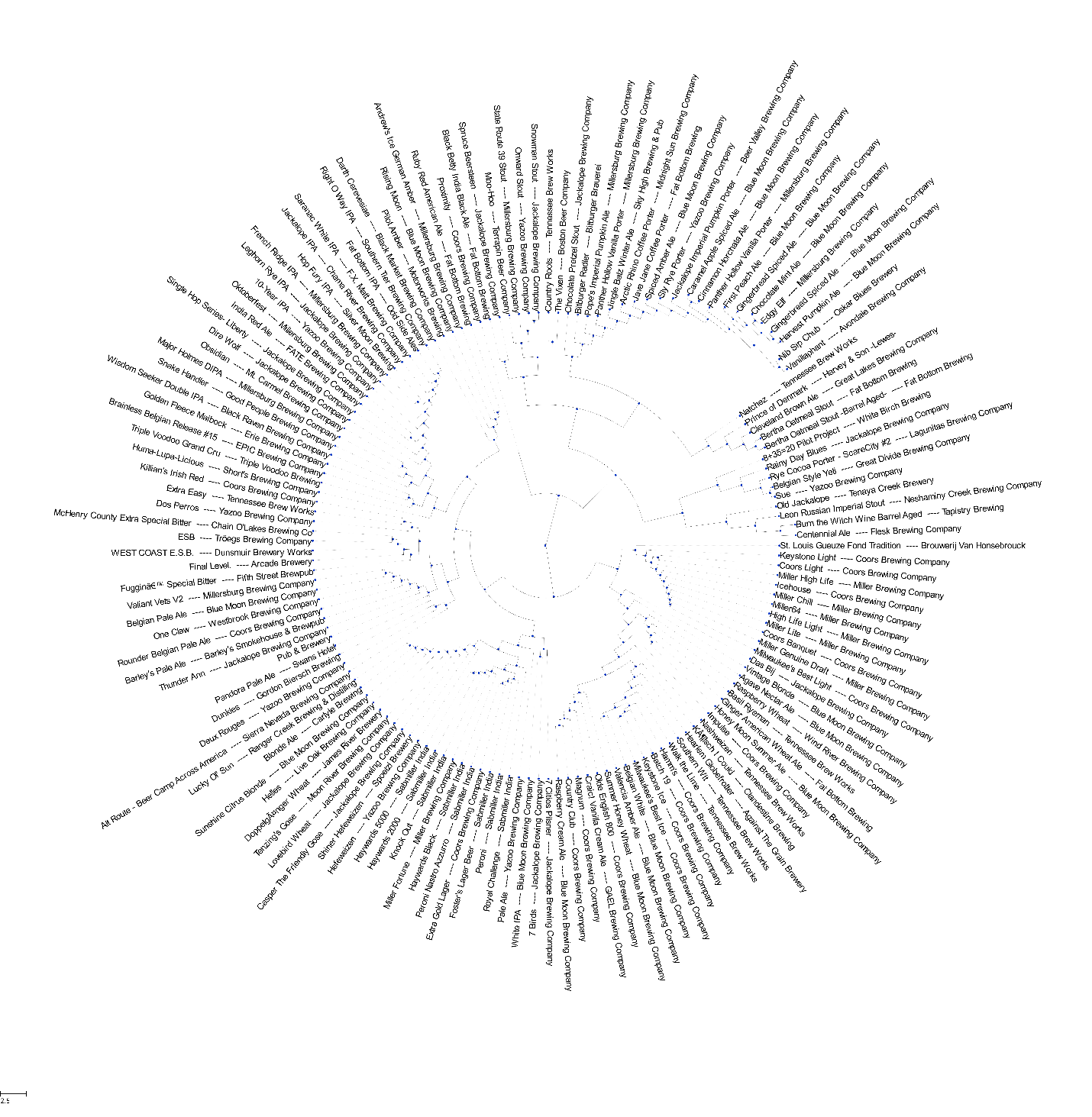
# Hierarchical Clustering

Derek Gloudemans

CS 6360: Advanced Artificial Intelligence

February 26, 2019



## Project Description:

In this project, a hierarchical clustering algorithm was implemented. This algorithm was used to cluster items from a dataset of commercially available beers. Several modifications and enhancements were made to the algorithm to make it more suitable for use with this dataset. In this report, the data type and source are described, the data preprocessing steps are detailed, the implementation of the hierarchical clustering algorithm is explained, and the additional algorithm work is detailed. The results are presented as dendrograms. Lastly, promising directions for future work on this project are identified.

## Data Description and Preprocessing:

This section contains a description of the data source, the raw data, and preprocessing steps completed prior to clustering.

**Raw Data:**

Raw data was obtained from a GitHub repository[[1]](#footnote-2) that compiled data from breweryDB[[2]](#footnote-3), an online and app-based, crowd-sourced collection of information on different types of beer. The data files in this Github repository were created by scraping data from the breweryDB API. This data source was selected over other similar datasets (such as the Kaggle Craft Beers Dataset[[3]](#footnote-4) and the Open Beer Database[[4]](#footnote-5)) due to the completeness of its data and the presence of both numeric (continuous) variables and descriptive (discrete) variables. This raw data was included as the file “beers-brewerydb.csv”, and an example was included in Table 1.

**Table 1: Raw Data**



The goal of clustering this data was to identify clusters of beers that consumers would consider similar in taste to one another, which could be used to inform future selections. The raw data file contained irrelevant information for clustering items (beers) based on similarity of taste (such as recommended serving glass and unique identification codes) as well as redundant information (such as *Standard Reference Metric* (SRM), minimum SRM, and maximum SRM). Irrelevant information was removed and redundant information was used to fill in missing values whenever possible. An example of the resulting, cleaned dataset was shown in Table 2. This data cleaning was carried out in Microsoft Excel. The full, cleaned dataset was included in the submitted files and was named “raw\_data.csv”. All subsequent work was done in Python 3.6. All items with missing data fields were removed. The resulting, cleaned and reduced file contained 25,439 items.

**Table 2: Cleaned Data**



**Feature Extraction from Description:**

In Python, this data was converted into a number of *features* or variables for which every item had a value. Perhaps the most relevant information for classifying taste characteristics was the item description field, which contained a brief summary of beer history, considerations during brewing, and major flavor notes. This field was parsed into relevant features for clustering by the following process. First, all unique words from all descriptions were extracted and ordered by frequency of occurrence. The most common terms were ignored (such as “the” “an”, and “beer”) as these terms were common to nearly every item. Likewise, the least common terms (arbitrarily those that occurred less than 100 times throughout all descriptions) were discarded as they were considered too sparse to contain much discriminatory or associative information between items. These terms were often highly specific terms used to add interest to the description such as “motor” (15 occurrences) and “Mayflower” (14 occurrences) .

