

Jan Oberst

jan.oberst@student.hpi.uni-potsdam.de



Efficient Data Clustering and How to Groom Fast-Growing Trees

Clustering is one of the most important unsupervised learning problems. It deals with finding a structure in a collection of unlabeled data points.

Pattern recognition uses clustering to filter out faces from pictures or video feeds. In medicine clusters represent different types of tissue and blood in three dimensional images, for example to help diagnose tumors. Biologists find functionally related proteins or genes that are co-regulated by scanning large DNA sequences.

This report focusses on the BIRCH clustering algorithm[15]. The authors of BIRCH focus on a scalable architecture. In this report we will discuss the scalability aspects in the field of clustering algorithms and especially those of the BIRCH algorithm. In the section on *Clustering Feature* (CF) trees we then present the main contribution of BIRCH in more detail. A review on further research based on the BIRCH paper concludes this report.

1.1 The BIRCH Algorithm

BIRCH uses a hierarchical data structure called Clustering Feature tree (CF tree) for partitioning the incoming data points in an incremental and dynamic way. This section is organized as follows. First we give an overview of the four phases in the BIRCH algorithm and the construction of the CF tree. Theoretical basics of Clustering Features are given and it is described how distance metrics can work solely on Clustering Features. We finish this section by revisiting how BIRCH can fit large datasets into RAM using these Clustering Features to compress the data.

1.1.1 BIRCH's Four step process

BIRCH employs four different phases during each clustering process.

1. Linearly scan all data points and insert them in the CF tree as described earlier.
2. Condense the CF tree to a desirable size depending on the clustering algorithm employed in step three. This can involve removing outliers and further merging of clusters.

3. Employ a global clustering algorithm using the CF tree's leaves as input. The Clustering Features allow for effective distance metrics. The authors describe different metrics with time complexity of $O(N^2)$. This is feasible because the CF tree is very densely compressed at this point.
4. Optionally refine the output of step three. All clusters are now stored in memory. If desired the actual data points can be associated with the generated clusters by reading all points from disk again.

1.1.2 B^+ Trees for Clustering

BIRCH's CF tree is a height-balanced tree modeled after the widely used B^+ tree. With such a tree no insert or update mechanisms will influence the balance of the tree. This allows fast lookups even when large datasets have been read. It is based on two parameters. CF nodes can have at maximum B children for nonleaf nodes and a maximum of L entries for leaf nodes. As discussed earlier T is the threshold for the maximum diameter of an entry.

A CF tree is built on the fly as the data is scanned. At every level of the tree a new data point is inserted to the closest subcluster. Upon reaching a leaf the point is inserted to the closest entry, as long as it is not overcrowded (diameter $D > T$ after the insert). Otherwise a new CF entry is constructed and the data point inserted. Finally all CF statistics are updated for all nodes from the root to the leaf to represent the changes made to the tree. Since the maximum number of children per node (branching factor) is limited, one or several splits can happen. Splits also result in updated CF values which have to be propagated back to the root of the tree.

1.1.3 Clustering Features

In the BIRCH tree a node is called a Clustering Feature. It is a small representation of an underlying cluster of one or many points. BIRCH builds on the idea that points that are close enough should always be considered as a group. Clustering Features provide this level of abstraction.

Clustering Features sum up a group of points that are close. The authors name such nodes *Leaf Entries*. Every entry represents of a number of data points that are close. The authors further distinguish between *leaf nodes* and *nonleaf nodes*. A leaf node has many entries and represents a cluster that itself is made up of the subclusters of these entries. A nonleaf node follows this principle and represents a cluster that is made up of its child-node subclusters. As suggested before every node in the CF tree is a Clustering Feature. The whole CF tree consists of clusters that themselves have subclusters. This is a classic example of hierarchical clustering[6].

Clustering Features are stored as a vector of three values: $CF = (N, \vec{LS}, SS)$. The linear sum (\vec{LS}), the square sum (SS), and the number of points it encloses (N). All of these metrics can be calculated using only basic math:

$$SS = \sum_{i=1}^N \vec{X}_i^2 \qquad \vec{LS} = \sum_{i=1}^N \vec{X}_i$$

If divided by the number of points in the cluster the linear sum marks the *centroid* of the clus-

ter. As the formulas suggest both of these values can be computed iteratively. Any Clustering Feature in the tree can be calculated by adding its child Clustering Features:

$$CF_1 + CF_2 = (N_1 + N_2, \vec{L}S_1 + \vec{L}S_2, SS_1 + SS_2)$$

1.1.4 Distance metrics

Clustering algorithms use distance measure to discover close data points.

