Evaluating K-means on Newsgroup data

Marius Maaland
maaland@uw.edu

Jonas Palm
kjpalm@uw.edu

Abstract

Document clustering is a way of analyzing and categorizing textual documents. In this report we compare different variations of K-means (regular K-Means, K-Means++ and MiniBatch K-Means). We show that by using MiniBatch K-means we can decrease runtime substanstially by only sacrificing a small amount of accuracy. We also show that K-means performs well compared to other clustering techniques (namely Birch and HAC). On the newsgroups dataset we didn't see any big improvements from using K-Means++ initialization over regular K-means initialization.

1 Introduction

This report explores different clustering algorithms using the 20 Newsgroups dataset. The 20 Newsgroups dataset is a collection of roughly 20,000 documents collected from 20 different newsgroups [1]. We have split this report into two parts, feature selection and document clustering. The feature selection chapter mainly covers how we decided on which set of features to use and how we decided to preprocess those. The clustering part is mainly an exploration of how different K-means variations perform on the dataset, but also gives a brief insight on how K-means in general compares to other clustering algorithms. In this report we try to answer the following questions (note that the answers will be based on evaluation on the Newsgroup dataset and may be different in other scenarios).

- Which set of features gives the best representation of the dataset? We explore term count (TC) and term frequency - inverted document frequency (TF-IDF) with different n-gram combinations.
- How does K-Means++ affect the performance compared to regular K-Means initialization?
- How does MiniBatch K-Means (MB K-Means) work and what are the advantages / disadvantages of using it over regular K-Means?
- How does K-Means compare to other clustering algorithms, namely Birch and Hierarchical Agglomerative Clustering (HAC).

2 Feature selection

In this project we define feature selection to mean the process to select which **set** of features to use. A set of features in our case is some n-gram combination of either term count or tf-idf (term frequency - inverse document frequency) weights. We do this by training multiple logistic regression classifiers[2] each with one set of features and then select the set which yields the highest accuracy. When using n-grams, a larger n means the classifier becomes more context sensitive (as each term contains more information about the context it appears in), if $n = 1$ each feature is a single word, if

[1] http://qwone.com/ jason/20Newsgroups/

[2] The TA who reviewed our project proposal told us to do this. This is a footnote to remind David Wadden of this fact, as per discussion by e-mail.

$n = 2$ each term consists of two following words from the original document and so on. Notably, if we increase n we simultaneously decrease the "reach" of each term, in other words, a single word will appear more times in all documents than any n-word combination that contains that single word.

2.1 Preprocessing

In order to remove noise or terms that might trick our classifiers and clustering algorithms we also preprocess the documents. The reason for this is that we don't want to cluster or classify based on irrelevant features (like make a decision based on whether the document is written by a certain author). To visualize the effects of preprocessing we used PCA on a subset of the newsgroups (Figure 2). The preprocessing steps used were:

1. Remove English stop words (common words).
2. Remove headers containing subject-fields, "from"-fields etc.
3. Remove email addresses.
4. Remove quotes.

Figure 1: Result of doing feature selection using Logistic regression. The left image is before preprocessing and the right image is after. The training accuracy is indicated by a dashed line. cv-ng are the term count n-grams and tf-idf-ng are the tf-idf n-grams.

