of the size of datasets. Surprisingly, a "square" storage format is preferred by both of the algorithms and can enhance the algorithms performance in certain scenario.

1. Introduction:

In recent years, data has become a ubiquitous resource. A lot of applications of machine learning to BigData in recent years has result in very large datasets, which include millions to billions of example and features. Moreover, many applications in statistical data analytics is also benefited from large datasets and in turn put the research of management and practice of large datasets forward.

In common machine learning (include a lot of statistical data analytics) tasks settings, especially in supervised learning tasks. Researchers are given a training set of data, which is usually called examples or instances, each example or instance includes a set of features and a label. Machine learning models are trained to predict the label of instance without knowing the label of instance or example. In this problem settings (shown in Fig. 1), training dataset can be viewed as two part, one is m × n matrix, each row of data corresponding to one example or instance of the training set containing n features. The n is usually called the dimension of the training dataset. The label of the datasets can be viewed as another m × 1 or m × k matrix. The machine learning model is treated as a n-dimensional vector, each prediction of an instance is the cross product of the instance and the model, the prediction result is usually a number of a k-dimensional vector corresponding to label. In machine learning tasks, the goal is find a model that can best predict the training datasets, which is, find a d-dimensional vector that can minimizes a certain objective function, also known as loss function, which describe the discrepancy of predicted value and labeled value.

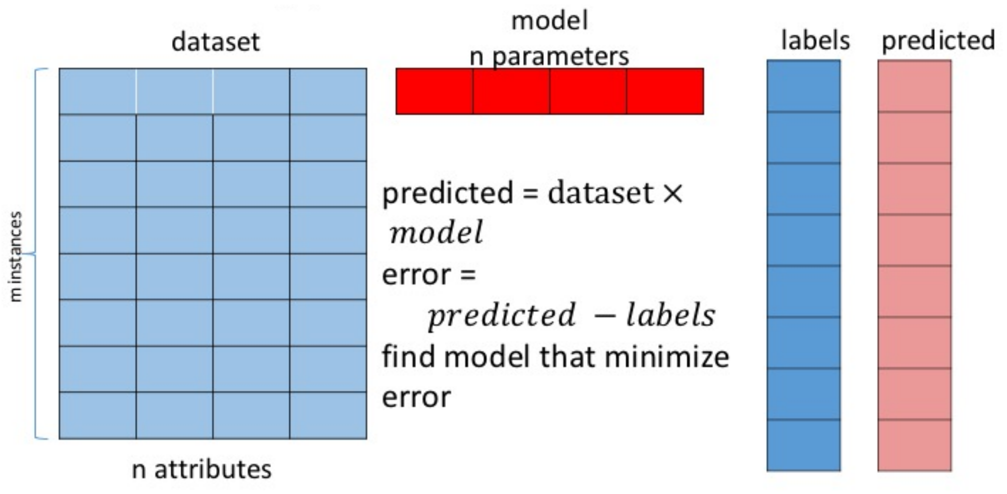
dataset
model
n parameters
labels
predicted
m instances
n attributes
predicted = dataset × *model*
error = *predicted − labels*
find model that minimize error

Fig. 1 Machine learning problem settings.

In the above problem settings, the process to find a d-dimensional vector that can minimize a loss function can be viewed as an optimization problem in d-dimensional attributes space. Thus, many developed optimizations can be utilized to achieve this goal such as Newton Method, Interior Point Method, Gradient Descent, Coordinate Descent, etc. However, in dealing with large datasets, many traditional approaches such as Newton Method and Interior Point rely on basic linear algebra routines such as Chelosky, LU, which are prohibitive in computational cost, and moreover, makes algorithms hard to be performed on a parallel and distributed machine setting. Thus, Gradient Descent and Coordinate Descent, in particular, Stochastic Gradient Descent (SGD) and Stochastic Coordinate Descent (SCD), have attracted many research interests and extensive effort have been made to enhance the performance of the above algorithms.

