Rock Climbing Problem Classifier

Hoyin Chu

04/22/2019

Abstract

To classify rock climbing routes by their difficulty grade, we presented the 2017 Moonboard dataset in two ways: one-hot and symbol sequence. In one-hot representation we experimented with Naive Bayes, random forest, SVM, and DNN models. In symbol sequence representation we experimented with CNN and the FastText model. We propose a new metric named n-off-accuracy and found evidences that suggests symbol sequence may be the better representation for this problem.

1 Introduction

As recreational rock climbing gain popularity, the ability to automatically classify the difficulty of a rock climbing route (commonly referred to as *problem*) becomes desirable for climbers who want a consistent way to benchmark their performance and new route setters to get immediate feedback on the problem they are designing. Currently, one of the most widely used difficulty grading system for bouldering, a form of rock climbing involving climbing short walls without protective gear, is the Fontainebleau grading system, which typically ranges from 5 to 9A (5 being the easiest and 9A being the hardest).

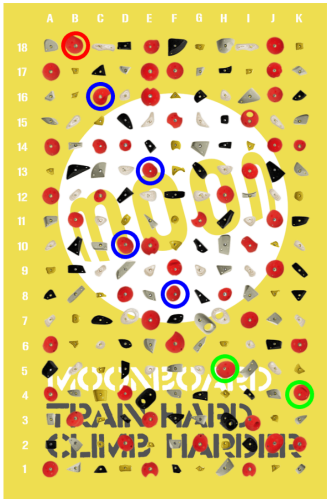


Figure 1: A sample moonboard problem, where the circled holds are the ones the climber is allowed to use, the green ones are the starting holds, and the red one is the finish hold

Moonboard is one of the most popular standardized bouldering walls. A Moonboard can be think of as a 2D-grid consists of $18 \times 11 = 198$ climbing holds, and a moonboard problem can be described as a series of holds, as indicated on the figure above. Since all Moonboards of the same version have the exact same layout and holds, users can create and share their problems online with others on the Moonboard website, which we use as our data source.

2 Technical Approach

To obtain our dataset, we wrote a web scraper written in javascipt using the library Puppeteer.js, scraped about 30000 problems, and loaded them into MongoDB Atlas. Each problem has a list of holds that the user is allowed to use and a grade associated to it. In previous work[1], the list of moves are represented as a one-hot vector: the 2D Moonboard matrix is flattened to a 1D vector where the holds that are available to be used by the climber have value 1 while the others have value 0. While this is a simple way to represent a list, it has several disadvantages: the result data is a high dimensional sparse matrix, the information on hold ordering is lost, and the same technique cannot be applied to any other rock climbing route classification tasks outside of this specific version of Moonboard. Nevertheless, we applied 4 different classification models, namely Naive Bayes, random forest, support vector machine, and vanilla deep neural network using this data representation, and showed that using simpler models with less training time can achieve the same performance as the ones used in previous work.

To make use of the fact that our target data is ordinal, we experimented with ordinal classifiers where instead of directly predicting the class label using one multi-label classifier predicting n classes, we train $n-1$ binary classifiers, where each classifier is responsible for producing the probability of a sample having a grade greater than i, where $0 <= i <= n-1$. In the end, we can predict the probability of the sample belonging to class y by calculating the difference between the probability of it having a grade greater than $y-1$ and the probability of it having a grade greater than y. For example, $p(grade = 0) = 1 - p(grade > 0)$, $p(grade = 4) = p(grade > 3) - p(grade > 4)$, $p(grade = n) = prob(grade > n-1)$. However, due to the poor performance and the significant increase in training time, we only experimented this method on the Naive Bayes classifier.

Due to the ordinal nature of our target variable, we also propose a new evaluation metric called n-off-accuracy, where the prediction of a model is considered correct as long as it is within (ground truth $\pm n$). For example, if the model predicts the grade of a climb to be 3 while the ground truth is 2, it will be considered as a correct prediction in 1-off accuracy. The basis for this metric is to take into account that due to the subjective nature of climbing grades, climbers generally tolerate grades that are 1 off of what they would expect it to be.

The data representation that we propose is to simply join all the holds used as a single string, delimited by space, ordered from starting holds to finish holds. We draw inspiration from the field of natural language processing, where significant progress have been made due to the advancement of deep learning. In particular, we are interested in models that perform well on multi-label sentimental analysis, as they have the capability to map a sequence of symbols to a discrete scalar value. In our case, a climbing problem is nothing but a sentence where the holds are the words, and the sentiment of the sentence is the difficulty of the climb.