From the remaining terms, 412 terms were manually selected based on their perceived relevance for describing flavor or other characteristics relevant for distinguishing between types of beers. For each item, the terms that were present in that item’s description were added as features. This was accomplished by means of k-hot encoding. That is, a new feature was created for every term. Each item was assigned the value one if that term was present in its description, and value zero otherwise.

**Feature Extraction from Short Name and Style:**

The “Style” field and “Short Name” field from the data contained descriptive, categorical information about the common categories each item was placed in. The possible values for these two fields were included in Appendix I. For instance, some values for the Short Name field were “Pilsner” and “Red ale”. These values could not simply be assigned unique integer tags, as this would imply some linear ordering of the values. Thus, k-hot encoding was also used to transform the Style and Short Name fields into features for clustering.

**Final Data Form:**

After all data preprocessing, the final data was stored in matrix form and had dimension 25,439 items x 546 features. Of these features, the first 5 corresponded to the specific gravity, original gravity, alcohol by volume, international bitterness units, and standard reference measure for each item. The next 13 correspond to the possible styles, the next 116 correspond to possible short names, and the final 412 to relevant flavor terms from the description. Each of these features was normalized across all items, and multiplied by a weighting factor (to be explained later).

**Data Selection for Clustering:**

The full 25,000+ item dataset was considered too large for efficient algorithm development, testing and visualization. Thus, a subset of items was selected by two user-input means: specified text and down-sample ratio. The first consisted of a list of strings. Any item with any of the listed strings present in its name was included in the selected dataset. The down-sample ratio randomly reduced the number of items included by a factor equal to the ratio. Thus, for a down-sample ratio of 1000, one out of every 1000 items was included.

**Weighting:**

Lastly, each normalized feature column was multiplied by a weighting value. This was done to give higher-weighted features more significance than others in the clustering algorithm. Since the actual significance of each feature in influencing the perceived taste for consumers was hard to quantify, empirical values were determined by qualitatively assessing the quality of clustering. Weightings were assessed on a small subset of data, and some were included in Appendix II as examples. The final selected values were summarized in Table 3. Note that the features pertaining to Short Name were excluded because they are roughly analogous to “correct” clusters and may have artificially simplified the clustering process or given expected rather than unusual results.

**Table 3: Selected Weighting Values**

|  |  |
| --- | --- |
| **Feature Type** | **Feature weighting** |
| Numerical Metrics (og, fgmin, srm, abv, ibu) | 1 |
| Styles | 0.015 |
| Short Names | 0 |
| Relevant Description Terms | 0.015 |

## Hierarchical Clustering:

The following section describes the clustering algorithms developed for this data.

**Approach:**

A bottom-up clustering approach was used to cluster the data hierarchically. This approach was inspired by Kruskal’s Algorithm[[5]](#footnote-6). To begin, every item was made a unique node. (Here, the term node is recursively defined as any single item, or a combination of two constituent nodes). The similarity between each pair of nodes was calculated and stored as describe next. The two most similar nodes were grouped into a single node, and the features of this node were set to the centroid of the constituent nodes as described later. The similarity of this node to each other node was calculated, and the new node’s two constituent nodes were removed from further consideration. Then, the next new pair of most similar nodes was selected and grouped into a single node. This process was iteratively repeated until only one node remained, the “root” of the hierarchical clustering tree (dendrogram).

**Similarity:**

Euclidean distance was used as the metric to appraise similarity between items. By convention, the distance from every node to itself was set to infinity for each set of items (i,j) with *F* features, the distance between items was calculated as:

Then, the set of most similar clusters was determined by finding the pair (i,j) with the lowest Euclidean distance.

**Centroid Calculation:**

The centroid was calculated as the average of the centroids of the two constituent nodes, each weighted by the number of items in each of these nodes. Thus, the value for each feature *fnew* for the new node was calculated as shown below. In this equation, nr corresponds to the number of items in the right constituent node (directions assigned arbitrarily) and fr corresponds to the value of feature *f* in the right constituent node, and likewise for fL and nL.

**Data Representation:**

The following objects were used to implement this algorithm. The total number of features is represented by *f,* the total number of nodes with more than one item is represented by *n,* and the number of original items is represented by *i* :

* ***x*** – an (*i+n*) x *f* matrix containing the feature values for every item. As new nodes were created, their features (which were the average features of their constituent nodes) were added as new rows to *x.*
* ***dist***– an (*i*+*n*) x (*i*+*n*) matrix containing the distance between items *k* and *j* for every position (*k,j*) in the matrix. The distance of (*k,k*) was set to infinity for all *k*.
* ***active***– a list of the indices corresponding to active items. Initially, the indices of all items (nodes with one item in them) were added to *active*. Once a set of nodes was selected for clustering, their indices were removed from *active* and the new index of the resulting, new node was added to *active*.
* ***node\_list***– a list of tuples which represented the structure of the hierarchy. Each tuple corresponded to one node added to the hierarchy and contained the following values:
  + constituent node 1
  + constituent node 2
  + distance between constituent nodes
  + number of items in node

**Pseudocode:**

The following pseudocode describes the process for hierarchical clustering on a dataset of *i* items and *f* features. This algorithm was implemented in Python using NumPy matrices and lists as primary data structures. The unabridged Python function for this algorithm was included in the file “Utilfn.py”.

)

## Balanced Hierarchical Clustering

The algorithm described above yields a hierarchical structure of flexible depth suitable for representation of many types of data clusterings. However, in some cases, the algorithm results in a deeply nested series of clusters. In this case, much of the intuition about the structure of the data is lost by the representation. To avoid this issue, a “balanced” approach was developed and implemented.

**Approach:**

The balanced clustering algorithm was largely the same as the unbalanced version, with one notable difference: rather than immediately appending the index of new nodes to the *active* list of indices, the indices of new nodes were added to a second list, *next­\_active.* Then, when the size of *active* reached 1 or 0, the indices in *next­­\_active­* were added to *active*, thus making the new nodes available for consideration again. Thus, at each step, every node is added to a new cluster before any of the new clusters are added to a new cluster again. This results in a balanced hierarchy.

**Pseudocode:**

The following pseudocode describes the process for balanced hierarchical clustering on a dataset of *i* items and *f* features. The unabridged Python function for this algorithm was included in the file “Utilfn.py”.

)

## Additional Major Functions:

The following section describes the major functions implemented or used from existing packages. The full list of implemented utility functions can be found in “Utilfn.py”.