The psudocode of SGD and SCD are shown in Fig. 2. In SGD, the initial input is data matrix, which include every instance and their features, the step size, which a number determine how fast every step move to convergence, usually a small positive number and the gradient formula of objective loss function. The procedure of SGD is that there is initially a model m, and the model is given an instance, the model calculates the predicted value of that instance according to the instance's features. Then, based on the given label and predicted label, objective loos function is calculated and model is updated along the gradient of objective loss function in features space until the convergence of objective loos function is reached. It is seen that in SGD, every instances are used to do the calculation and update the model. In common machine learning tasks, especially in neural network, instances are even scanned for several times, and each full scan is called one epoch. The advantage of SGD is that in SGD, the descent direction is along the steepest descent direction and this feature ensures a relative rapid convergence rate. However, the disadvantage is that it requires the evaluation of gradient after each prediction, for some complex function, this can be pretty expensive. Similarly, in SCD, the data matrix and step size are given, there is an initial model and initial output. In each step in SCD, a coordinate is randomly chosen to update the model. The key issue here is how to update the model. One way is to take certain amount of step, this a simple but working approach. The advantage of this approach is that gradient function is not needed and the calculation of gradient is avoided. However, the disadvantage is that not every step is guaranteed to descent, for some step, the objective loss function will increase its value. Another way is take step along the partial derivative direction along the chosen coordinate. For this approach, descent is guaranteed at each step, however, the tradeoff is that gradient function has to be evaluated at each step. After update of the model, the error of each instance will be updated and also objective loss function will be updated until convergence. Similarly, a full scan, or certain times of scan when choosing coordinate randomly, of coordinates is called an epoch. Certain number of epochs are needed to train the model.
From the procedures of SGD and SCD, it is observed that in SGD, each instance or row of data is scanned and in SCD, each coordinate or column of data is scanned. The former dataset access method is called row-wise access and the later dataset access is called column-wise access. Moreover, in traditional relational database, relations or tables are set of tuples which include several attribute values. Tuples are grouped naturally in row-wise storage, which is good for transaction processing. However, in machine learning settings, the access pattern of SGD and SCD is different from transaction processing in relational database. Thus, it is worth to explore how dataset laid out can influence the performance of SGD and SCD, especially, explore how SGD and SCD will perform when dataset is grouped in a row-wise storage style or column-wise storage style.

Stochastic Gradient Descent(SGD)

```
Input: Data matrix A, Stepsize α,
       Gradient function grad.
Output: Model m.
Procedure:
    m ← initial model
    While not converge
          ┌ For each row z in A:
    epoch ┤     m ← m - α grad(z, m)
          └ Test convergence
```

Stochastic Coordinate Descent(SCD)

```
Input: Data matrix A, Stepsize α,
       Gradient function grad.
Output: Model m.
Procedure:
    m ← initial model
    While not converge
          ┌ For each rand(coordinate) c in A:
    epoch ┤     m ← m - α grad(c, m)
          └ Test convergence
```

Fig. 2 Psudocode of Stochastic Gradient Descent (left) and Stochastic Coordinate Descent (SCD)

The following of paper are organized as follows: in section 2, we will clearly describe the problem settings and how they relate to current work; in section 3, we will discuss out approach methods and experiments setup; in section 4, the results are discussed and evaluated and in section 5, several conclusions are reached.

2. Problem Settings and Related Work:

As stated in above session, in this project, we present some exploration on how storage format influence the performance of SGD and SCD. To address the key point in this issue, there are several problem settings in this project.

First, in this project, row-wise storage, in which data instances are stored in a row, and column-wise storage, in which data instances are stored in a column-wise storage, will be utilized to examine the performance of SGD and SCD. The reason we choose these two storage format is that these storage format corresponding to the data access pattern in SGD and SCD respectively.

Second, by reviewing the algorithms operation in SGD and SCD, two stopping criteria are used to terminate the loop, when convergence is reached or certain number of epochs are reached. In common machine learning tasks especially in neural network, epochs are usually used as stopping criteria, since the evaluation of convergence is usually computational expensive and the reach of global minima usually result in other issue such as overfitting. In this sense, the performance of SGD and SCD algorithms can be decomposed to two parts, one is resource consumed to scan of data and updating the model, the other is convergence rate of algorithms. Convergence rate is inherited performance of certain algorithm, thus, it is reasonable to isolate this issue with scan of data. In this project, we will focus on the resource consumed to scan of data and updating model.

Third, the price of DRAM is continuously dropping and the memory densities is continuously increasing in recent year, which makes it possible to build high-performance main-memory database and analytics system. Systems that keeping all data in memory all the time have unique advantages that I/O cost is avoided. Thus, in this project, out storage manager will operate in memory and all our experiments will be performed in memory.

In summary, in this project, we will build an in memory storage manager to explore how row-wise storage or column-wise storage format will influence the performance of SGD and SCD, in particular, we will focus on the scan of data part of SGD and SCD, which is close to many machine learning task setting.

