DIMENSION REDUCTION USING HASHING AND SAMPLING ALGORITHMS

MAJOR TECHNICAL PROJECT (DP 401P)

to be submitted by

SURYAKANT BHARDWAJ, B16117

and

HRUSHIKESH SUDAM SARODE, B16032

for the

MID-SEMESTER
EVALUATION

under the supervision of

PROF. RAMESHWAR PRATAP

SCHOOL OF COMPUTING AND ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MANDI

KAMAND-175005, INDIA

SEPTEMBER, 2019

Contents

Abstract

With the rise of new technologies and Internet, collection of large dataset with has been made easy for solving particular data related problems. However these dataset often has high dimensionality. Due to this one of the most important key challenges that we face is computation on these large datasets. Even with more resources and computational power, performing algorithmic operations on these large datasets has beeen a problem because computaion to be done on these datasets simply outsource resource we have.

One solution to this problems is data compression using dimensionality reduction. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. Though there are several proposed techniques already in field of dimensionality reduction. But one which has caught most interest of the people is dimensionality reduction using hashing techniques.

Hashing is a common routine for large scale data-sets. Hashing addresses both the challenges regarding large data-sets. The challenges in the large data-sets are large cardinality and high dimensionality. With this project, we are aimed to come up with efficient and scalable hashing algorithm which can address both or one of the challenges in large data-sets.

Introduction

In the area of machine learning classification problems, one of the most important factor that drive the final classification is the number of features which are basically variables in these large datasets. The higher the number of features, the harder it gets to visualize the training set and then work on it. Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

Other aspect to consider is that since in the area of Machine Learning and Deep Learning, the size of data we use for training and testing is large. Therefore, it requires the resources which are capable of handling this data. In real life, having those resources for everyone or for every purpose is not possible. For example, in the mobile phone devices, there are applications that uses models locally. Usually mobile phone does not have enough memory or computation capabilities. In this project. we are trying to solve the large-sized data problem using hashing and sampling algorithms. Available techniques, the main objective and solution approach has been explained in this report[2].

Background and Literature Survey

Generally, the data is represented in the form of vectors or tensors(matrices). The data used for Machine Learning or Deep Learning has large dimensions. The data has some property by which we train and predict from models. One of the challenges involved in using this data is its dimensions. In machine learning, the data represents the features over which models work.

Since these datasets are high dimensional, compression schemes for converting them to lower dimensional data must hold a basic property that similarity between data object must be preserved. Though there are various compression schemes that have already been studied for different similarity measures, but the most useful are only those which ensure that when any two data objects from dataset are nearby they should remain nearby in compressed version also and when they are far they should remain far in compressed version also. In following upcoming chapters we would discuss few notable schemes for binary and real valued datasets. And our objective in the entire course of this project will be to overcome its disadvantages and to explore what can be acheived by it.

There are multiple research papers available in this domain. The above technique has been explained in the paper - Feature Hashing for Large Scale Multitask Learning. The paper - Compressing Neural Networks with the Hashing Trick has some techniques to compress trained models.

3.1 Data Compression schemes

First of all compression scheme which a person wants to study depends upon which measure for similarity he want under consideration. Suppose one say two data objects are similar in compressed form if their euclidean distance remains same or if their Inner product is preserved or if their hamming distance is preserved. Here we see Inner product as the important property to be preserved and that's why compression schemes that we would be practising will be focusing on preserving the Inner Product.

One such scheme that had been proposed in the paper "Similarity preserving compression's of high dimensional sparse data" is scheme that we would be working on, which is explained below for two different type of datasets. That is Binary dataset and Real valued dataset.

3.1.1 For Binary Dataset

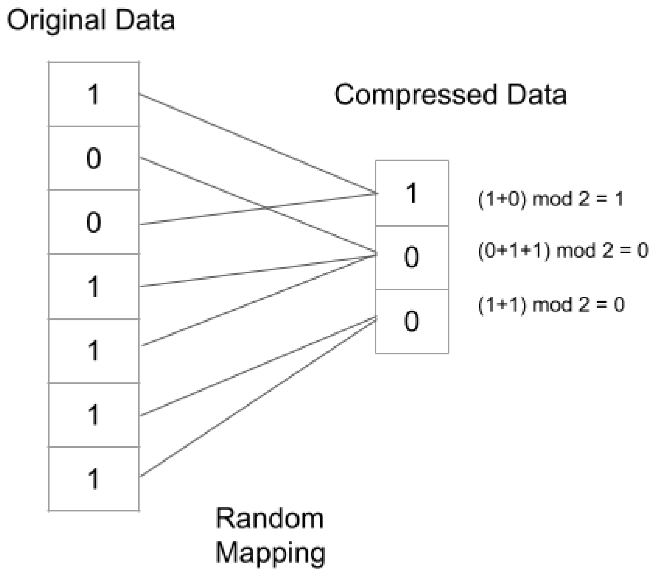

Figure 3.1: Compression scheme example for Binary dataset

Given a Binary vector of N dimension, our mapping converts it into Binary vector of M dimension. About choosing the value of M would be discussed in later part. First a random mapping is generated which maps every feature of higher dimension vector to lower dimension vector. Depending upon mapping, value for each feature in lower dimension vector is generated by adding all the mapped feature values from higher dimension and then taking the mod 2 of the whole summation as can be seen in above example.