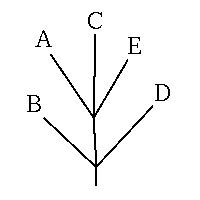
**Dendrogram plotting:**

Since the focus of this project was on the clustering algorithm, an existing dendrogram plotting library was used for dendrogram structure visualization. Scipy’s dendrogram module was used for this purpose[[6]](#footnote-7). (Note that though this module also can generate hierarchical clusterings, this functionality was not used nor referenced prior to the implementation of the algorithms described in this paper.)

**Conversion to Newick Tree Format:**

Newick tree format is a commonly-used form for expressing tree formats, especially for use in phylogenetic work[[7]](#footnote-8). In this format, each node is expressed in the following form. dn represents the distance between the node and its parent node, and each subnode, if any, is a recursive Newick tree.

An example tree was included in Figure 1. The Newick equivalent for this tree is: (B:1,(A:1,C:1,E:1)x:1, D:1)y:0; . All distances from parent nodes are assumed to be one except for at the root node, and names for branch nodes are arbitrarily assigned (x and y).



**Figure 1: Newick Tree Example**

Figure from: http://evolution.genetics.washington.edu/phylip/newicktree.html

The Newick tree structure was necessary for working with the ETE3 tree plotting package. Conversion to Newick tree format was implemented as a recursive algorithm which was included in “Utilfn.py”.

**Depth-Bounded Newick Tree Format:**

It was desired to have an interactive level of depth in the hierarchical structure, and this was easiest to implement in the Newick Tree format. Thus, a depth-bounded Newick conversion algorithm was also implemented. The major change from the algorithm briefly described above was as follows: the depth bound for each Newick tree or subtree was input to the algorithm, and every recursive call to construct a subtree was assigned a depth bound one less than its parent. When a subtree was to be generated with depth bound 0, all items contained in any of its subtrees were identified and made direct children of that node. The implementation for depth-bounded conversion to Newick tree format was included in “Utilfn.py”.

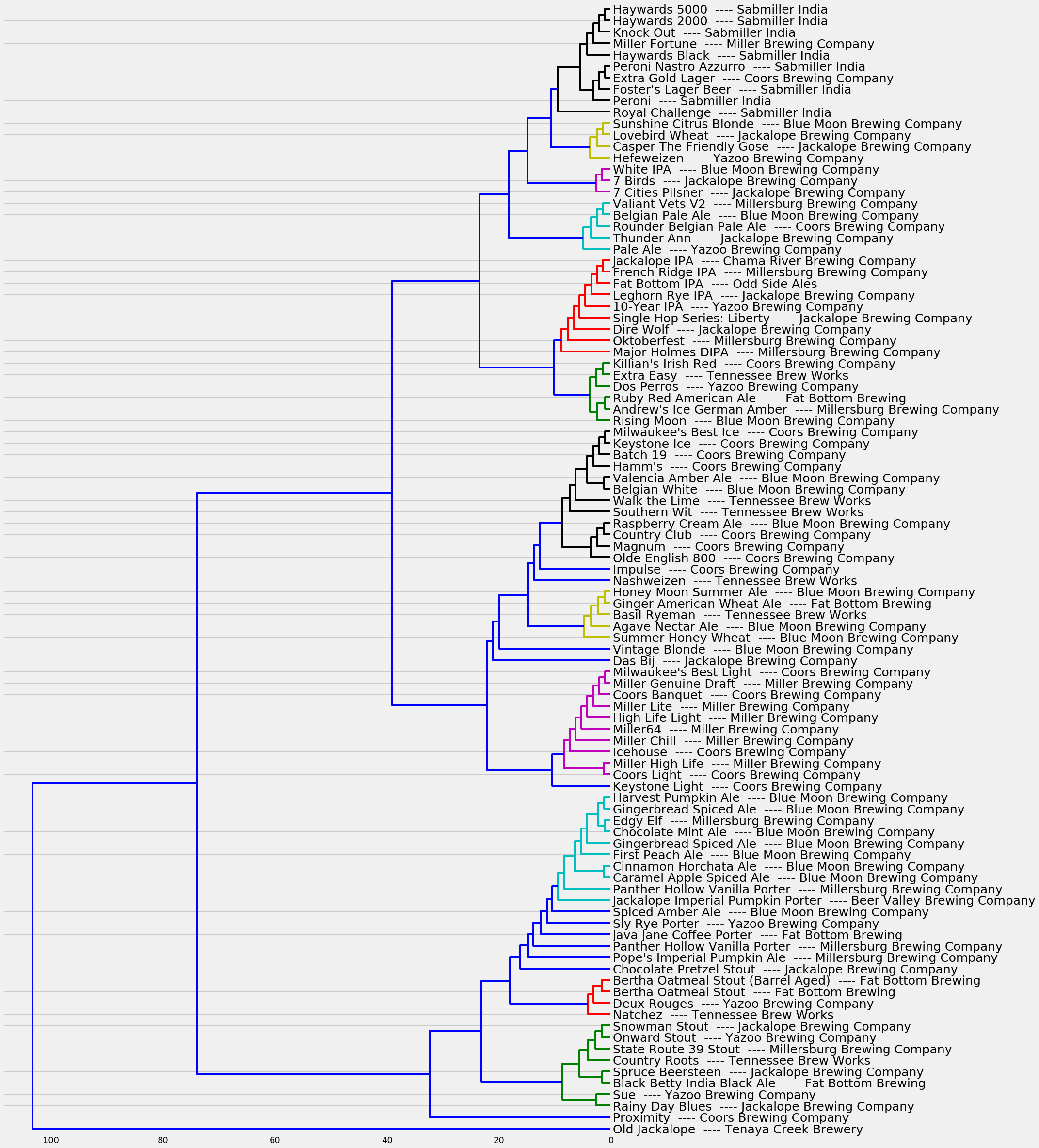
**Plotting of Newick Trees:**

Plotting of Newick Trees was accomplished using the ETE3 tree visualization package7,[[8]](#footnote-9). This allowed for an interactive, neat visualization of trees of various user-specified depths of branching.

## Results

**Hierarchical Clustering with 100 Items:**

To qualitatively assess the quality of clustering, a sample plot was generated with 100 items and was shown in Figure 2. The distance from the vertical bar of a node (split) to the vertical bar of its closest subnode to the right represents the weighted feature Euclidean distance between these nodes (this structure was necessary to prevent overlapping nodes). In general, though, taller branches indicated less similar subgroups than shorter branches. Distance could only be regarded relative to other distances. For instance, the Haywards Knockout2000 and the Haywards Knockout5000 (near top) are more similar than the Sue and Rainy Day Blues (near bottom). Colors were assigned to all subgroups with total distance less than a constant value which was empirically determined to make human-understandable categories such as Pilsners, IPAs and Stouts most evident. While larger hierarchical clusterings could be generated, the time complexity increased with the square of the number of items, and plots were hard to interpret above 500 or so items.