Many researches regarding different style of SGD (HogWild!, batch SGD, etc.) and other algorithms have been carried out recently to explore the performance especially the statistical efficiency in newly emerged machine settings such as in a distributed system or in a parallel style [1, 2, 3, 4]. In DimmWitted paper [1] regarding study of main-memory statistical analytics, data access methods are treated as a tradeoff to increase the performance of system. In that paper, data access method is identified as a important issue to SGD and SCD. However, storage format is not taken into consideration in their research. In another paper regarding evaluation of storage organizations for read-optimized main memory database [5], it is found that both row-wise storage and column-wise storage format display performance advantages for different types of queries. That is, for certain data access pattern, storage format can play an important role in the queries performance, in our case, epochs or scan of data can be viewed as a kind of special queries such as select instances one at a time and scan over whole relation. Thus, by reviewing current researches, it is found meaningful in this project to explore the effect of storage format on the performance of SGD and SCD to supplement current researches.

3. Method and Experiments Setup:

As stated in above section, the focus of this project is to explore the effect of storage format on the performance of SGD and SCD by building an in-memory storage manager. Thus, the first part of this project is to build an in-memory storage manager and the second part of this project is examine performance of SGD and SCD on different storage format.

3.1 Storage Manager.

We choose to utilize vector in C++ language as implementation of our storage manager. The reason is that we can easily insert, delete, update data instances by using vector. The other advantage is that in vector, data instances are stored in continuous memory, which provide provide efficiencies to scan of data and a fair benchmark to compare the performance of SGD and SCD in different conditions.

The key part of the storage manager is that it can change the storage format from row-wise storage to column-wise storage as shown in Fig. 3. In row-wise storage, each row of data is one data instance, the attribute values of the instance are stored as a row in dataset. Each row of data is indeed a vector, and many vectors are grouped by a parent vector as dataset. In contrast, in column-wise storage, data instances are stored as a column in dataset, that is, data instances are actually separated by their attributes, and each attribute values of all instances are grouped together as a vector, and many vectors are grouped by a parent vector as dataset. The transformation of storage format is very similar to the transpose operation to a matrix in linear algebra. However, since for dataset or relation, each attribute value of data instance has its meaning, during and after transformation, the access of data instance's attribute values are carefully take care of to guarantee the consistency.

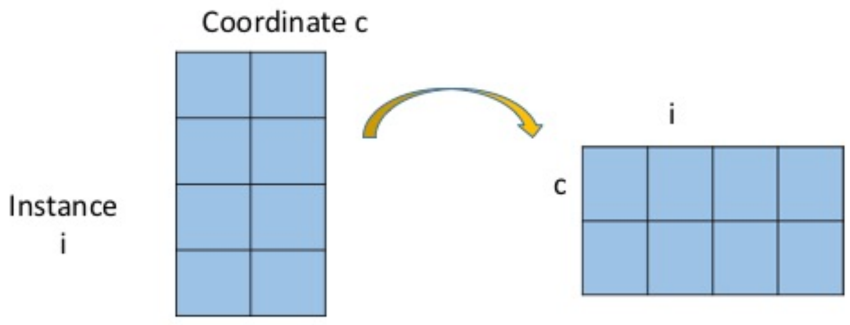

Fig. 3 Illustration of transformation from row-wise storage to column-wise storage.

3.2 Experiments Setup

Our experiments are carried out on a machine equipped with a single CPU clocked at 3.20GHz and 16GB memory. The operating system is Ubuntu 14.04LTS. In our experiments program, learning rate is 0.01 and epochs are set as 5. A single layer neural network classifier is used to model the machine learning tasks. Sigmoid function is used as activation function of output unit in this single layer neural network. SGD and SCD are implemented as the back propagation algorithms to update the model. The size of datasets is controlled by the number of attributes of each instances and the number of data instances. These two numbers are taken from arguments of main function, which is convenient to control. Datasets with different size are randomly generated. Noises are introduced to datasets to mimic real data. During the operation of experiments, memory use is monitored by program provided program and every dataset is guaranteed to be stored in memory without swap to disk. Time consumed by SGD and SCD are recorded in program. To be fair, each algorithm is performed on two storage format on same dataset. Since we focus on the time consumed in scan of data in each algorithm, convergence rate is not considered. In summary, we build a simple in-memory storage manager and provide a fair benchmark to both algorithms in our experiments setup.

4. Results and Evaluation:

4.1 General Comparison Between SGD and SCD

We first perform experiments and measure the time consumed by SGD and SCD in different storage format. In our experiments settings, the learning rate is set as 0.01 and number of epochs is set as 5. As for the datasets, the number of attributes is set 100 in this experiments and the size of datasets are varied by change the number of instances from 10 to 100000. The actual size of datasets is changing from several KB to hundreds of MB.