Even the highly compressed Clustering Features still allow the use of exhaustive distance metrics. The authors describe how five different metrics, including the Euclidean and Manhattan distances work on Cluster Features to measure the distance between points and clusters, and the intra-cluster distance.

1.1.5 Data compression with CF trees

Every Clustering Feature encloses points that are “close”. This closeness criterion is based on the *diameter* of a cluster. Because Clustering Features are vector-based the calculation of diameter D or radius R is straightforward (\vec{X}_i are the vectors in a cluster, \vec{X}_0 is the cluster’s centroid):

$$D = \left(\frac{\sum_{i=1}^N \sum_{j=1}^N (\vec{X}_i - \vec{X}_j)^2}{N \cdot (N - 1)} \right)^{\frac{1}{2}} \quad R = \left(\frac{\sum_{t=1}^N (\vec{X}_t - \vec{X}_0)^2}{N} \right)^{\frac{1}{2}}$$

The construction of Clustering Features immediately compresses all incoming data. Even for hundreds of close points only one Clustering Feature has to be stored in RAM. BIRCH can be set to use more or less extensive data compression by setting the *threshold* value T . New points can only be inserted to an entry if its Clustering Feature is smaller than T after the insertion. If the point does not fit it is inserted to a newly created entry.

Noisy datasets generally pose a problem to clustering algorithms. Outliers will generally not fit in any existing cluster, and therefore bias the algorithm to generate clusters with only one data point. BIRCH will never add outliers to existing clusters, since T already limits the cluster radius. To resolve this problem BIRCH automatically filters outliers during tree rebuilds and writes them to disk. Once it finishes the CF tree it will try to re-fit those outliers. This mechanism improves clustering quality because it avoids clusters with only one entry, and increase performance by removing outliers from the CF tree.

Perfect cluster size It can be shown that there is no absolute best criterion which would be independent of the final aim of the clustering: every clustering problem is different. For the same reason there cannot be a “best” cluster size. The more points can be compacted to one Clustering Feature the better the compression ratio gets. On the other hand more information is lost if more points are represented by a cluster. A balance between better level of detail (smaller clusters) and better compression ratio (larger clusters) is crucial for good overall clustering quality.

BIRCH solves this problem by always trying to maximize RAM usage (which maximizes precision). The algorithm starts with maximum precision at $T = 0$ and as the CF tree grows larger than the available memory it iteratively tries to find suitable cluster sizes. T has to be increased to be larger than the smallest distance between two entries in the current tree. This

will cause at least these two entries to be merged into a coarser one, it reduces the amount of clusters produced, and clusters will grow larger in diameter.

Threshold heuristics Rebuilding the CF tree is generally fast enough to allow an iterative approximation to a “best” suitable threshold. All performance analysis in the original paper start with $T = 0$ and the authors note that beginning with a good initial value for T would save about 10% time[16]. The authors also supply basic heuristic to derive a new threshold from an existing CF tree.

Once BIRCH has to rebuild the CF tree it needs to increase T based on some measure of cluster volume. BIRCH maintains a record of leaf radii as a function of the number of points in the cluster. Using least-squares regression it can estimate to future growth of the tree, and extrapolate an “expansion factor”, which will generally be higher for high-volume trees. The CF tree is then rebuilt using T multiplied with the expansion factor as the new threshold.

1.2 Test of time

The authors of BIRCH have achieved groundbreaking results for the scalability of clustering algorithms. The following paragraphs discuss the four most important contributions of the BIRCH paper and their impact on the data mining community.

On the other hand, BIRCH is sensitive to the order of the input stream, only functions properly on spherical data, and is limited to low dimensional datasets. To conclude this section we show how these problems have been solved in other systems.

Single-pass Today many datasets are too large to fit into main memory. From this point on the dominating cost of any clustering algorithm is I/O, because seek times on disk are orders of a magnitude higher than RAM access times.

BIRCH can typically find a good clustering with a single scan of the data. After the initial CF tree has been built additional scans can help to improve the clustering quality.

Adaptive compression As described above BIRCH frequently rebuilds the whole CF tree with a different threshold setting and tries to merge as many CF nodes as possible. The rebuild happens sufficiently fast since all needed data is already in RAM. At the same time outliers are removed from the tree and are stored to disk.

