Algebraic Structures Applied to Machine Learning.

Edward Hernando Lesmes Maldonado
Trabajo dirigido por Fabio Augusto Gonzalez Osorio
Departamento de Matemáticas, Universidad Nacional de Colombia
19 de Junio de 2015

Abstract

Machine learning algorithms can automatically infer statistical models from data in a more effective and efficient way than a manual analysis. It has applications ranging from spam filtering to drugs design. By applying algebraic ideas, it is possible to greatly reduce the time used to evaluate and train machine learning algorithms. We show the results of applying these ideas to the KMeans and Hierarchical clustering algorithms, obtaining running time improvements without sacrificing accuracy.

Preliminaries

Machine Learning tasks can be roughly classified into three different problems, depending on the nature of the data and feedback available to the algorithm.

Supervised learning consists in inferring a model from a set of labeled data.

Reinforcement learning algorithms try to maximize the reward that results from its actions in an environment.

Unsupervised learning is the task of trying to find hidden structure in the available data. It differs from the previous tasks in that there is no feedback signal available to the algorithm.

In this section, we discuss the application of the algebraic notions of monoid and monoid homomorphism to Machine Learning; in the next section we discuss our contribution to the Unsupervised Learning problem ; and finally we test empirically the performance of the proposed algorithm.

Algebraic structures in Machine Learning

Algebraic structures are implicit in most of the techniques and tasks of machine learning. A big first contribution consists in making those structures explicit to facilitate it's recognition and use in other scenarios. Here, we focus in the monoid structure inherent in the training of machine learning algorithms.

In the supervised and unsupervised learning problems we are given a set $\mathcal{D}$ of data points, $\mathcal{D} \subset \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_t$ where $\mathcal{A}_i$ is the type of the i-th attribute and t is the number of attributes.

The algorithm defines a model space $\mathcal{M}$ and consists of a training function $T : \mathcal{D} \longrightarrow \mathcal{M}$.

We consider $\mathcal{D}$ as a monoid, where the operation is given by union. We also require the model space $\mathcal{M}$ to form a monoid with a suitable operation, and the training function T to form a monoid homomorphism. Under these assumptions we can achieve fast online training, near perfect parallelization and fast cross-validation.

Online training is the problem of dealing with data streams, where data points are not available from the beginning, but rather become known as time passes, it is often unfeasible to wait for all the data to be available and then proceed to execute the algorithm. This problem is in general much harder than the offline counterpart.

However, using the algebraic structure, it is trivial to derive a simple and exact algorithm that solves the problem, since the training function is a monoid homomorphism. In most cases, the monoid operation on the model space takes constant time, making it much more efficient than retraining the model from scratch. Furthermore, if the model space has a group structure and the training function is a group homomorphism, it is even possible to *untrain* data points exploiting the algebraic structure.

Parallel training achieves near perfect parallelization. Dividing the data points equally between the p processors and assuming the training function is $O(f(n))$, the total running time is $O(f(n)/p + \log p)$ again exploiting the fact that T is a monoid homomorphism.

Finally, *cross-validation* allows us to estimate the performance of the algorithm when applied to unseen data, by training it in a subset $\mathcal{D}_T \subset \mathcal{D}$ of the available data, and evaluating it in $\mathcal{D} \setminus \mathcal{D}_T$.

This is usually done by partitioning $\mathcal{D}$ into $\mathcal{D}_i$ for $i = 1, \ldots, m$ and performing m iterations, taking $\mathcal{D}_T = \mathcal{D}_j$ in the j-th iteration. By exploiting the algebraic structure we obtain an $O(n + m)$ algorithm, which is much faster than the classical $O(nm)$ algorithm.

Summary of previous work

These ideas are implemented in the Haskell library *HLearn*, along with the application of more algebraic structures, such as groups, rings, modules, vector spaces and categories to machine learning problems. A brief summary is shown below.

Structure	What we get
Monoid	Parallel batch training
Monoid	Online training
Monoid	Fast cross-validation
Abelian group	*Untraining* of data points
Abelian group	Faster cross-validation
R-Module	Weighted data points
Vector space	Fractionally weighted data points
Functor	Fast simple preprocessing of data
Monad	Fast complex preprocessing of data

Table 1. Algebraic structures and their advantages.