The recorded time consumed values are shown in Fig. 4. It is shown that, no matter the dataset is stored in row-wise or column-wise format, SGD consumes less time than SCD generally. Since our focus lies on the time consumed in scan of data, comparison between SGD and SCD here is fair enough. From our results, only considering scan of data, SCD is much slower than SGD in general. Moreover, it is worth to note that, for same optimization problem settings, especially non-convex optimization case, SGD usually converge in hundreds of iterations while SCD usually converge in thousands of iterations.

Recall that during SCD, after generating a random coordinate to update the model, since the model has been updated, every output of data instances will be updated. That is, at every update, output of data instances will be scanned, this may be the reason that SCD is slower than SGD in general no matter which storage format is used.

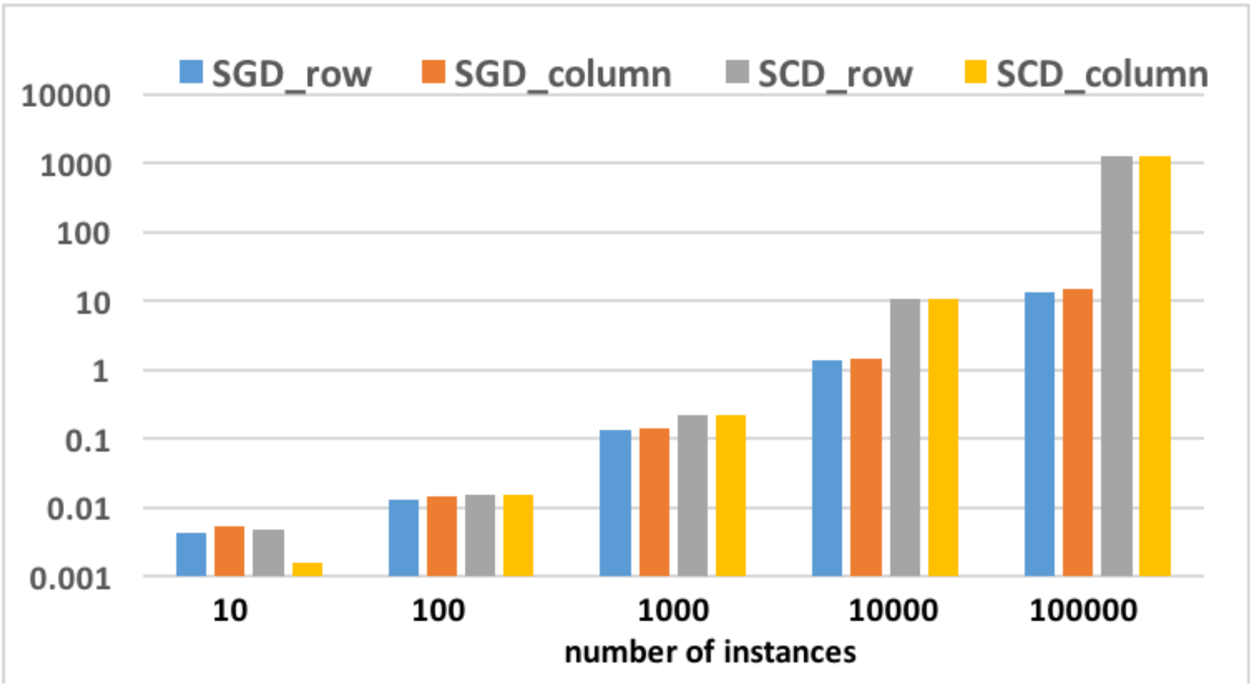

Fig. 4 Time consumed by SGD and SCD in row-wise and column-wise storage with different number of instances, learning rate is set as 0.01, epoch number is set as 5, number of attributes is set as 100.

4.2 Comparison of Time Consumed by SGD and SCD in Row-wise Storage and column-wise Storage

We perform this set of experiments in the same experiment settings as last experiment, in which number of attributes is set as 100 and the size of dataset is increased by increasing the number of instances in dataset. The number of instances range from 10 to 10000000 and the actual size range from several KB to over 5GB

The time consumed by SGD in different storage format are shown in Fig. 5. From the results, it is shown that, for every dataset, time consumed by SGD in row-wise storage is less than that in column-wise storage. The difference is around 10%. This result indicate row-wise storage is preferred by SGD. The other interesting result is that the the time consumed by SGD increases linearly with the increasing of number instances. This is a very good good property since it is a reasonable estimation when deal with dataset at large scale.