BIRCH is one of the first algorithm to effectively use an in memory index-structure for spatial data. The authors of STING[13] employ similar techniques and claim to have outperformed BIRCH by a very large margin (despite not having compared the two algorithms side by side). The CURE algorithm[3] uses a similar approach to BIRCH’s CF tree: Each cluster is represented by a certain number of “well scattered” points. CURE relies on drawing random samples from the dataset to derive starting clusters. This preclustering phase is somewhat similar to the first phase in the BIRCH algorithm.

Non-local A new data point is always added to the closest subcluster in the CF tree. The first advantage of this approach is the limited amount of comparisons that are needed when a data point is inserted to the tree. Being based on a B^+ -tree insert operations require at

most $\log(n)$ operations. Data points are inserted to the *closest* subcluster at all levels of the tree. This means that only those subclusters are checked that have at all a chance to absorb the added data point.

Suspendable, stoppable, resumable Even scalable architectures like BIRCH have run-times of many hours when large datasets have to be clustered. At this point it is impossible to restart the algorithm every time a customer proceeds to checkout[8]. Once built a CF tree can be used to sequentially add as much data as needed. If the growing CF tree hits the memory limit it is rebuilt and more data can be added.

Order sensitivity BIRCH is sensitive to the order of the input stream. Different orders of the same input data may result in different clusters[4]. This problem becomes even more relevant when the input stream itself is sorted. Each data point is immediately added to the CF tree and all Clustering Features on the path to the inserted leaf have to be changed. If nodes had to be split during the insert even more Clustering Features change.

Two points with the same coordinates could very well end up in different clusters. Both rebuilding the CF tree and the refinement phase 4 of the algorithm try to resolve this problem, but fail to eliminate it completely[3].

Non-Spherical BIRCH uses the concepts of radius and diameter to control the boundary of a cluster. This approach may not work well when clusters are not spherical, because clustering is limited to a vector space. BUBBLE[2] allows single-pass scalable clustering in arbitrary metric spaces based on the BIRCH framework.

Low Dimensionality Agrawal et al.[1] tested three algorithms on a dataset with varying dimensionality from 5 to 50 dimensions. At more than five dimensions BIRCH was unable to identify the true clusters. BIRCH gives equal importance to all the dimensions when computing the distance between two points. Clustering algorithms with a specific focus on high dimensionality tend perform better than BIRCH [9, 1, 11].

1.3 Conclusion

The *SIGMOD Test of Time* award is a significant recognition that is given to a paper submitted to the conference ten years ago. To the SIGMOD judges BIRCH is the most important contribution from 1996 and that with the most impact today.

BIRCH was the first algorithm to run in $O(N)$ time. Its authors claim that their algorithm performed 15 times better than CLARANS[10], the leading spatial clustering algorithm at the time. While BIRCH was still limited to low-dimensional and vectorized datasets, but sparked lots of research on how to effectively cluster large datasets. The algorithms CURE and CLIQUE[1] showed that higher dimensionality can be clustered effectively. With BUBBLE, CURE, and WaveCluster[12] also irregularly shaped clusters became possible.

The reader is referred to data clustering surveys[7, 5, 14] for a more in-depth comparison of the aforementioned clustering algorithms.

1.4 Appendix: Clustering Bundesliga News

Ligageschichte¹ gathers many different sources around the German soccer-league Bundesliga and displays them in many different ways. One of these so-called “portlets” is a classic example for data clustering: Finding popular news topics and clustering them accordingly. The following sections give a short introduction on how BIRCH can be used to cluster news texts. We test how different settings change the CF tree and analyze the resulting clusters. We also compare BIRCH against the k -means clustering Ligageschichte uses today to find out if BIRCH would be the better choice.

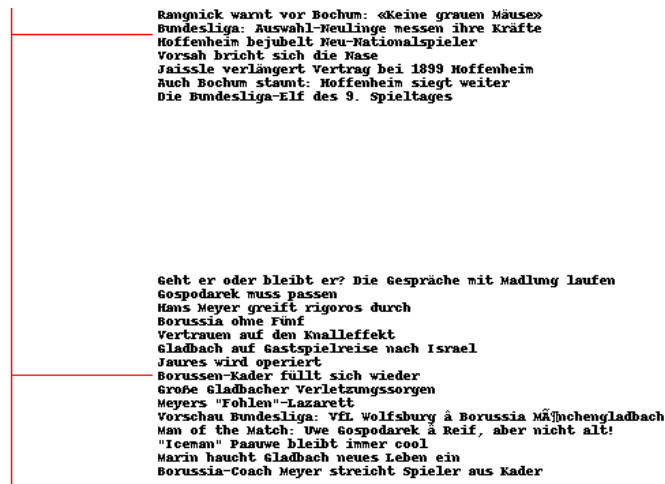


Abbildung 1.1: Two example clusters with news covering Hoffenheim and Mönchengladbach