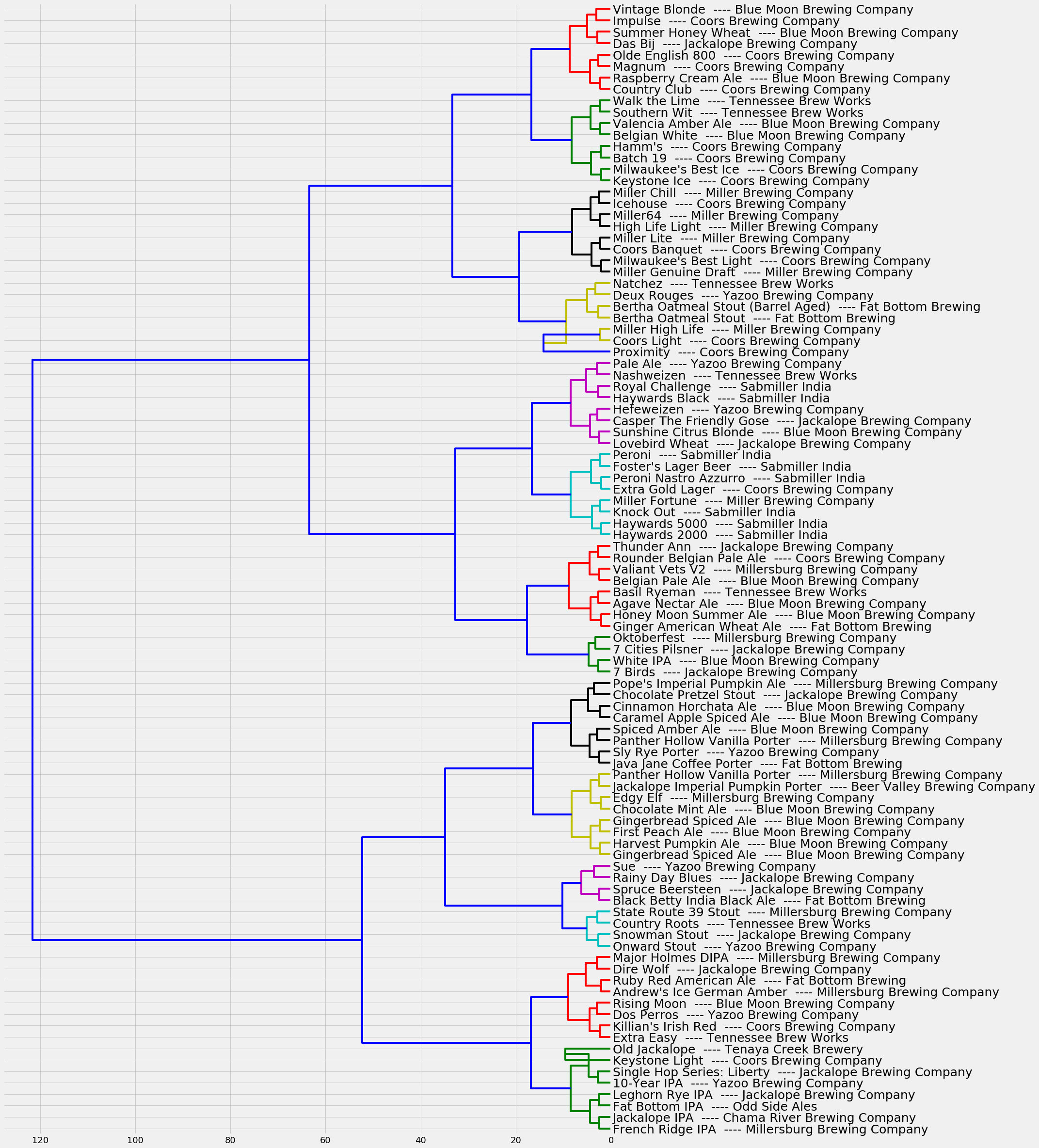


**Figure 2: Hierarchical Clustering with 100 Beers**

In general, this hierarchical structure was found to match fairly closely with expected categories. For instance, many light Pilsners were grouped together (magenta). However, some items were in unexpected clusters, and some unexpected groups relatively close to one another in the hierarchy (spiced ales and porters were grouped closely together, likely because these two groups are often flavored with the same spices and flavorings). In some cases, it is hard to tell whether these unexpected clusterings are mistakes (i.e. a human would find these beers very different in taste) or whether they are indicative of hidden flavor similarities.

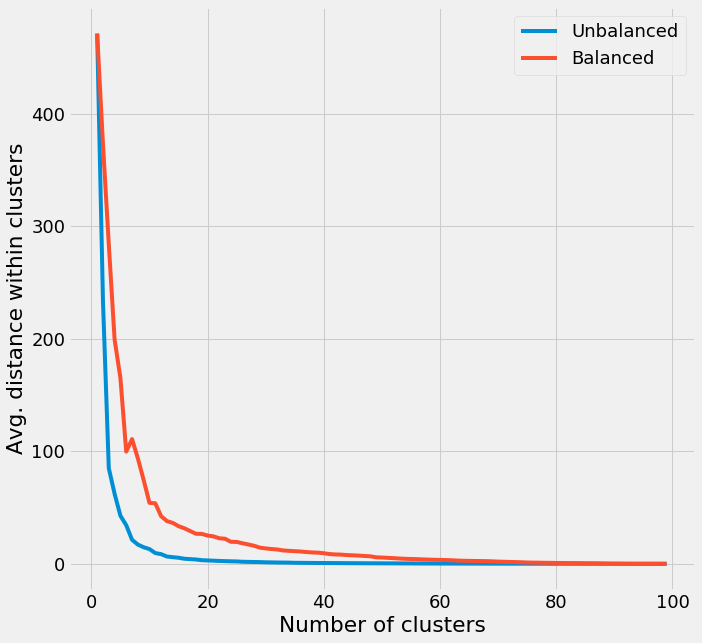
**Balanced versus Unbalanced Clustering:**

In Figure 3, the same items were clustered, this time using a balanced hierarchical clustering approach.



In general, items that were located close to one another in the unbalanced hierarchy were close to one another in the balanced hierarchy as well. The most notable difference is that the balanced hierarchy has no “stacks” of nested branches as are present in the unbalanced tree. Resultantly, though, the average distance between clusters was slightly higher.

