CS6300 Final Exam

Chinmay Chandak, Tapan Sahni

December 14, 2019

Abstract

This report analyzes the result of loop unrolling with different machine learning techniques over SPEC 2006 benchmarks (Positive/Negative performance on different varieties of loops). We have experimented with most of the well known machine learning techniques (parametric and non-parametric) that deals with small training data as in our case of loop unrolling.

1 Introduction

We experimented with quite a few machine learning algorithms to analyse the results of loop unrolling over the SPEC2006 benchmarks. Table 1 below shows the accuracy of the predicted unroll factor labels (the baseline labels were obtained using a Genetic Algorithm) we obtained from various learning algorithms.

1.1 Outline

The report is divided into the results section that will be dealt in part 2. This is followed by the explanation of the results for machine learning accuracies and run-time improvements with loop unrolling improvement.

2 Analysis of Loop Unrolling for different Machine Learning Classifiers

We now analyse the above results in the below table and provide a comprehensive explanation as to why we get these kinds of results.

Nearest Neighbors	Support Vector Machines	Random Forests	Artificial Neural Network	Decision
45.04%	42.0%	41.3%	39.1%	38.7%

- We had initially experimented with Logistic Regression, and the results were really poor, around 22%. This owes to the fact that although Logistic Regression seems to be a convenient classifier because it provides convenient probability scores for observations,

it doesnt perform well when the feature space is too large. Also, it doesnt handle large number of categorical features/variables well, and relies on transformations for nonlinear features. Logistic regression is a pretty well-behaved classification algorithm that can be trained as long as you expect your features to be roughly linear and the problem to be linearly separable. It is also pretty robust to noise and you can avoid over-fitting and even do feature selection by using regularization. Unfortunately, it cannot be used in our case.

- We now move onto Tree Ensemble methods: Random Forests and Gradient Boosted Trees. Tree Ensembles have different advantages over Logistic Regression. One main advantage is that they do not expect linear features (take into account variable interactions) or even features that interact linearly (LR can hardly handle categorical (binary) features). Tree Ensembles, because they are nothing more than a bunch of Decision Trees combined, can handle this very well. The other main advantage is that, because of how they are constructed (using bagging or boosting) these algorithms handle very well high dimensional spaces as well as large number of training examples. We can see that Random Forests perform better than Decision Trees, and that is obvious, because Decision Trees are highly biased to the training set unlike Random Forests.

- As for the difference between Random Forests (RF) and Gradient Boosted Decision Trees (GBDT), but it has been observed empirically that GBDT will usually perform better, but they are harder to get right. More concretely, GBDT have more hyper-parameters to tune and are also more prone to over-fitting. Random Forests can almost work out of the box and that is one reason why they are very popular. Overall, ensembles of decision trees (such as Random Forests, which is a trademarked term for one particular implementation) are very fast to train, but quite slow to create predictions once trained. More accurate ensembles require more trees, which means using the model becomes slower.

- Result of Support Vector Machines are not as not as intuitive as decision trees for a layman. The advantages are that SVMs can handle large feature space, can handle non-linear feature interactions, and do not rely on entire data. However, they are not very efficient with large number of observations, and it can be tricky to find appropriate kernel sometimes. There are cases when you would expect SVM to work efficiently, but it wont. The possible causes include the use of an inappropriate kernel (e.g. a linear kernel for a nonlinear problem), poor choice of kernel and regularization hyper-parameters. Good model selection (choice of kernel and hyper-parameter tuning is the key to getting good performance from SVMs, they can only be expected to give good results when used correctly). SVMs often do take a long time to train, this is especially true when the choice of kernel and particularly regularization parameter means that almost all the data end up as support vectors (the sparsity of SVMs is a handy by-product, nothing more). Lastly, the no free lunch theorems say that there is no a-priori superiority for any classifier system over the others, so the best classifier for a particular task is itself task-dependent. However, there is more compelling theory

for the SVM that suggests it is likely to be better choice than many other approaches for many problems.