1.4.1 Test Implementation & Dataset

BIRCH’s authors limited their experiments to two dimensions. In order to test BIRCH on a high-dimensional dataset and to experiment with the different settings for branching factor and threshold we implemented the CF tree in Python. Please note that this section is based completely on phase one of the BIRCH algorithm. Furthermore we exclude automatic tree rebuilds because we’re interested in different CF-tree layouts. The complete source code of our implementation can be found online². All tests were run on a 2.5 GHz Intel Core 2 Duo with 4 GB of RAM. As a result even large CF trees with $T = 0$ will fit into RAM. The smaller test data set with 1000 data points also still fit into disk cache.

The large dataset consisted of 58,096 news texts, including sources from Kicker.de, Bundesliga.de, FAZ.net, and Fussball24.net. Duplicate removal was not performed before clustering. However, all testing with k -means had to be limited to a smaller dataset of only 10,000 texts, because its implementation did not use the main memory as efficiently as it could have.

To generate a proper input dataset for BIRCH we transformed the texts to high-dimensional vectors. Each text was represented by a single vector and each word represented by a single dimension. Ligageschichte uses a fine-tuned Named Entity Recognition (NER) algorithm³. The final step in preparing the news texts for clustering is to assign the proper values to each

¹<http://www.ligageschichte.de>

²http://www.janoberst.com/_academics/2009_01_27-BIRCH-Python-Implementation/

³More details about Ligageschichte can be found at http://www.janoberst.com/_academics/2008_07_16-Ligageschichte-Clustering-LSI-Semantic-SVD.pdf

| Dimension (Entity) | TF/IDF value |
|---------------------------|--------------|
| ‘hingegen’ | 0.0434 |
| ‘niederlage’ | 0.028 |
| Team{VfB Stuttgart} | 0.189 |
| Player{Mario Gomez} | 0.040 |
| ‘roter’ | 0.060 |
| ‘fans’ | 0.175 |
| Player{Maik Franz} | 1.299 |
| Team{Bayer 04 Leverkusen} | 0.642 |
| ⋮ | ⋮ |
| Team{Karlsruher SC} | 1.010 |

Abbildung 1.2: Example vector as prepared for the BIRCH algorithm

dimension. The TF/IDF value represents the importance of a word based on the number of occurrences in the text and the number of occurrences in all texts combined. Term weights for named entities are computed accordingly. Figure 1.2 shows the example of a text vector as used in our example clustering.

1.4.2 Impact of different threshold settings

We tested the algorithm with five different threshold settings $T = \{0.5, 1, 2, 4, 8\}$. Our sample dataset were 1000 normalized and cleaned news items from November 2008. As a result we used a total of 18,099 dimensions. The branching factor was set to $B = 7$, which we found to be a good fit for our dataset.

| T | # Clusters | # Splits | % outliers | Average cluster size | Average cluster radius |
|-----|------------|----------|------------|----------------------|------------------------|
| 1 | 491 | 258 | 0.000 | 2.020 | 0.419 |
| 2 | 362 | 210 | 0.000 | 2.740 | 0.887 |
| 4 | 232 | 176 | 28.879 | 4.276 | 1.694 |
| 8 | 148 | 107 | 17.568 | 6.703 | 3.677 |
| 16 | 83 | 54 | 18.072 | 11.952 | 7.423 |
| 32 | 44 | 35 | 22.727 | 22.545 | 15.933 |
| 40 | 39 | 29 | 20.513 | 25.436 | 18.718 |
| 64 | 24 | 12 | 20.833 | 41.333 | 33.841 |

Tabelle 1.1: Results of the BIRCH clustering phase 1

When the threshold is set as low as $T = 1$ most ‘clusters’ only consist of two or three items. Accordingly, the amount of outliers is low as BIRCH uses the proposed limit of 25% of the average cluster size to find out which clusters are to be marked as outliers. The overall clustering result is not acceptable.

Large thresholds generally produce fewer and bigger clusters. The “average cluster size” as described in Table 1.1 represents the amount of news items per cluster. The most promising results were created with $T = 40$. At that point the overall quality of the clustering was even better than the already highly tweaked clusters that are currently used by Ligageschichte.