The following scheme works under the following consideration.

Theorem 1. *[4] Consider a pair of binary vectors* $\mathbf{u_i}, \mathbf{u_j} \in \{0,1\}^d$ *such that the maximum number of* 1*s in any vector is at most s. If we set* $N = 10 \times s^2$, *and compress them into binary vectors* $\mathbf{u_i}', \mathbf{u_j}' \in \{0,1\}^N$ *via algorithm of [4], then the following holds with probability* $9/10$

$$\langle \mathbf{u_i}, \mathbf{u_j} \rangle = \langle \mathbf{u_i}', \mathbf{u_j}' \rangle.$$

3.1.2 For Real Valued Dataset

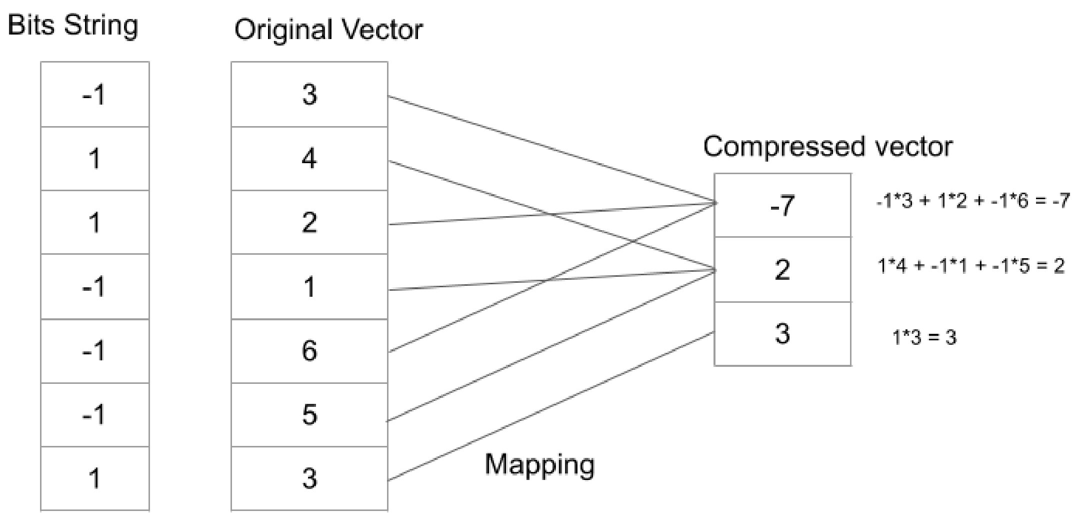

Figure 3.2: Compression scheme example for Real valued dataset

Given a Real valued vector in N dimension, our mapping converts it into Real valued vector of M dimension using the following scheme. First a bit mapping is generated for each feature of a high dimension vector. Then each feature is randomly mapped to features of compressed vector. Now value of each feature in compressed vector is calculated by first multiplying each feature by their bit mapping and then taking summation of all as can be seen in the above example.

The following scheme works under the following consideration.

Theorem 2. *[1]*

Consider a pair of normalised real valued vectors $\mathbf{u_i}, \mathbf{u_j} \in R^d$*. If we set* $N = 10/\epsilon^2$*, (where* ϵ *is the error tolerance, and N is the reduced dimension) and compress them into* $\mathbf{u_i}', \mathbf{u_j}' \in R^N$ *using the feature hashing algorithm of [1], then the following holds with probability* $9/10$

$$\langle \mathbf{u_i}, \mathbf{u_j} \rangle = \langle \mathbf{u_i}', \mathbf{u_j}' \rangle.$$

3.2 Problems to overcome

Now one main point to consider here is that the property that inner product almost remain same under this compression scheme only if mapping generated is completely random. Now this may not seem a problem when one say that for a given N dimension dataset where N is very large a mapping with high randomness can easily be generated, which maps N dimension dataset to M dimension dataset where $M << N$.

But here comes the 2 problems, 1. What should be the value of M for given value of N. Which clearly depends upon how much error in compressed dataset can we tolerate. 2. Since dataset can be changing this means new features can be inserted in dataset as well as deletion may happen. This may break our very first principle to consider that mapping should be highly random. So this creates the need to change mapping such that it remains random with each insertion and deletion.