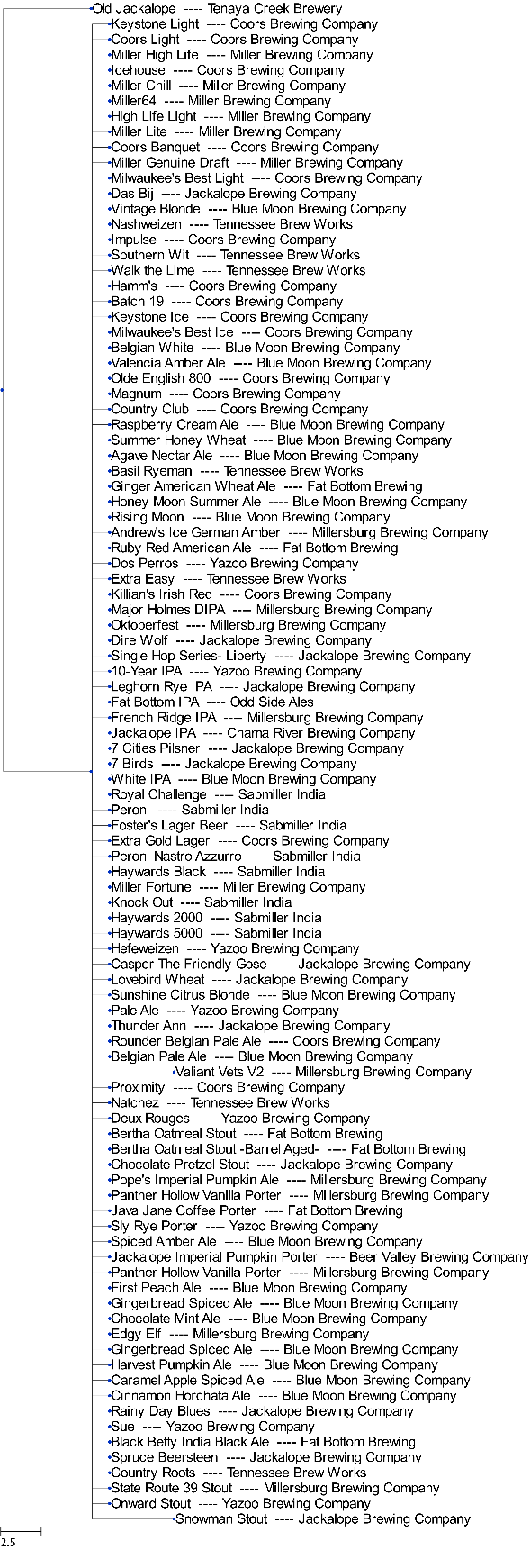
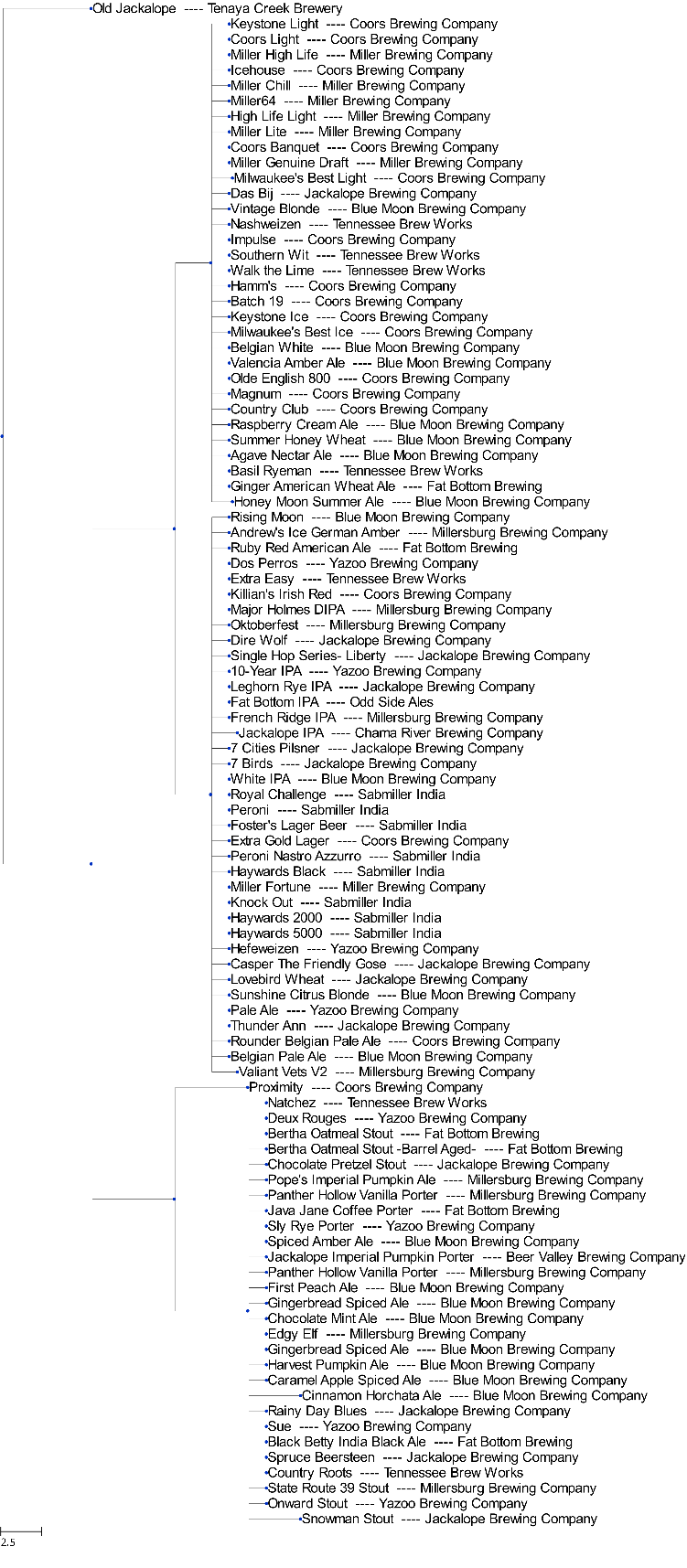
In Figure 4, the average distance of each item in a cluster to the cluster centroid was plotted as a function of the number of clusters for the balanced and unbalanced hierarchical clustering algorithm. For any number of clusters, this average distance was lower for the unbalanced than balanced algorithm, and the intra-cluster distance leveled off faster for the unbalanced than the balanced algorithm. This was because while both algorithms were greedy, the unbalanced algorithm was greedy over all possible next node additions, while the balanced algorithm was only greedy over a subset. Note that neither algorithm gave the optimal hierarchical structure; in practice, finding this structure for large sets of data would be NP-hard.