- We also experimented with Neural Networks, because it would wisely select only the features which were non-redundant and helpful for classification. Thus, the large feature space would be taken care of automatically. The VC dimension (the VC dimension (for VapnikChervonenkis dimension) is a measure of the capacity (complexity, expressive power, richness, or flexibility) of a space of functions that can be learned by a statistical classification algorithm) of neural networks is unclear, i.e., there is a chance of over-fitting due to the model being complex, and, regularization, along with a lot of other hyper-parameter tuning would be required. This is very important when you want to consider how good a solution might be. Also, Neural Network take a long time to train in our case, which just negates all the other advantages.

- Finally, we landed on K-Nearest Neighbors. This seemed to be a wise choice considering the facts that KNN has been proven to be effective when the training data is large, and is robust to noisy training data. K-nearest neighbors requires no training. Also, we can tune k, and get optimal results. KNN can learn non-linear boundaries as well. Unlike Logistic Regression which predicts probabilities, which are a measure of the confidence of prediction, KNN predicts just the labels. Basically, it is an extremely easy-to-implement, simple non-hyper-parametric classifier that can work for any kind of data. All these factors are apt for our case, and the results we got, conform to this.

- Hence, the results show that K-NN performs the best, followed by SVM, Random Forests, NN, and finally Decision Trees. GDBTs and Logistic Regression, as mentioned earlier, have poorer results.

3 Analysis of Loop Unrolling Improvement on Different SPEC Benchmarks

We now analyse the above results in the below table and provide a comprehensive explanation as to why we get these kinds of results.

$bzip2$	$astar$	mcf	$sjeng$	$omnetpp$	$hmmer$	$h264ref$	$gobmks$
8.36%	9.41%	3.69%	15.06%	9.51%	-7.1%	0.897%	-2.3%

Having seen the different accuracies we obtained on various learning techniques used, we now analyze the final run-time improvements we obtained over the SPEC 2006 benchmarks. What aspects or characteristics of the loops culminate into a positive improvement, and what characteristics cause negative performance?

The maximum run-time improvement was seen on $sjeng$, and the minimum was seen in $hmmer$. The results conform to the facts that $hmmer$, $h264ref$, and $gobmk$ are the largest of the chosen benchmarks, and there is a high possibility that code bloating could play a prominent detrimental role when large unroll factors are applied to loops.

We analyzed the feature data we had for all the benchmarks, and also the difference between the accuracy of predicted labels we obtained from our model; this accuracy translates to the difference in the final run-time improvement values. We now try to explain how the features chosen for training interacts with the run-time improvement.

- Loops having greater number of data dependences (flow dependences, anti dependences, control dependences, or output dependences) show lesser accuracy in predicted labels than the ones having lesser number of dependences; thus, the number of memory dependences is inversely proportional to the accuracy of predicted labels. This result seems the most intuitive, because, larger the number of dependences, more number of variables and instructions are intertwined within the loop, and higher is the register pressure, and thus predicting the optimal unroll factor is more difficult in this scenario.
- Extending the first point, more the height of the memory-to-memory dependences(used as a feature), lesser the accuracy.
- Greater the number of parallel computations in the loop, lesser the accuracy. Cloning parallel instructions inside the loop might lead to increase in runtime because the number of parallel instructions might decrease; so this result seems on point.
- Higher the number of branches in the loop, lesser the accuracy. When a loop has multiple exits, the compiler isnt able to figure out the optimal unroll factor because a lot of time, dynamic control flow leads to an exit of the loop, and thus all the predictions are in the wrong.

Thus, benchmarks which are pervaded with any of the above factors show a lesser run-time improvement (maybe even negative) as compared to the ones which have somewhat simpler, less parallelizable instruction set, and void-of-dependences loops. For example, most of the *sjeng* loops do not have a lot of dependences, and almost all of them are single-exit loops; this is contrary to the loops of *gobmk*, and *hmmer*. This translates to the fact that we got the maximum run-time improvement of 15.1% on *sjeng*, and the minimum improvement of -7.1% on *hmmer*.

4 Conclusions

We have given a comprehensive analysis of how different machine learning classifier perform when finding a function for optimal loop unrolling. We have also tried to explain the interaction of the features used and the run-time improvement obtained on different benchmarks.