Objective and scope of the Work

4.1 Objective

Since most of the large data sets or deep learning models are just pool of large matrices representing them, our aim is to somehow reduce the size of these matrices such that the very essence of information while reducing dimensionality must not be destroyed. In short, we need algorithm for reducing dimensionality of data such that the results driven from these data would not change considerably. While looking out for such algorithms, we have been influenced by some sampling and hashing algorithms as discussed above, and we hope to derive an optimal algorithm from the ideas centered in sampling and hashing algorithm.

4.2 Scope

If we are able to find an optimal algorithm for dimensionality compression, then it can be used in many ways in reducing size of datasets for Machine learning as well as for compressing deep learning models. Large data will be compressed while maintaining the essence of information they contain reducing the computational time given earlier to operate on them.

Methodology

Our whole methodology revolves around coming up with suitable data structure and algorithm which can maintain a uniform mapping for a given high dimensional dataset, while changes in dataset are still going on. Which means that still new features can be added and deleted in this large dataset but keeping the mapping uniform and consistent is a key challenge that we are working on.

Our main lookout from Feature hashing technique is to compress two vectors while maintaining their inner product same. Like suppose, we have two vectors of size N as input. We define a size of output vector(Less than N) and make two vectors of that size. We then define a random mapping from input vector to output vector and store it. We also define a random bit string of size N containing $+1$ and -1. Now, we add each element from input vector to its mapping in output vector by multiplying the corresponding bit. This way, we get the two vectors which has size less than N.

We assume the principle that maintaining the inner product of any two vectors from dataset will preserve the features of original data. We have tried the above algorithm and verified that the inner product of two vectors is approximately same using our commpression scheme. Some results are shown in the next chapter. The selection of output dimension also affects the results. For this, we set some value C where $C < N$ such that the errors are minimal for the output vector size greater than C and less than N. However, This technique does not support the insertion and deletions. The naive approach to insert and delete features from input will be to remap everything again. But this will be the heavy operation.

For now, we are trying to come-up with some algorithm that will insert and delete the features holding the above property.

Results

The algorithm that we discussed in previous chapter was coded. We have run the code for variable size if M (Size of compressed output vector). The nature of the Error vs M was as expected. Error decreases with increase in M. From this graph, we get some idea about how we should choose M.

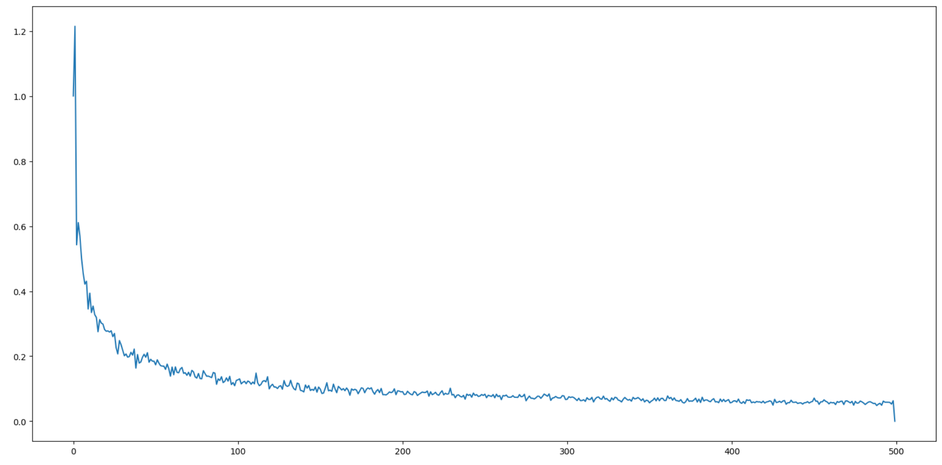

Figure 6.3: Mean square Error vs compressed size for input size = 500.

Conclusion and Future Work

The graph shown in Fig. 6.3 clearly shows that as we increase the value of M, the Mean Square Error decreases. But in other hand, increasing M would demand more space hence more computational power for several purposes. Our first task will be to choose appropriate M such that it reduces the dimensions as well as has less error. In practice, we need to set the parameters for how much space and error we can afford. This may vary for the different ranges of input data size.

In the data compression schemes explained above, feature insertion and deletion are the challenges. The naive way to o it is insert the feature in the input and remap everything. But as this operation is costly, we are trying to find some efficient approach to support these operations. This will require results and statistics of different approaches.

Bibliography

[1] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In Proceedings of the 26th Annual International Conference on Machine Learning (ICML '09). ACM, New York, NY, USA, 1113-1120. DOI: https://doi.org/10.1145/1553374.1553516

[2] Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. 2015. Compressing neural networks with the hashing trick. In Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15), Francis Bach and David Blei (Eds.), Vol. 37. JMLR.org 2285-2294.

[3] Raghav Kulkarni and Rameshwar Pratap, Similarity preserving compressions of high dimensional sparse data, CoRR abs/1612.06057 (2016).

[4] R. Pratap, R. Kulkarni and I. Sohony, "Efficient Dimensionality Reduction for Sparse Binary Data," 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 152-157. doi: 10.1109/BigData.2018.8622338