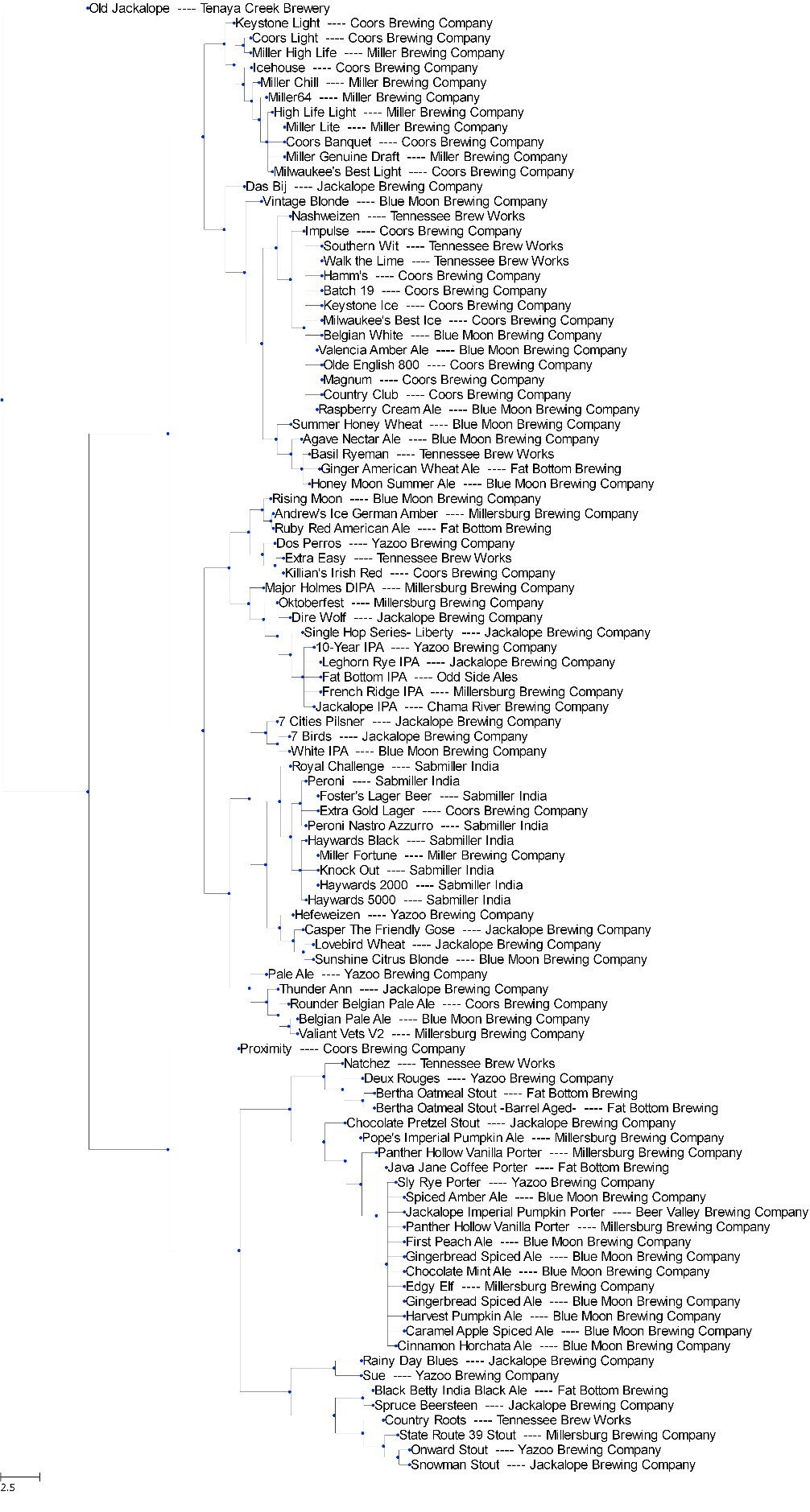
**Figure 4: Intra-cluster Distance to Centroid versus Cluster Count**

**Depth-Bounded Trees:**

The same data was plotted in using the Newick tree format and ETE3 plotting package to a variety of degrees of depth in Figures 5 and 6.



**Figure 5: Depth-bounded Hierarchy (Bound = 1, Bound = 4)**



**Figure 6: Depth-Bounded Hierarchy (Depth = 10)**

## Discussion

The clustering algorithms implemented in this work showed hierarchical clusterings that were intuitively fairly accurate. However, there are a few notable limitations to this implementation. First, relevant descriptive terms were extracted as 1-grams from the descriptions of each beer, and relevant labels were selected manually. This is a crude simplification of a rather complex natural-language processing task. Ideally, the parsing of descriptions would take the context of each term into consideration, group similar terms as a single concept, and automatically extract terms relevant to taste, color, and aroma based on knowledge extraction from input or web sources. This presented a formidable challenge for a project based primarily on clustering, so the simplified approach used in this work was adopted instead.

Next, weightings were somewhat arbitrarily assigned to feature categories. In practice, the optimal weightings are impossible to know exactly but would likely vary at least somewhat based on personal taste. The core issue is that while this problem was treated as an unsupervised learning clustering task, in fact the items are not unlabeled, because the perceived flavors of each beer place it fuzzily into some sort of category. In other words, while the data is not fully labeled, there is some association with various categories. An out-of-box classification algorithm naively assumes that each feature has equal importance in clustering data because it cannot know which items should be clustered closely and the feature weightings that would be necessary to produce this result.

One potential solution to this problem would be to partially “train” the clustering algorithm with input triplets (item 1, item 2, item 3). For each input triplet, the algorithm would then look for weightings that constrained item 1 to be more similar to item 2 than item 3, thus training the unsupervised learning algorithm on which features were important for grouping the data into clusters that roughly correspond to the groups a human might classify them into. It is unclear how many such training inputs would be necessary to tune the weightings, but in theory, the resulting feature weightings would be optimized to make hierarchical distinctions based on the personal taste preferences of the user. This application and the area of trained feature weightings for clustering in general are interesting fields.