Aside from the starting holds and the finishing holds, there is no explicit requirement on the order of the holds in which they should be used. Therefore, we tried to order the holds in a way such that it is as close as possible to the order in which a real climber would use them in. To do this, we created a simple ordering algorithm that, on a high level, starts with one of the specified starting hold, then grabs the next easiest to get to hold as the next hold, and repeat until it reaches the finish hold. Keep in mind we only have to reorder "middle holds" (holds that are not starting nor finish holds). The detail of the algorithm is as follow:

Algorithm 1: HoldReordering($h_0, (h_1, h_2, ..., h_n)$)

Let h_0 be one of the starting holds;
Let $l = (h_1, h_2, ..., h_n)$ be the list of "middle" holds;
if *l is empty* **then**
  return ();
else
  Find the hold h_i with minimum modified Manhattan distance (defined below) to h_0;
  modifiedManDist = horizontal distance (h_0,h_i) + 1.5×vertical distance(h_0,h_i);
  remove h_i from l;
  return append(h_i,HoldReordering(h_i,l));
end

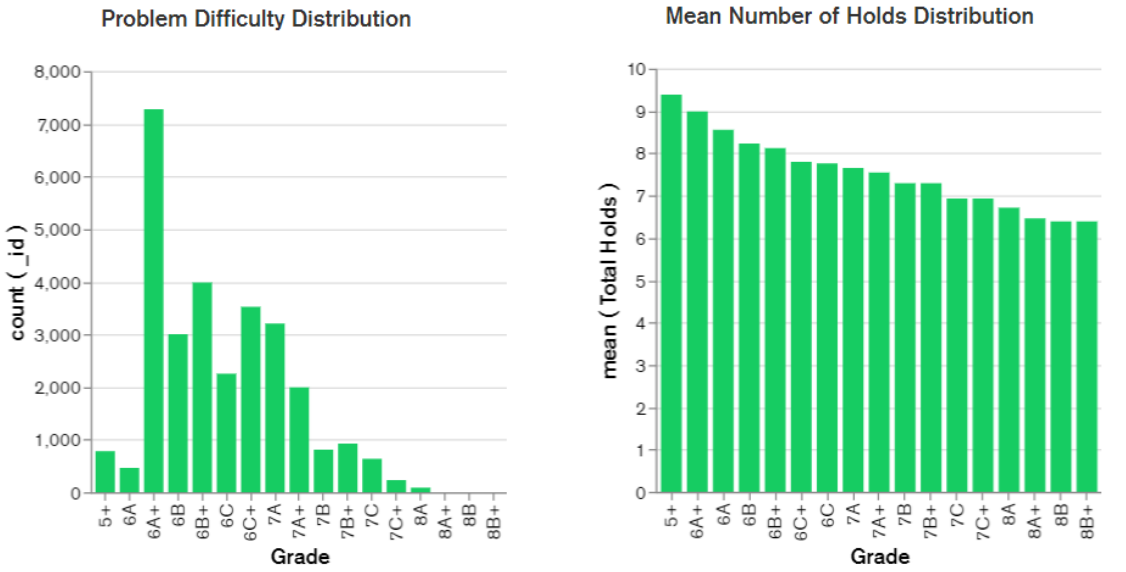
The reason behind putting weight of 1.5 on vertical distance is that in real life in most cases it's harder to reach a hold vertically than it is to reach a hold horizontally, hence when comparing two holds

that are both the same number of holds away, the one that is closer horizontally is preferred.

Since this is a simple heuristic that will not always produce the optimal ordering of holds, to reduce the influence of ordering without completely ignoring it, we use a convolutional neural network (CNN) to tackle our sequence classification problem. We also used the popular FastText Model[2] as a baseline comparison of performance with respect to this NLP approach

To understand if our results hold any statistical significance, we also created a dummy classifier that always pick the most frequently seen class and a SVM classifier that only uses the number of holds in a problem as its only feature.

3 Experimental Results



(a) Grade Distribution

(b) Average amount of holds used per problem per grade

Figure 2: Some distributions about the dataset

Since grades are ordinal variables, we replace grades by integers starting from 5+ being 0, 6A being 1, 6A+ being 2 and so on. Some simple data analysis reveals that our data is skewed toward 6A+, but since it's not too overly-represented (26% of all data) we leave it as it is. We can also see that there aren't too many problems above 7C, so we relabel anything more difficult than 7C as 7C to indicate the new category "7C or harder". Since any user, including inexperienced ones, can post problems they created and give them grades, we choose to only include problems with user rating 2 or above (3 being max) to reduce noise. We can also see as the grade increases, the number of holds a climber is allowed to use decreases. Base on this observation, we created a SVM model that only used the number of holds as feature, and used it as a baseline to test if our results had statistical significance. Other properties about the dataset can be explored via an online interactive dashboard we created on MongoDB Atlas[3].