To interpret the results, since in our approach of storage manager, vectors are used as implementation. In this approach, elements of vector are arranged in continuous memory. When data instances are stored in row-wise storage format, SGD is performed by scanning several times of continuous memory. However, when data instances are stored in column-wise storage, attribute values of a data instance are separated, SGD need to go back and forth several times in memory to find values to read, write and update. This results in extra time consumed. It is worth to note that this results is very similar to the result of time consumed in queries of scanning dataset stored in row-wise storage and column-wise storage shown in [5]. Our result indicates when using SGD, it is better to store the data in row-wise storage.

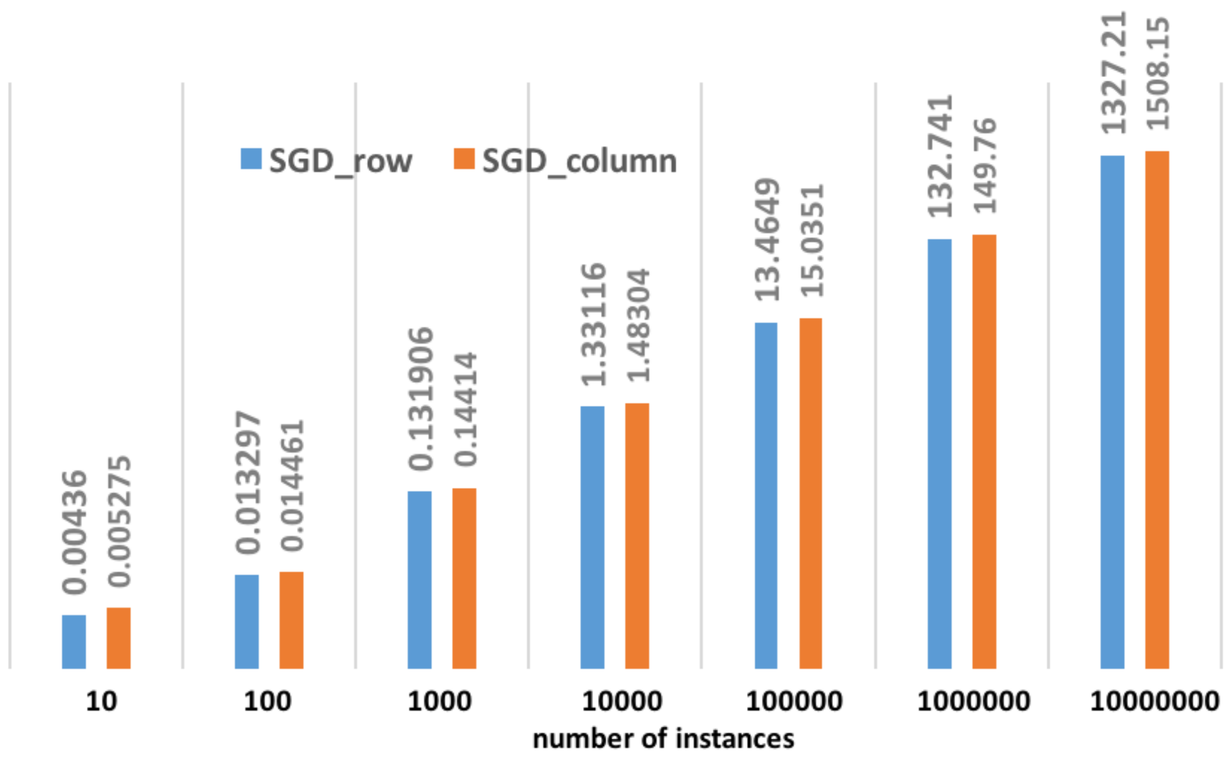

Fig. 5 Time consumed (in seconds) by SGD in row-wise storage and column-wise storage.

The results of time consumed by SCD in different storage format is shown in Fig. 6. The experiments settings is identical to that in SGD experiment shown above. We only be able to scale

the number of instances up to100000 since the time consumed for more instances is too much to record.
From the shown results, it is found that time consumed by SCD is different in different storage format. For most of datasets, the time consumed by SCD in column-wise storage format is less than that of in row-wise storage format. This results are similar to the results of SGD shown above. The reason is that, in SCD, attribute values are scanned several times during an epoch and thus column storage format is more suitable. Another important issue to address is that the scalability of SCD is not good compare to that of SGD. The time consumed by SCD increase much faster than that of SGD, which possess a linear growing rate. This is a drawback of SCD, especially when dealing with large scale dataset, the time consuming will be tremendous.