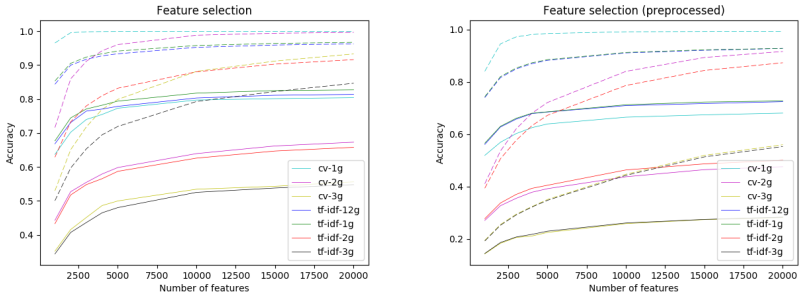
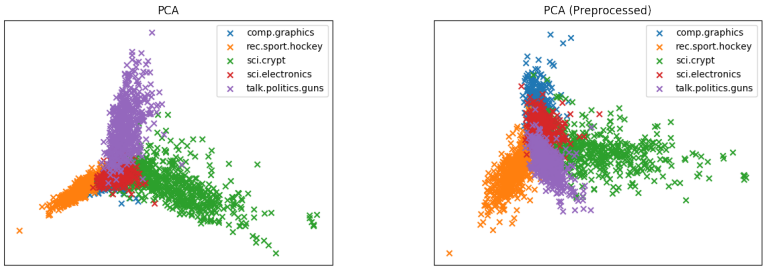


Figure 2: Visualization of 5 newsgroups using PCA before and after preprocessing. After preprocessing, comp.graphics and sci.electronics are shown as more "separable".



2.2 A note on implementation

To do feature selection we used (Multiclass) Logistic Regression classifiers and PCA implementation from SciKit-Learn. The feature vectors were created by CountVectorizer and TfidfVectorizer (also from SciKit-learn). We implemented a dataset library in Python that allowed us to download and fetch categories and labels from the Newsgroup dataset and also all preprocessing steps (except the removal of stop words, which was provided by the vectorizers).

2.3 Result and analysis of feature selection

We trained seven logistic regression classifiers using tf-idf and term count weights with n-grams of sizes $n = 1, 2, 3$. We also included a tf-idf vector with the n-gram range 1-2 ($n = (1, 2)$, i.e. a combination of both unigrams and bigrams. We see in Figure 1 that without preprocessing we quickly overfit (training accuracy approaches 1) for low values of n. The best accuracy was achieved when using tf-idf weights with unigrams or a combination of unigrams/bigrams (labeled as tf-idf-12g in the figure). However, we noted that the words with the highest weights (i.e. the words that influence the classification the most) for the unigram/bigram-combination were almost exclusively unigrams. Therefore, we decided to use tf-idf weights with unigrams when moving on to step two of the project (clustering).

3 Implementation

3.1 Algorithms

We implemented MB K-Means and K-means++ in Python, with pseudocode shown below. MB K-Means is a more efficient variation of regular K-means. Regular K-means is expensive for large datasets, requiring $O(knsi)$ where k is the number of clusters, n is the number of examples, s is the maximum number of non-zero elements in any example vector and i is the maximum number of iterations (in our case 100, but we noticed that when running regular K-means it usually needed much fewer iterations to converge). MB K-Means improves on this by selecting smaller subsets (batches) of the dataset ($b = 100$ in our case) for each update step, with the motivation that these smaller subsets have lower stochastic noise than individual examples. Each sample in the batch is assigned a cluster, and the update step is a gradient descent update. Each batch update is then run t times ($t = 100$ in our case).

Algorithm 1 MB K-Means

Given: k, mini-batch size b, iterations t, data set X
Initalize each $c \in C$ with an x picked randomly from X
$\mathbf{v} \leftarrow 0$
for $i = 1$ *to* t **do**
    $M \leftarrow b$ examples picked randomly from X
    for $\mathbf{x} \in M$ **do**
        $\mathbf{d}[\mathbf{x}] \leftarrow f(C, x)$ *// Cache the center nearest to x*
    for $\mathbf{x} \in M$ **do**
        $c \leftarrow \mathbf{d}[\mathbf{x}]$ *// Get cached center for this x*
        $\mathbf{v}[\mathbf{c}] \leftarrow \mathbf{v}[\mathbf{c}] + 1$ *// Update per-center counts*
        $\eta \leftarrow \frac{1}{\mathbf{v}[\mathbf{c}]}$ *// Get per-center learning rate*
        $c \leftarrow (1 - \eta)\, c + \eta x$ *// Take the gradient step*

K-means++ is a variation of regular K-means that aims to choose the centers for the K-means algorithm in a more effective way than random, a way of avoiding the sometimes poor clusterings found by the standard K-means algorithm. It is described by the pseudocode below.

