Assigning Style Grade and Providing Style Feedback by k-means Clustering and Softmax Regression

Roman Roman, Homero
Stanford University
homero@stanford.edu

2016-06-06

1 Abstract

While identifying functionality mistakes in a program can be done with near certainty for a finite number of final requirements, identifying style mistakes is subject to subtle conventions and an infinite number of possible programs. This poses a problem for students trying to get feedback on their program style while they are still working on the assignment. Also it makes it harder for teacher assistants (TA's) to agree on a certain style grade. Therefore, the goal of this research is to figure out how to automatically assign a style grade to a program and provide style feedback. More specifically, the procedures employed are, first, k-means clustering of the data according to one of three different strategies, then, from each cluster fitting a logistic regression or naive Bayes model for classifying functions into those well decomposed and another for well formatted functions, and, finally, fitting a softmax multi-class classification model to assign a program style grade.

2 Introduction

Developing good programming style is a vital part of the CS106A class and to enforce this practice all assignments are graded not only on functionality but also on style. But what exactly constitutes "good" style is not very clear. As such, students who are just learning about programming struggle to develop this "good" style. Unlike functionality which they can test themselves by running their program on their computer, style is not given feedback until the program has been graded by the TA by which time it is too late to fix. During the developing stage of the program writing, the student's program is not necessarily functionally complete but we still wish to provide feedback on the style and format of the parts already written so that the student may back trace if necessary and rethink his current implementation. By providing both style and functionality feedback at every stage, the aim is to have the

student improve both at the same time rather than pushing style as an afterthought for the end and as a result make bugs harder to debug.

3 Hypothesis

Since we want to classify into more than one grade bucket a multi-class classification model seems appropriate for the situation. Intuitively, the three most important factors when considering whether a program has good style are whether it is well decomposed, well formatted and whether it works. But since a single program may have several possible correct solutions it may be the case that different combination of these characteristics produce different style grades for each solution. Therefore, it is of interest to explore whether running Softmax only on those programs that have an implementation similar to the test program has more accuracy than simply fitting Softmax to all the training data. And since we also want to provide feedback for non-working programs we explore the possibility of determining the features of the Softmax model by running logistic regression or naive Bayes on the functions alone to decide whether each function is well decomposed and well formatted and finally take an average for an entire program to determine whether the program is well decomposed and well formatted.

4 Implementation

4.1 Pre-processing

First, 125 programs were collected from the Karel midpoint finding assignment from the CS106A class. 70 of them working programs. 30 non-working and 5 from each grading bucket category (plus, check plus, check, check minus, minus). Initially, the plan was to obtain hundreds of programs from professor Chris Piech but due to student privacy issues, he was only able to provide a couple. As such, most of the programs were personally written making sure to include all major possible solutions and collected from friends. Then, after obtaining the programs, the training programs were pre-processed to obtain feature vectors describing each program and each function in each program. For the training data, the function vectors also contained a label on whether it is well decomposed and well formatted and the program vectors contained the actual grade of a program. As for the testing data, no program vectors were given.

Then, for clustering, 3 different strategies were explored.

4.2 Clustering

4.2.1 Strategy 1: Cluster by command counts

For this strategy, I parsed the 100 train set programs into the basic command primitives Karel could understand (move(), putBeeper(), pickBeeper(), turnLeft(), turnRight(), turnAround()) and counted how many of each were in the program. Then I ran k-means to cluster them in 6-dimensional space. Below is a 2D cross section of the two dominant features move() and turnLeft():

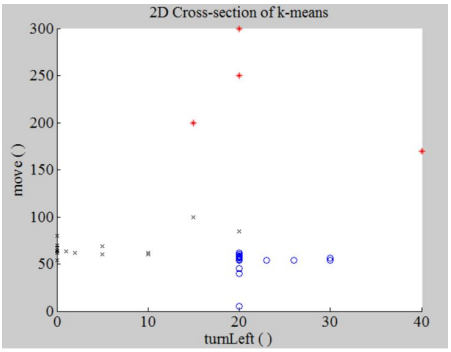

Figure 1: move() vs turnLeft() 2D cross section for k = 3

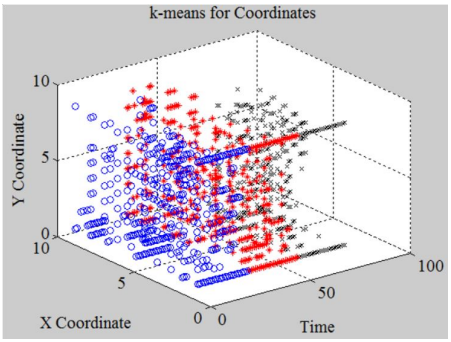

Figure 2: Coordinate clustering for k = 3

4.2.2 Strategy 2: Cluster by pouring all coordinates

For the second strategy, I kept track of the coordinates Karel was in during each time step in a program. This produced a set of [time, x-coordinate, y-coordinate] vectors for each program and a set of vector sets for all 100 train programs. Then I proceeded to run k-means by pouring all coordinates together.

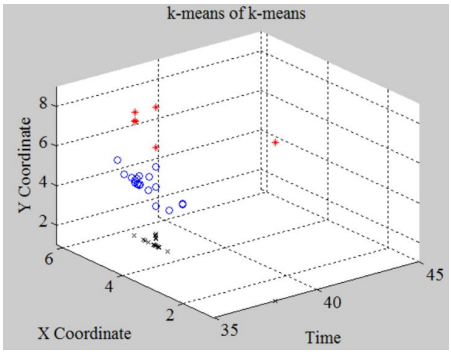

Figure 3: k-means clustering for k = 3

4.2.3 Strategy 3: Cluster by running k-means twice.

Then for the third strategy, I tried something more clever. Namely, to run k-means twice. Once to find the the average of the coordinates within a single program and the second time to cluster these averages into k clusters.