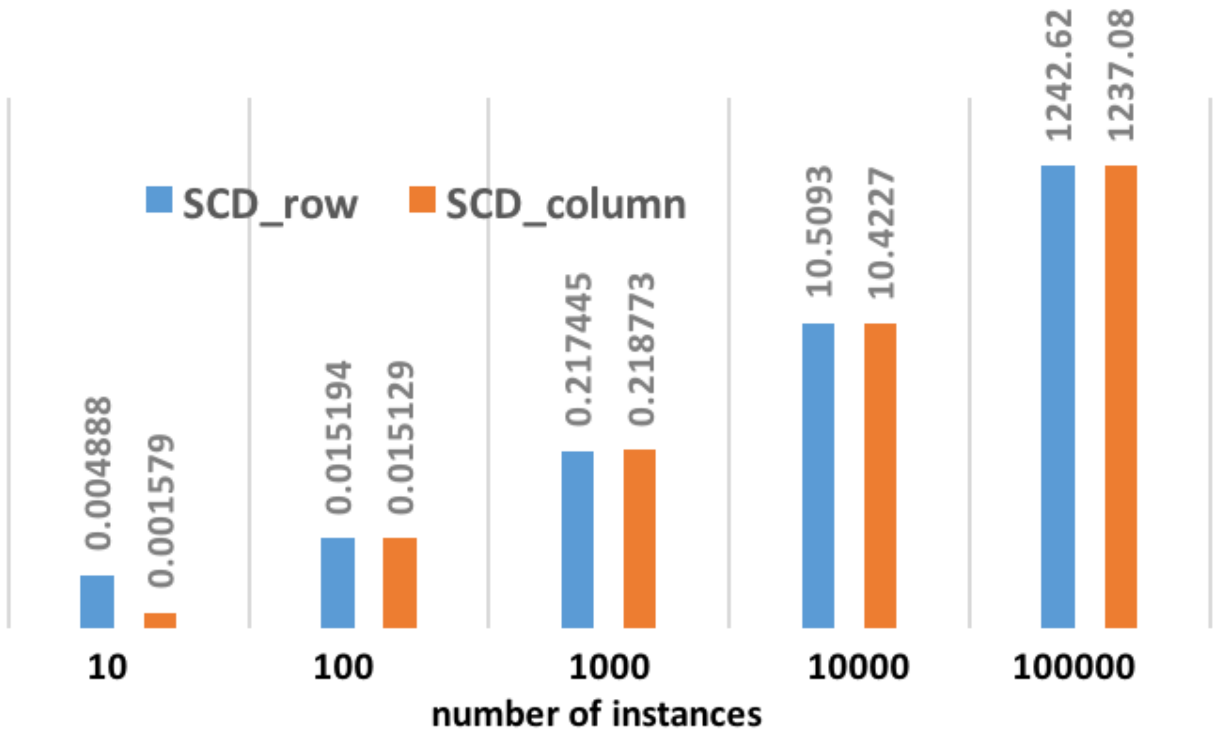

Fig. 6 Time consumed (in seconds) by SCD in row-wise storage and column-wise storage.

From the results of time consumed by SGD and SCD in different storage format shown above. It is found that row-wise storage format is preferred by SGD and column-wise storage format is preferred by SCD. Thus, choose appropriate storage format according to the algorithms to be used is a worthwhile consideration.

4.3 Comparison of Time Consumed by SGD and SCD with Different Matrix Layout

In the above parts of experiments, we set the attributes number to 100 and explore the how SGD and SCD performs in different storage format with different number of instances. In traditional relational database, attribute values are meaningful in each data instances and thus each row represents each tuple. However, if we think the whole dataset as a matrix, and we can adjust the number of rows and columns of this dataset. It is interesting to explore what is performance of SGD and SCD when we vary the matrix layout of dataset. Thus, in this experiments, we fixed the amount of dataset as 1 million data and change the matrix layout of and record the time consumed by SGD and SCD in each matrix layout.

The results are shown in Fig. 7. The matrix layout is represented as number of rows * number of columns. It is interesting to find the results is symmetric in general. When data matrix is long, which means have much rows and less columns, the time consumed by SGD is much less than that of SCD. When data matrix is wide, which means dataset have much columns and less rows, the time consumed by SCD is much less than that of SGD. This consist the symmetric results pattern observed. It is easy to interpret this results since from previous result, we already know that row-wise storage is preferred by SGD and column-wise storage is preferred by SCD.

The most interesting part of the results is that there is a minimal point in time consuming by both SGD and SCD when data matrix are arranged as "square" matrix, which means the number of rows and number of columns are same. This result indicates that the "square" matrix layout is preferred by both SGD and SCD.

