Improving k-means Clustering with Genetic Programming for Feature Construction

Andrew Lensen
School of Engineering and Computer Science
Victoria University of Wellington
PO Box 600
Wellington 6140, New Zealand
andrew.lensen@ecs.vuw.ac.nz

Bing Xue
School of Engineering and Computer Science
Victoria University of Wellington
PO Box 600
Wellington 6140, New Zealand
bing.xue@ecs.vuw.ac.nz

Mengjie Zhang
School of Engineering and Computer Science
Victoria University of Wellington
PO Box 600
Wellington 6140, New Zealand
mengjie.zhang@ecs.vuw.ac.nz

ABSTRACT

k-means is one of the most commonly used clustering algorithms in data mining. Despite this, it has a number of fundamental limitations which prevent it from performing effectively on large or otherwise difficult datasets. A common technique to improve performance of data mining algorithms is feature construction, a technique which combines features together to produce more powerful constructed features that can improve the performance of a given algorithm. Genetic Programming (GP) has been used for feature construction very successfully, due to its program-like structure. This paper proposes two representations for using GP to perform feature construction to improve the performance of k-means, using a wrapper approach. Our results show significant improvements in performance compared to k-means using all original features across six difficult datasets.

CCS CONCEPTS

•**Computing methodologies → Genetic programming;** *Cluster analysis; Feature selection;*

KEYWORDS

Cluster Analysis; Feature Construction; Genetic Programming; k-means; Evolutionary Computation

ACM Reference format:
Andrew Lensen, Bing Xue, and Mengjie Zhang. 2017. Improving k-means Clustering with Genetic Programming for Feature Construction. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 2 pages.
DOI: http://dx.doi.org/10.1145/3067695.3075962

1 PROPOSED METHOD

GP has been widely used to perform feature construction (FC) in classification tasks, but little work has applied GP to FC in clustering tasks. This work proposes a new approach for performing FC using a GP wrapper approach for clustering.

In the majority of GP work, each individual contains a single tree which outputs a single value (i.e. a single constructed feature). More than one constructed feature is needed to split a dataset into a large number of clusters effectively. In the following subsections we propose two representations for constructing multiple features in a single GP individual. In both representations, only the constructed features produced by an individual are used by the k-means algorithm.

1.1 Multi-Tree Representation

In a multi-tree representation, each individual contains multiple trees, each of which produces a single constructed feature. The function set contains standard arithmetic operators ($+, -, \times, \div, |+|, |-|$) as well as *max* and *min* functions. Each operator takes two inputs and produces a single output. $\div$ is protected division; it returns 1 if the divisor is 0. An if function is also used, which takes three inputs and returns the second input if the first input is positive, or the third input otherwise. The terminal set comprises of the features of the dataset, and a random double terminal drawing from the range $[0, 1]$

To perform crossover between two individuals, a random tree is selected from each individual, and then a random sub-tree from each selected tree. Crossover is then performed between the subtrees as normal. Mutation is performed by choosing a random tree in an individual, and then performing mutation as normal.

This approach requires the number of trees, t, to be set in advance and uses multi-tree crossover, which can decrease the efficacy of the crossover process. The following section proposes an alternative approach where only a single tree per individual is used.

1.2 Vector Representation

To produce multiple constructed features from a single tree, we propose using a vector representation, where a tree outputs a vector of constructed features.

The function set used in this approach is similar to that of the multi-tree approach; however, each function is altered to instead take two *vectors* as input and produce a single output vector. Each function node applies its function pair-wise to each element of the input vectors, producing an output vector equal in length to the shorter of the two input vectors. In addition, a *concat* function is used to allow vectors of varying length to be automatically constructed, which allows GP to automatically construct an appropriate number of features. This function takes two vector inputs and produces an output equal to appending the second vector to the end of the first. The terminal set is unchanged from the multi-tree approach, with each terminal outputting a vector of length one.

1.3 Fitness Function

One of the most popular fitness functions in clustering literature minimises the intra-cluster variation [1] (i.e. the distance between instances in the same cluster), as shown below:

$$\sum \text{Intra} = \sum_{i=1}^{K} \sum_{I_a \in C_i} d(I_a, Z_i) \tag{1}$$

where C_i is the i^{th} cluster, $I_a \in C_i$ is an instance in the i^{th} cluster and Z_i is the mean of the i^{th} cluster. Minimising this function encourages compact clusters, but is biased towards hyper-spherical clusters as distance is minimised to the cluster mean.