Algorithm 2 K-means++

Take one center c_1, chosen uniformly at random from X
Take a new center c_i, choosing $x \in X$ with probability $\frac{D(x)^2}{\sum_{x \in X} D(x)^2}$
Repeat step 2, until we have taken k centers altogether.
Proceed as with the standard k-means algorithm.

Two compare K-means to other clustering algorithms we selected Birch and HAC (for their simplicity to use) from the Scikit-Learn package.

3.2 Method

When performing clustering while increasing the number of true clusters (category subsets), we chose the same categories, in the same order, for every iteration, avoiding the factor of "luck". I.e. before each trial we shuffled the full list of categories and shuffled the order. Then for test k we used the k first categories in that list, which meant that test $k + 1$ contained all categories from test k plus one more.

To counter the problem of K-means ending up in local minimum, we used five restarts and selected the restart that yielded the lowest distortion (within group sum of squares).

When comparing Birch and HAC to K-means, we used the Scikit-Learn implementations of K-means, to achieve equal grounds of comparison with regards to computational efficiency.

4 Results of variations of K-means

Figure 3: Comparison of mistake rate using K-Means, K-means++, MB K-Means and MB K-Means++

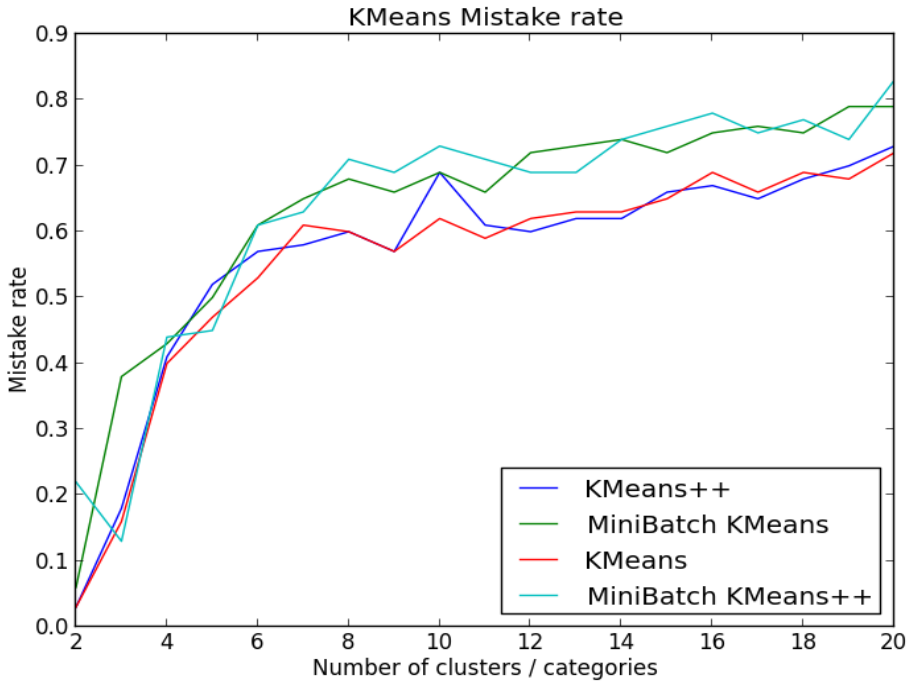


Figure 4: Comparison of computation time using K-Means, K-means++, MB K-Means and MB K-Means++