It is interesting to see how this result can be utilized. One intuitive thought is that since "square" matrix is preferred, data instances should be grouped such that similar to "square" matrix, however, this would cause unnecessary cost when scanning the data. In a certain case of SGD, which is called batch stochastic gradient descent, several instances are taken together to do prediction, calculate objective loss function and update the model. Several data instances of batch size can be grouped together and stored as a row in data matrix.

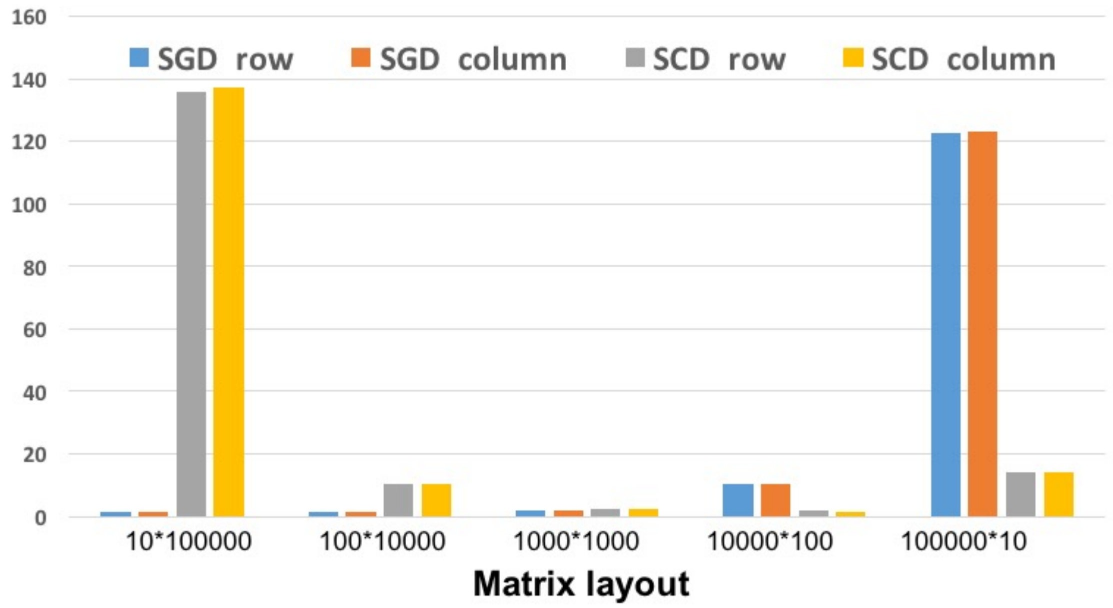

Fig. 7 Time consumed by SGD and SCD for datasets are stored in different matrix layout. The dataset layout is represented as number of rows * number of columns.

From the results shown above, it is found that row-wise storage format is preferred by SGD and column-wise storage format is preferred by SCD. When we change the data matrix layout, a "square" matrix is preferred by both SGD and SCD.

4.4 Discussions and Evaluations

There are several other issues need to be addressed to our experiments settings and this will be expressed as follows.

First, our storage experiments are based on the in-memory storage manager. During the whole experiments, the whole dataset is kept in memory and no data needs to be swapped to disk during the computing process. This is an unrealistic approach in practice, especially for the large scale dataset. However, when the dataset is too large to fit into memory, we would expect row-wise storage would have much better performance than column-wise storage for SGD. The reason is that the column-wise storage would require much more I/O cost to find value to read and update and results in huge time consuming. Thus, storage format will play a more important role when dataset is too large.

Second, our storage experiments are focus on single threads. As for the performance of SGD and SCD on a multithreads or multicore machine settings, there are several research regarding this this issue [1, 2, 3, 4]. On other machine settings, some other issues are worth more considerations. For example, if a dataset is stored in a distributed file system, a SGD can still be performed on each node (HogWild! style SGD), however, the SCD would require much communications between nodes to transfer information. It is worth to note that some of our results would be valid in these cases, for example, row-wise storage format is preferred by SGD. Since SGD can be performed and model can be updated on each node, it is inferred that row-wise storage format is preferred in a node.

Third, our storage experiments focus on basic SGD and SCD. As for other style of SGD such as batch SGD, some of our results is really meaningful. As stated above, "square" matrix layout is preferred by both SGD and SCD. Thus, store the data instances as similar to "square" as possible is better. In batch SGD, several instances are grouped together as a batch and each procedure of batch SGD takes a batch rather a data instance to do the prediction, calculating the error and update the model. Thus, we can store the data in such a way that each batch is stored as a row of data, and this storage format will be suitable for batch SGD, shown in Fig. 8.