An alternative approach is to instead use a fitness function which considers connectedness. Connectedness measures how well instances which are close to each other are allocated to the same cluster; if instances are similar, they should be in the same cluster. The mean connectedness can be computed as follows:

$$\text{Connectedness} = \frac{1}{K} \sum_{i=1}^{K} \frac{1}{|C_i|} \sum_{I_a, I_b \in C_i} d_{inverse}(I_a, I_b) \tag{2}$$

$$d_{inverse}(I_a, I_b) = min\Big[\frac{1}{d(I_a, I_b)}, 10\Big] \tag{3}$$

Maximising the above fitness function encourages clusters to contain instances that are close together.

2 EXPERIMENT DESIGN

Six difficult synthetic datasets were used to evaluate the proposed methods, chosen from the widely used datasets provided by Handl et al. [2]. Three of these datasets (50d10c, 50d20c, 50d40c) contain 50 features; 10,20, and 40 classes; and 2699, 1255, and 2335 instances respectively. The remaining three datasets (100d10c, 100d20c, 100d40c) contain 100 features; 10, 20, and 40 classes; and 2893, 1339, and 2212 instances respectively. The six datasets were scaled so features had values in $[0, 1]$ to prevent feature range from biasing training. A range of synthetic and real-world datasets were used to comprehensively evaluate the proposed methods. Datasets were scaled so that each feature had values between zero and one, to prevent bias towards features with large ranges.

Each of the two proposed representations were tested with each of the two fitness functions on the above six datasets. k-means using all of the original feature set was also tested on the six datasets as a baseline. Each method was run 30 times using different seeds to account for the stochastic nature of GP and k-means. Standard GP parameters were used, with a population size of 1,024, crossover and elitism rates of 80% and 20%, and top-10 elitism. $t = 7$ is used for the multi-tree approach. Each method is run for 100 iterations.

We evaluate each method using a variation of the F-measure, which measures how well the clusters produced match the "gold-standard" reference clusters provided by the dataset authors. Each pair of instances in the dataset is considered in turn. If both instances are in the same reference cluster, then they are a true positive (TP) if they fall in the same produced cluster, and a false negative (FN) otherwise. If the two instances are in different reference clusters, they are a true negative (TN) if they fall in different produced clusters, and a false positive (FP) otherwise.

Table 1: F-measure performance on the datasets

Method	50d10c	50d20c	50d40c	100d10c	100d20c	100d40c
MTConn	0.5167^+	0.4996^+	0.4397^+	0.5311	0.4657^+	0.4629^+
MTIntra	0.4785^-	0.4776^+	0.4269^+	0.5825^+	0.4598^+	0.462^+
VectorConn	0.5005	0.4832^+	0.4106^+	0.5446	0.4451^+	0.4418^+
VectorIntra	0.4795^-	0.4351^+	0.3759^+	0.5854^+	0.4331^+	0.4028^+
k-means AF	0.4939	0.3823	0.2618	0.5255	0.3800	0.2675

3 RESULTS

Table 1 shows the mean F-measure performance of the four proposed GP methods and k-means (using all features (AF)) on the datasets described previously. The multi-tree approaches using connectedness and $\sum$ Intra are notated as MTConn and MTIntra. The vector approaches are notated as VectorConn and VectorIntra. Each GP result is labelled with a "+" or a "−" if it is significantly better or worse than the k-means baseline according to a Student's t-test performed with a 95% confidence interval.

The four GP methods are significantly better than normal k-means in 79% of results, and significantly worse in only 2 results, on the easiest 50d10c dataset. Normal k-means clearly performs poorly on the datasets with 20 or 40 clusters, where the dataset must be split into a large number of different groups. The GP methods are able to produce more powerful constructed features to overcome the limitations of k-means on these datasets.

The connectedness fitness measure is slightly better than $\sum$ Intra on most datasets with the exception of 100d10c. The multi-tree approach also generally outperforms the vector approach, again with the exception being the 100d10c dataset.

4 CONCLUSION

This work proposed two approaches to using GP for constructing multiple features on k-means using a wrapper technique. In addition, two potential fitness functions were tested to investigate how k-means could be improved by using alternative measures of cluster quality. All of the representations and fitness functions showed a significant improvement over normal k-means across a range of hard clustering datasets.

The proposed methods could be further extended in future work, by developing novel fitness functions to allow k-means to work well on a range of different types of datasets, and by extending the vector or multi-tree approaches to be used when K is not known in advance. It would also be useful to investigate a way of determining the number of trees in the multi-tree approach automatically.

REFERENCES

[1] Charu C. Aggarwal and Chandan K. Reddy (Eds.). 2014. *Data Clustering: Algorithms and Applications.* CRC Press.

[2] Julia Handl and Joshua D. Knowles. 2007. An Evolutionary Approach to Multiobjective Clustering. *IEEE Trans. Evolutionary Computation* 11, 1 (2007), 56–76.