With $T = 10$ almost a third of the clusters would have been marked as outliers. As discussed

in section 1.2 we suspect that the very high dimensionality of our dataset is responsible for this behavior. However, the currently used k -means approach does not have any outlier detection at all. Data points that actually are outliers are mixed with proper clusters and worsen the results. Ligageschichte already tries to eliminate this problem by removing the largest and the smallest cluster after the first few passes restarting the algorithm. With BIRCH outliers could be discarded much easier, because they do not influence clustering quality.

1.4.3 Compared to k -means

We conducted a second experiment where we compared the runtime of BIRCH to that of the standard k -means that is built into Ligageschichte.

We had to limit the amount of news items to cluster to 10,000, because the k -means implementation in Ligageschichte already used 930MB of RAM at that point. This caching is done for performance reasons, because the algorithms would need to re-fetch all data from disk for each of the 30 iterations. We suspect that the performance differences will grow with larger test datasets.

k -means was set to use 22 iterations and the number of clusters k was fixed at 30. This amount of clusters has proven to be adequate. Based on the tests described above we chose $T = 40$ as an appropriate threshold for BIRCH.

The clustering output generally consists of 18 clusters for the 18 Bundesliga teams. Another large cluster usually represent news reports on a whole weekend (where all teams appear in the same text). News about player transfers, injured players and the performance of the referees are generally also placed in individual clusters. The remaining (small) clusters are usually outliers which are discarded by Ligageschichte.

| Algorithm | Runtime | # Clusters |
|------------|-------------|------------|
| k -means | 209 minutes | 30 |
| BIRCH | 97 minutes | 38 |

Abbildung 1.3: Comparison of k -means and BIRCH phase 1 for 10,000 news items

Most of the time in both algorithm is lost because of the barely optimized Python implementation. Since real-world clustering experiments are only comparable to themselves, these results do not represent the algorithmic performance differences between k -means and BIRCH.

1.4.4 Conclusion

After tweaking of both algorithms the results of BIRCH were slightly better than those of the standard k -means approach built into Ligageschichte while being two times as fast.

This comparison between BIRCH and k -means makes actually sense in the case of Ligageschichte. We are trying to figure out which clustering algorithm can replace the quite ineffective k -means implementation. The observed speedup of factor two is certainly worth considering a switch.

We suspect that BIRCH would greatly benefit from using a larger dataset. It would also be interesting to see how BIRCH performs with the automatic generation of T using the heuristics described in section 1.1.5.

Literaturverzeichnis

- [1] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 94–105. ACM Press, 1998.
- [2] Venkatesh Ganti, Raghu Ramakrishnan, Johannes Gehrke, and Allison Powell. Clustering large datasets in arbitrary metric spaces. In *ICDE '99: Proceedings of the 15th International Conference on Data Engineering*, page 502. IEEE Computer Society, 1999.
- [3] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: an efficient clustering algorithm for large databases. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 73–84, New York, NY, USA, 1998. ACM.
- [4] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *J. Intell. Inf. Syst.*, 17(2-3):107–145, 2001.
- [5] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.
- [6] Stephen Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.
- [7] Erica Kolatch. Clustering algorithms for spatial databases: A survey, 2001.
- [8] Erendira Rendón Lara and R. Barandela. Scaling clustering algorithm for data with categorical attributes. In *ICCOMP'05: Proceedings of the 9th WSEAS International Conference on Computers*, pages 1–6, Stevens Point, Wisconsin, USA, 2005. World Scientific and Engineering Academy and Society (WSEAS).
- [9] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178, New York, NY, USA, 2000. ACM.
- [10] Raymond T. Ng and Jiawei Han. Clarans: A method for clustering objects for spatial data mining. *IEEE Trans. on Knowl. and Data Eng.*, 14(5):1003–1016, 2002.
- [11] Cecilia M. Procopiuc, Michael Jones, Pankaj K. Agarwal, and T. M. Murali. A monte carlo algorithm for fast projective clustering. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 418–427, New York, NY, USA, 2002. ACM.
- [12] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *VLDB '98: Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 428–439, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.
- [13] Wei Wang, Jiong Yang, and Richard R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *VLDB '97: Proceedings of the 23rd International*

Conference on Very Large Data Bases, pages 186–195, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

- [14] Rui Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.
- [15] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 103–114. ACM Press, 1996.
- [16] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.