Our contribution

The above Haskell library implements a number of Machine Learning algorithms, but only for the Supervised Learning problem. Our contribution consists of a proposed extension of these ideas for the Unsupervised Learning Problem.

The clustering problem consists of grouping the datapoints in a number of clusters, such that the difference of elements between different clusters is greater than the difference of elements within the same cluster.

In this section we describe two algorithms to solve the clustering problem; propose a general framework that applies the notion of monoid to achieve the advantages previously mentioned and apply it to the described algorithms.

KMeans

KMeans is an iterative algorithm that partitions the initial set of n data points into k clusters $C_1, C_2, \ldots, C_k$, so as to minimize the distance of points within the same cluster, i.e. to minimize

$$\sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

where $\boldsymbol{\mu}_i$ is the mean of each cluster.

Initially, a set of k points are selected as initial means. The algorithm proceeds by repeating two phases, until convergence or a fixed number of iterations is reached.

In the *assignment* phase, each of the n data points is assigned to the cluster with the nearest mean. In the *update* phase, the mean of each cluster is recalculated as the mean of the assigned data pointns.

This algorithm converges to a local optimum and is greatly sensitive to the selection of the initial means.

Hierarchical Clustering

Hierarchical clustering builds a hierarchy of clusters, proceeding in one of two ways: By merging the closest pair of clusters, until a single cluster remains; or by dividing each cluster, starting with a cluster containing all data points, until each data point is in it's own cluster.

Different notions of distance between clusters give raise to different merging and splitting strategies and different cluster shapes. These algorithms are usually $O(n^3)$, but $O(n^2)$ algorithms are known for specific notions of distance.

HMeans

The proposed framework relies on the notion of microcluster. In our implementation a microcluster is a tuple $(n_C, \mathbf{x}_C)$, where n_C is the number of data points in the microcluster, $\mathbf{x}_C$ is the sum of the data points in it.

This pair is enough to calculate the cluster centroid without having to store all the datapoints in memory. Furthermore, it has the required algebraic structure described in the preliminaries, and thus has all of the advantages previously described.

The framework starts by selecting a number $m << n$ of data points, and associated to each one an empty microcluster is created. This can be done at random, or by selecting a data point if it is greater than an specified distance d of any previously selected one.

In the second step, each data point is assigned to the microcluster associated to the closest selected data point. These two steps aim to reduce the number of data points, distributing uniformly the data points in a smaller set of microclusters. This is the step in which all the advantages described in the preliminaries can be exploited, to improve the efficiency of the algorithm.

In the last step, a modified version of a clustering algorithm is executed to provide the final clustering of the initial data points. It is usually trivial to modify the algorithms to deal with microclusters instead of individual data points.

Only two passes over the data are required, one to select an initial set of data points and another to assign each data point to a microcluster. Thus, it is expected to outperform algorithms like KMeans, that require multiple passes over the data. Specially in big datasets that don't fit on main memory and have to be read from disk.

On the other hand, the number of microclusters in chosen to be much less than the number of data points. The clustering algorithm chosen in the last step will then run faster if run with the microclusters, than if run from the original set of data points. This makes possible to apply them to larger datasets than it has been previously done.

Results and experimental setting

The algorithms and the framework were tested on the MNIST dataset, a set of 60000 images of handwritten digits. Ten clusters were extracted from the dataset, one per handwritten digits.

The resulting clusters were matched with the ten digits using the Hungarian algorithm, as to minimize the total number of mislabeled images, and the percentage of correctly labeled data points was reported as the accuracy of the algorithm. This procedure was repeated ten times and the mean is displayed below.

For the Hierarchical clustering algorithm, the UPGMA distance between clustering was used, and for KMeans, a parameter of $k = 10$ was used.

The datapoints were selected in the HMeans algorithm with probability $\frac{200}{60000} = \frac{1}{300}$ so that the expected number of microcluster was 200.