4.2.4 Clustering Results

Clustering by counts of primitives is very susceptible to unnecessary command calls. Also it does not distinguish between call times. Clustering by pouring all coordinates may be susceptible to outlying programs that tend to spend too much time at a certain place. Double k-means overall seems to address the issues above and can even be implemented for variable times. However, it breaks down with infinite loops.

4.3 Logistic Regression or Naive Bayes for function classification

With the training programs clustered, we proceed to fit a logistic regression or a Naive Bayes model for all the function is each program cluster. For this

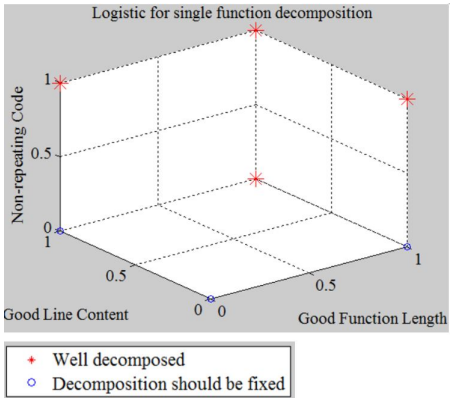

Figure 4: Logistic function regression

we use collected properties of each function. For decomposition we use the following feature vector: [Is the function between 2-10 lines?; does it have appropriate line length?;Is there no repeating code?;] And for function format we use the following: [Does it have correct indentation?; Is there no blank lines?; Is it commented?]. Shown above is an example of logistic regression for the functions in the train set.

4.4 Feedback

After fitting the cluster functions from our training data we can use them to provide feedback to a student on his functions as he is still working on the program. All we need to do is take his program and for each function decide whether it is decomposed or not and point to whether we found it to have appropriate length and non-repeating code as for format we can report whether we believe each function to be well formatted and point to whether it has correct indentation, no blank lines, and comments.

4.5 Softmax regression

Once we have decided whether each function is well decomposed and well formatted we take an average over all functions in the test program and test whether the program is functionally correct to come up with the program description vector [Program decomposed; program well formatted; Program Works]. Then we fit a Softmax regression model for each program cluster in the training data where $y^(i) \in \{1, 2, ..., k\}$. More specifically, we do gradient descent on the following cost function[1] including a weight decay term to ensure the cost function $J(\theta)$ is convex for any $\lambda > 0$

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\{y^{(i)} = j\} log \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}}] + \frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=0}^{n}\theta_{ij}^2$$

and taking the gradient gives:

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[x^{(i)}(1\{y^{(i)} = j\} - \frac{e^{\theta_j^T x^{(i)}}}{\sum_{l=1}^{k} e^{\theta_l^T x^{(i)}}})] + \lambda\theta_j$$

then plugging into the stochastic gradient descent rule we get:

$$\theta_j := \theta_j - \alpha\nabla_{\theta_j} J(\theta)$$

for each value $j = 1, ..., k$
where we set $\alpha := 0.001$ $\lambda := \frac{1}{m}$ $m :=$ Number of Train Programs and $k := 5$ Finally we use this model to assign a bucket style grade to a test program.

5 Softmax Results

The following table presents the accuracies obtained for each test set in each bucket category (consisting of 5 programs each as noted earlier.)

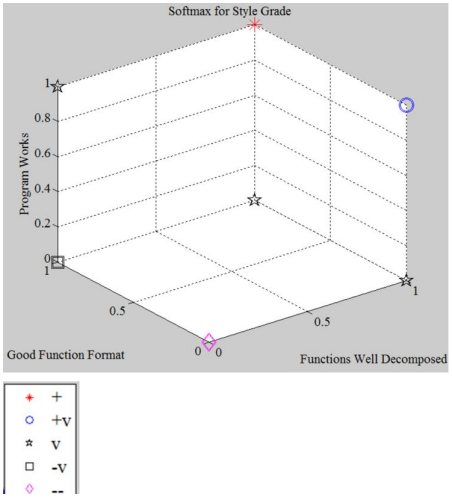

Figure 5: Softmax regression with logistic and clustering for training data.

BUCKET	Softmax with logistic and clustering	Softmax with logistic without clustering	Softmax with Naive Bayes and clustering	Softmax with Naive Bayes without clustering
PLUS	87%	75%	81%	70%
CHECK PLUS	66%	61%	60%	55%
CHECK	70%	65%	64%	60%
CHECK MINUS	64%	60%	59%	55%
MINUS	86%	75%	80%	69%

As we can see from the table, clustering before hand seems to perform better. Also running logistic regression on each function seems to have higher accuracy than logistic regression. This can be easily explained if we consider that the characteristics in the function vector are not necessarily independent. However, while our softmax model performed at 87% with logistic and clustering for the plus bucket, it performed poorly for the check plus and check minus buckets. On retrospect, we can explain this polarizing behavior by realizing that when we ran logistic regression on each function we only identified between well decomposed/formatted or not at all.

6 Future Work

A future possible fix would be to run Softmax on each function and take that average. However, that approach would be very susceptible to mal-formed functions and would likely require to increase the size of the feature function vector. Also we worked strictly within a Karel-world setting and as such it would be interesting to see whether these results also hold for purely java programs.

7 References

References

[1] Unsupervised Feature Learning and Deep Learning. "Softmax Regression" *http* : *//ufldl.stanford.edu/wiki/index.php/* $Softmax_Regression$