## Appendix I: Style and Short Name Values

**Table A1: Style Values**

|  |
| --- |
| Belgian And French Origin Ales |
| British Origin Ales |
| European-germanic Lager |
| German Origin Ales |
| Hybrid/mixed Beer |
| International Ale Styles |
| International Styles |
| Irish Origin Ales |
| Mead, Cider, & Perry |
| North American Lager |
| North American Origin Ales |
| Other Lager |
| Other Origin |

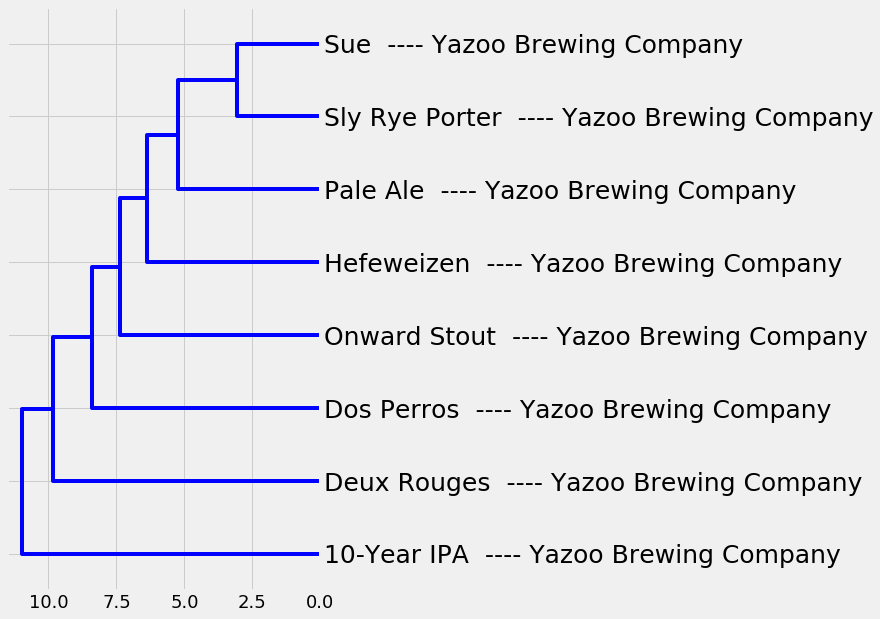
**Table A2: Short Name Values**

|  |
| --- |
| Adambier |
| Altbier |
| Amber |
| American Amber Lager |
| American Barleywine |
| American Brown |
| American Dark Lager |
| American IPA |
| American Ice Lager |
| American Imperial Porter |
| American Imperial Stout |
| American Lager |
| American Light Lager |
| American Low-Carb Lager |
| American Oktoberfest |
| American Pale |
| American Pilsener |
| American Premium Lager |
| American Stout |
| American Strong Pale |
| Austrailian Pale |
| Baltic Porter |
| Belgian Blonde |
| Belgian Dark Strong |
| Belgian Dubbel |
| Belgian Pale |
| Belgian Pale Strong |
| Belgian Quad |
| Belgian Table Beer |
| Belgian Tripel |
| Berlinerweisse |
| Bernsteinfarbenesweizen |
| Bitter |
| BiÃ¨re de Garde |
| Black Ale |
| Blonde |
| Bock |
| Bock Rauchbier |
| Bohemian Pilsener |
| British Barleywine |
| British Imperial Stout |
| Brown Porter |
| California Common |
| Chocolate Beer |
| Coffee Beer |
| Cream Ale |
| Dark Wheat Ale |
| Doppelbock |
| Dortmunder |
| Dry Irish Stout |
| Dry Lager |
| Dunkelweizen |
| ESB |
| English Brown |
| English Dark Mild |
| English IPA |
| English Pale |
| English Pale Mild |
| English Summer Ale |
| Euro Dark |
| Export Stout |
| Field Beer |
| Flanders Red |
| Fruit Beer |
| Fruit Cider |
| Fruit Lambic |
| Fruit Wheat Ale |
| German Pilsener |
| German Rye |
| Gingo/Sake |
| Grodziskie |
| Gueuze |
| Hefeweizen |
| Helles |
| Helles Rauchbier |
| Honey Beer |
| Imperial IPA |
| Imperial Red |
| International Pale |
| International Pilsener |
| Irish Red |
| Kristallweizen |
| Kolsch |
| Lambic |
| Leichtbier |
| Leichtesweizen |
| Leipzig Gose |
| Maibock |
| Malt Liquor |
| MÃ¤rzen |
| MÃ¤rzen Rauchbier |
| Oatmeal Stout |
| Oktoberfest |
| Old Ale |
| Pumpkin Beer |
| Rauchbier |
| Robust Porter |
| Saison |
| Schwarzbier |
| Scotch Ale |
| Scottish Export |
| Scottish Heavy |
| Scottish Light |
| Session |
| Smoke Porter |
| Special Bitter |
| Specialty |
| Spice Beer |
| Stout |
| Strong Ale |
| Sweet Stout |
| Tropical Light Lager |
| Vienna Lager |
| Weizenbock |
| Wheat Ale |
| Wheatwine |
| Witbier |

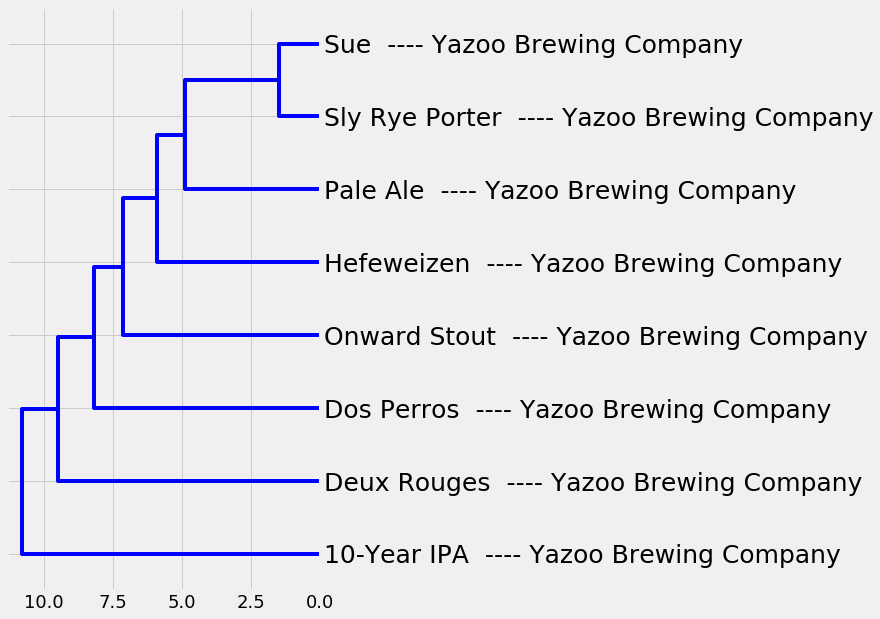
## Appendix II: Trial Weightings for Feature Types