Fourth, the transformation of format can be performed for in-memory data processing. As for the storage on disk, some format of storage will make the data layout difficult to be used and find. For example, if we group data instances together as a row, indexes will hard to be built.

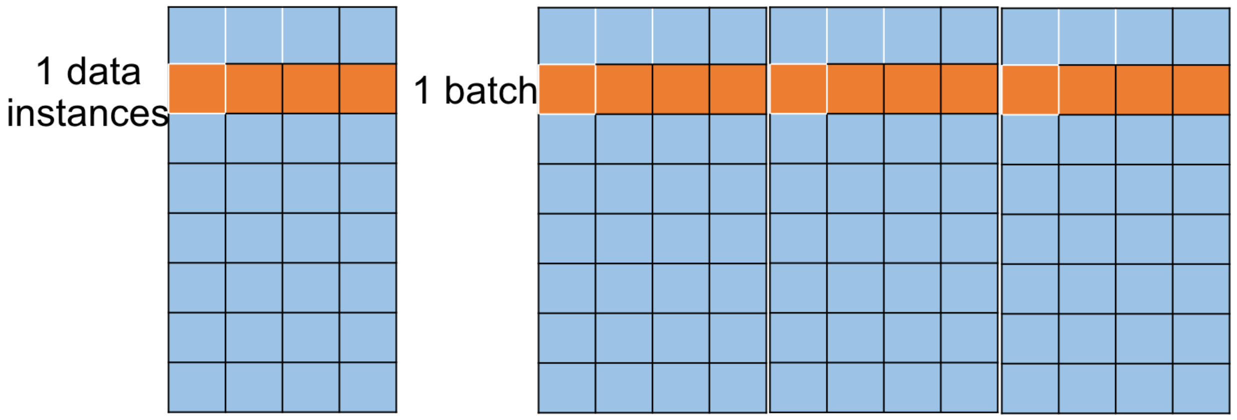

Fig. 8 Illustration of data grouped in data instances and data grouped in batch (three instances).

In summary, our results shown that different storage format can influence the performance of different algorithms, especially, row-wise storage is preferred by SGD and column-wise storage is preferred by SCD. As for the data matrix layout, "square" matrix is preferred by both SGD and SCD. It is inferred that some of results is still valid when algorithms are performed on a totally different machine setting.

5. Conclusions:

In this project, we explore how the storage format of dataset will influence the performance of common algorithms such as stochastic gradient descent and stochastic coordinate descent in machine learning tasks. A simple in-memory storage manager is built to provide row-wise storage format and column-wise storage format of dataset and two algorithms (SGD and SCD) are implemented on top of this storage manager. There are three main conclusions are reached:
First, in machine learning problem settings, storage format can influence the performance of SGD and SCD. In particular, row-wise storage is preferred by SGD and column-wise storage are preferred by SCD.
Second, as for scalability, for SGD, time consumed is increasing linearly with the number of instances while the time consumed by SCD increases much faster than SGD. This indicate that SGD has very good scalability and suitable to deal with large dataset.
Third, for the data matrix layout, it is found that "square" data matrix layout is preferred by both SGD and SCD. This results can be utilized to implement storage format very suitable for batch stochastic gradient descent.
We have explored the influence of storage format on the performance of SGD and SCD. There are still more aspects that can influence the performance of these algorithms and more researches is expected.

6. Reference:

[1] Zhang, Ce, and Christopher Ré. "DimmWitted: A study of main-memory statistical analytics." Proceedings of the VLDB Endowment 7, no. 12 (2014): 1283-1294.
[2] De Sa, Christopher M., Ce Zhang, Kunle Olukotun, and Christopher Ré. "Taming the wild: A unified analysis of hogwild-style algorithms." In Advances in Neural Information Processing Systems, pp. 2674-2682. 2015.
[3] Elgohary, Ahmed, Matthias Boehm, Peter J. Haas, Frederick R. Reiss, and Berthold Reinwald. "Compressed linear algebra for large-scale machine learning." Proceedings of the VLDB Endowment 9, no. 12 (2016): 960-971.
[4] Sallinen, Scott, Nadathur Satish, Mikhail Smelyanskiy, Samantika S. Sury, and Christopher Ré. "High Performance Parallel Stochastic Gradient Descent in Shared Memory." Parallel and Distributed Processing Symposium, 2016 IEEE International DOI: 10.1109/IPDPS.2016.107
[5] Chasseur, Craig, and Jignesh M. Patel. "Design and evaluation of storage organizations for read-optimized main memory databases." Proceedings of the VLDB Endowment 6, no. 13 (2013): 1474-1485.