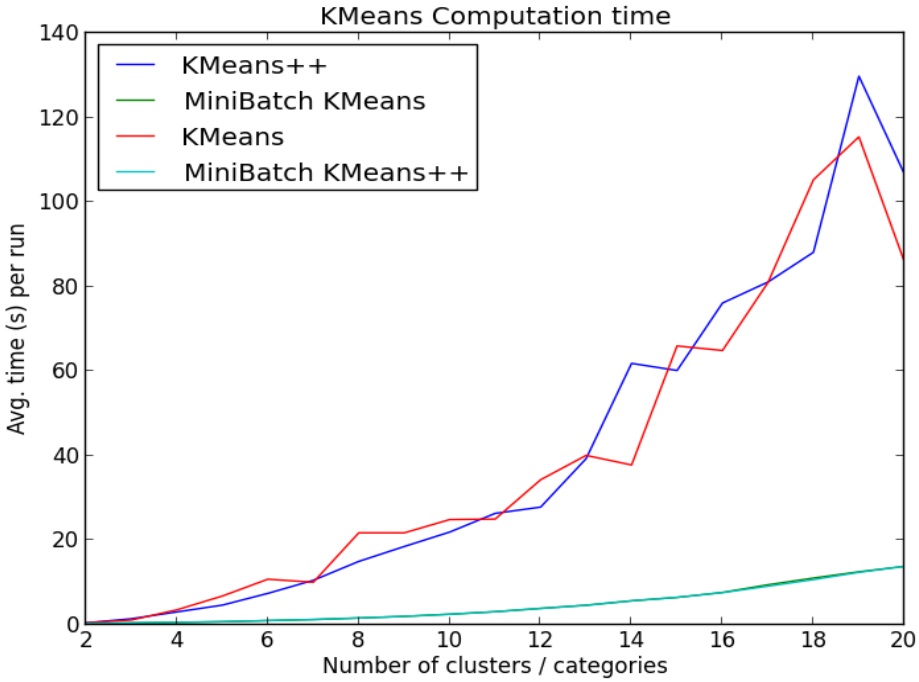
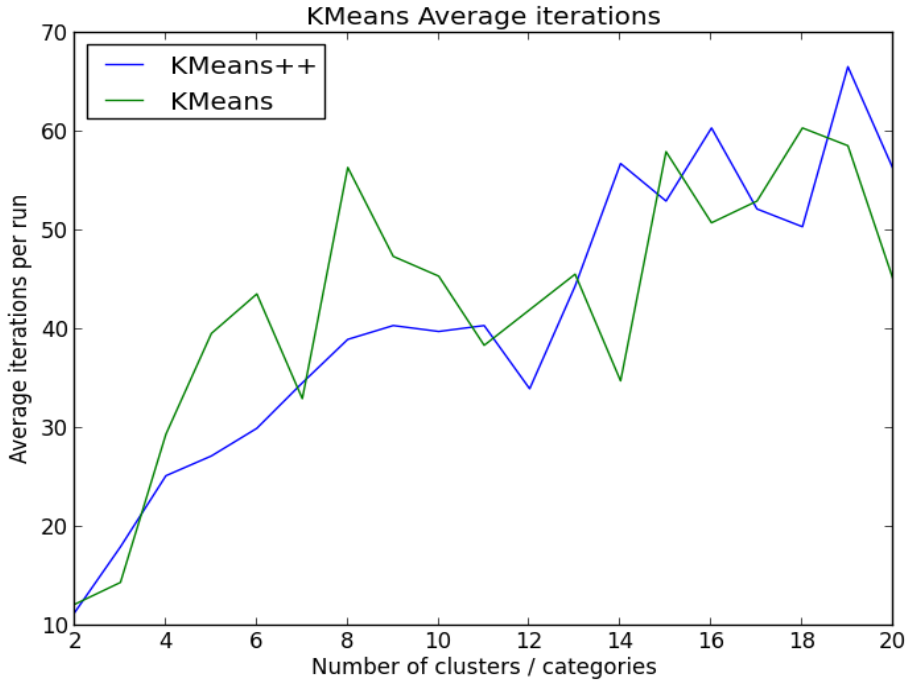