## 

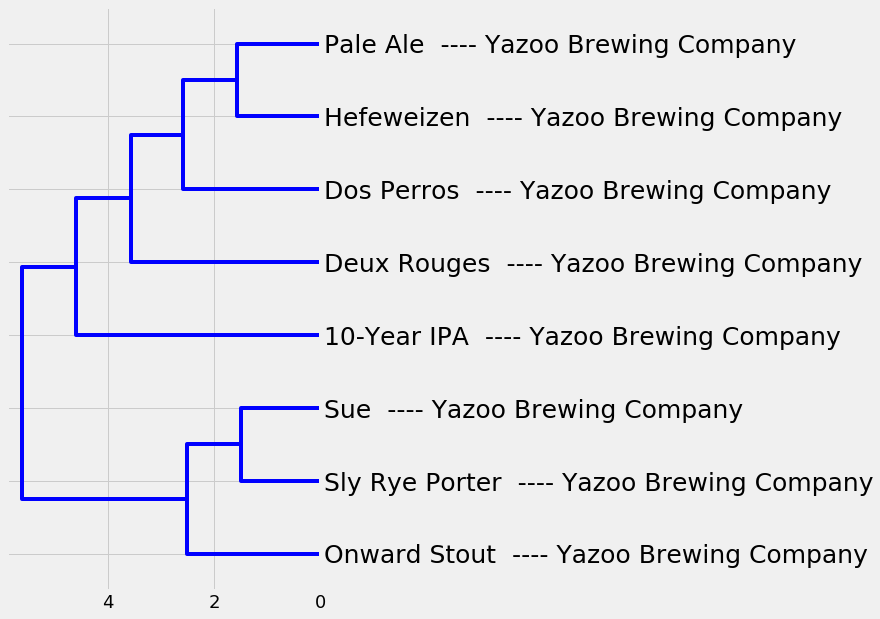
**Numerical – 1, Style – 0.015, Short Name – 0, Description Terms – 0.015**



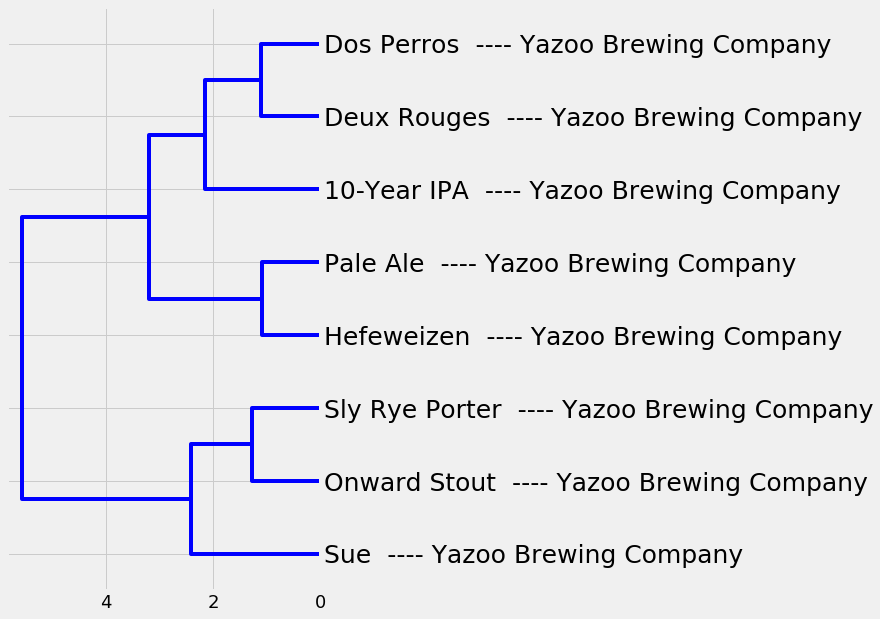
**Numerical – 1, Style – 1, Short Name – 1, Description Terms – 1**



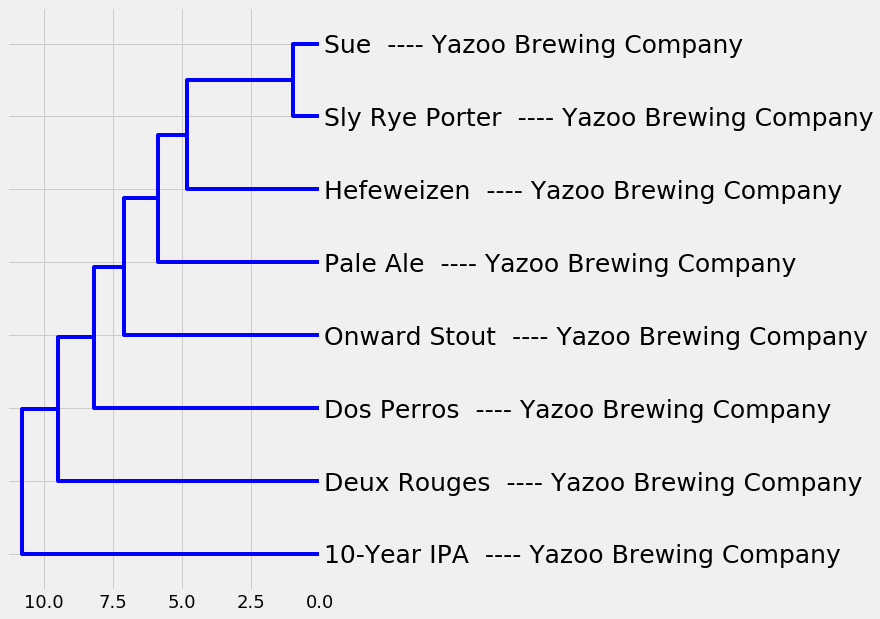
**Numerical – 1, Style – 0, Short Name – 0, Description Terms – 1**



**Numerical – 1, Style – 0, Short Name – 0, Description Terms – 0.015**



**Numerical – 1, Style – 0, Short Name – 0, Description Terms – 0**



**Numerical – 0, Style – 0, Short Name – 0, Description Terms – 1**

1. https://github.com/raybuhr/w210\_beer\_demand [↑](#footnote-ref-2)
2. https://www.brewerydb.com/ [↑](#footnote-ref-3)
3. https://kaggle.com/nickhould/craft-cans#beers.csv [↑](#footnote-ref-4)
4. https://data.opendatasoft.com/explore/dataset/open-beer-database [↑](#footnote-ref-5)
5. Greenberg, H. J. (1998). Greedy algorithms for minimum spanning tree. University of Colorado at Denver. Available at: https://glossary.informs.org/notes/spanningtree.pdf [↑](#footnote-ref-6)
6. https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.dendrogram.html [↑](#footnote-ref-7)
7. Huerta-Cepas, J., Dopazo, J., & Gabaldón, T. (2010). ETE: a python Environment for Tree Exploration. BMC bioinformatics, 11(1), 24. Available at: https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-

   11-24 [↑](#footnote-ref-8)
8. http://etetoolkit.org/ [↑](#footnote-ref-9)