Using the Scikit-Learn library[4], we constructed a Naive Bayes model with default parameters, an ordinal version of the Naive Bayes model, a random forest model with default parameters, a support vector machine (SVM) model with default parameters, and used the train_test_split method to split the dataset into training set and testing set with ratio of 0.75 and 0.25. We also constructed a deep neural network with 2 hidden layers, 512 neurons each, using relu activations between layers, cross entropy as the cost function and stochastic gradient descent as the optimizer using the Pytorch library[5]. These models were train and tested on the one-hot data representation, and their metrics are listed in Table 1.

For the sequential representations, we constructed a CNN model with dropout using the adam optimizer using a tutorial from Ben Trevett[6] on convolutional neural network on sentimental analysis. We

Table 1: Model Performance Using One-Hot Representation

	Naive Bayes	Ordinal Naive Bayes	Random Forest	SVM	DNN
Exact Accuracy	31.70%	26.67%	33.79%	**36.72%**	26.95%
1-off-accuracy	56.08%	47.72%	58.90%	**61.01%**	39.84%
2-off-accuracy	76.97%	72.53%	79.55%	**81.83%**	56.37%
MAE	1.60	1.87	1.46	**1.35**	2.73

Table 2: Model Performance Using Sequence Representation

	CNN	FastText	Dummy	SVM (Using only number of holds)
Exact Accuracy	**34.60%**	29.59%	13.48%	26.97%
1-off-accuracy	**56.35%**	45.49%	30.96%	40.55%
2-off-accuracy	**73.85%**	60.78%	49.28%	57.14%
MAE	**1.83**	2.52	3.04	2.66

then used the ax platform library[7] for hyperparameter tuning via Bayesian optimization, and trained the model, using parameters found during previous step, over 5 epochs on the training set, validated against an validation set, and saved the model with the best validation set performance. We also followed the tutorial and constructed a FastText model[2] to act as a sequence classification baseline reference. For statistical significance comparison, we used the scikit learn library to build a dummy model that generates prediction base on distribution it saw during training, and and SVM model that only uses the number of holds in a problem as its only feature. These models and their related metrics can be seen in Table 2.

From the results listed in the first table, we see that SVM wins in all metrics using one-hot representation. It is worth noting that the SVM we constructed here actually slightly outperformed the best classifier in previous work (softmax regression classifier with 36.5% exact accuracy and 1.37 MAE)[1]

From the results listed in the second table, we see that CNN wins in all metric using the sequence representation. This suggest the information on the ordering of the hold may have played a role as CNN takes that into consideration while FastText do not. While CNN still lag behind SVM, we believe with more layers and finer hyperparameter tuning, with the additional information on hold ordering and its flexible nature, it will be able to outperform SVM given more experimentation. This is supported by the fact that this CNN has a better exact accuracy compared to the CNN constructed in previous work that was trained using data in the one-hot representation (34.0%).

In conclusion, our work holds statistical significance as all models of interest were able to significantly outperform dummy models, and we were able to find a model using one-hot representation that outperforms previous work. We also found evidence that the sequential representation of climbing holds is a better representation than the one-hot representation, and came up with an reordering algorithm that can translate existing Moonboard problems into sequential representations.

4 Participants Contribution

All work in this project are done by Hoyin Chu, with some advices given by Prof. Elhamifar

References

[1] Alejandro Dobles,Juan Carlos Sarmiento,Peter Satterthwaite, *Machine Learning Methods for Climbing Route Classification.*
`http://cs229.stanford.edu/proj2017/final-reports/5232206.pdf`

[2] Armand Joulin, Edouard Grave, Piotr Bojanowski, Tomas Mikolov, *Bag of Tricks for Efficient Text Classification.* arXiv:1607.01759

[3] MongoDB Atlas
`https://charts.mongodb.com/charts-project-0-sypfa/public/dashboards/62bdc4dd-abe4-4906-a37a-9bef6b254f6b`

[4] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[5] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, Adam Lerer, *Automatic differentiation in PyTorch*.

[6] Ben Trevett, *https://github.com/bentrevett/pytorch-sentiment-analysis*

[7] Adaptive Experimentation Platform, *https://ax.dev/*