Figure 5: Comparison of average iterations per run, using K-means and K-means++



5 Results of comparison between K-means, MB K-Means, Birch and HAC

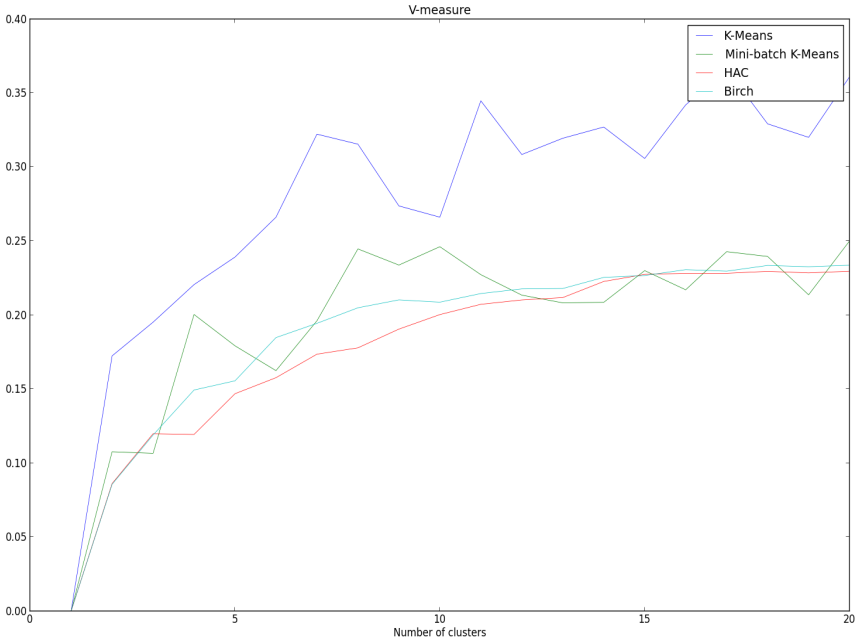
When assessing the performance of the different clustering algorithms compared to each other, we looked at V-measure score, the harmonic mean of $homogeneity$ and $completeness$. where a higher score is better.

$$\textit{V-measure} = \frac{2 * (homogeneity * completeness)}{(homogeneity + completeness)} \tag{1}$$

Where $homogeneity$ is defined as follows: "A clustering result satisfies homogeneity if all of its clusters contain only data points which are members of a single class." And $completeness$ is defined as follows: "A clustering result satisfies completeness if all the data points that are members of a given class are elements of the same cluster."

The reader can see, in figure 6, that K-means received the highest V-measure score when compared to MB K-Means, Birch and HAC. The remaining 3 algorithms received a near identical score, although they have different computation times, as shown in figure 4

Figure 6: Comparison of mistake rate using K-Means, K-means++, MB K-Means and MB K-Means++



6 Conclusion

The reason for using K-means++ is to improve the speed and accuracy of regular K-means initialization. In our experiments the difference was hardly noticable. One could argue that in figure 5 the curve for K-means++ is smoother for $k < 12$, which could indicate that the average number of iterations needed is more predictable. However, how well K-means++ will work is highly dependent on the dataset (as shown in the report by David Arthur and Sergei Vassilvitskii). Furthermore, one disadvantage of using K-means++ is the risk of selecting outliers as centers, since selecting a data point further away from the current centers has a higher probability.

As shown in figure 4 and 3, the runtime of MB K-Means is significantly shorter than regular K-means, with only a small sacrifice in accuracy. Depending on the use case this small sacrifice might be worth the increase in speed.

K-means is shown to outperform the other algorithms we used in the project (Figure 6). However, our knowledge of Birch and HAC is very limited but it is still interesting to see that K-means works really well for being a relatively easy algorithm (intuitively and implementation wise).

References

[1] D. Sculley, Google, Inc. Pittsburgh. PA USA. *Web-Scale K-Means Clustering.*

[2] David Arthur and Sergei Vassilvitskii. *k-means++: The Advantages of Careful Seeding.*

[3] Scikit-Learn Clustering. *http://scikit-learn.org/stable/modules/classes.html#module-sklearn.cluster*

[4] Scikit-Learn V-Measure. *http://scikit-learn.org/stable/modules/generated/sklearn.metrics.v_measure_score.html*