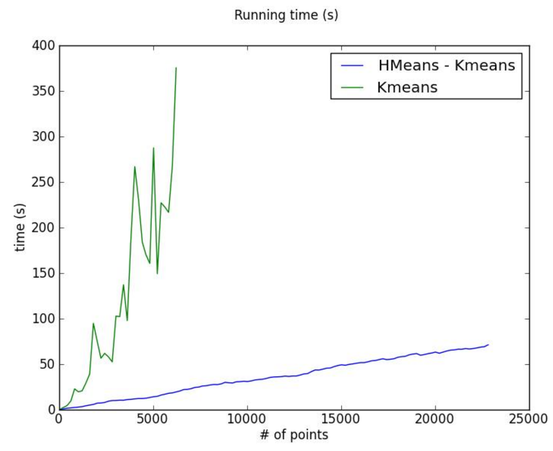

Image 1. Running time of HMeans vs. KMeans

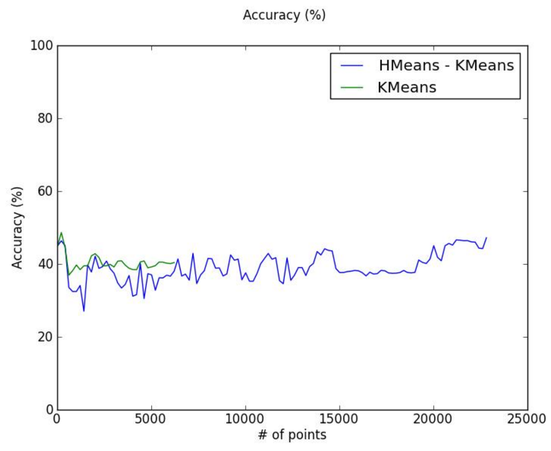

Image 2. Accuracy of HMeans vs. KMeans

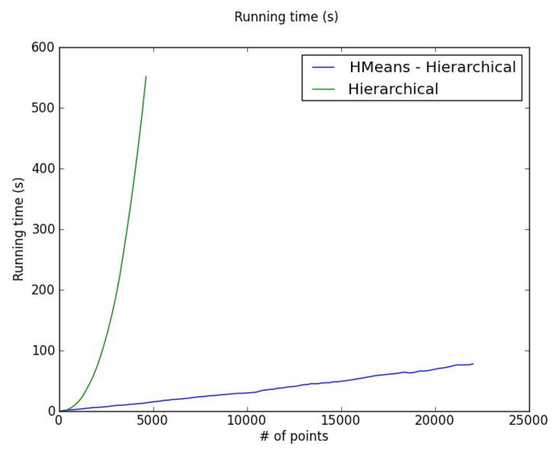

Image 3. Running time of HMeans vs. Hierarchical clustering

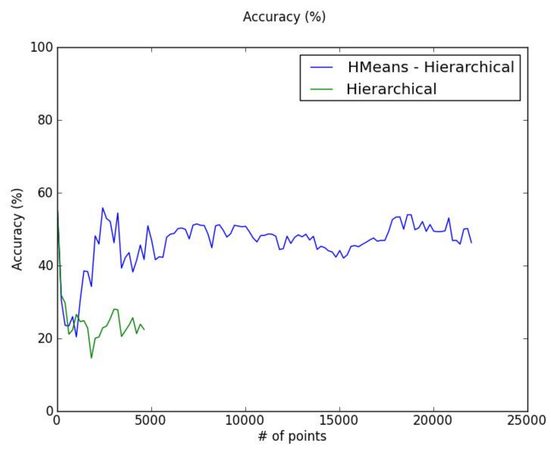

Image 4. Accuracy of HMeans vs. Hierarchical clustering

Conclusion

Algebraic structures provide a general way of improving the efficiency of machine learning algorithms by imposing simple and natural requirements to the training process.

Our work has extended the previous work on the supervised learning problem to the unsupervised learning problem, providing a general framework for the clustering problem that allows for further improvement of the selection strategy in the first step and of the clustering step in the last step.

References

[1] Izbicki, Michael. .[A]lgebraic classifiers: a generic approach to fast cross-validation, online training, and parallel training."Proceedings of the 30th International Conference on Machine Learning (ICML-13). 2013.

[2] Aggarwal, Charu C., et al. .[A] framework for clustering evolving data streams."Proceedings of the 29th international conference on Very large data bases-Volume 29. VLDB Endowment, 2003.