Analysis of Code Submissions in Competitive Programming Contests

CS 229 Project, Autumn 2018
Wenli Looi (wlooi@stanford.edu)

Motivation

- Competitive programming provides an opportunity to gain insights into coding techniques.
- In a typical contest, participants solve 5-10 well defined algorithmic problems.

Problem

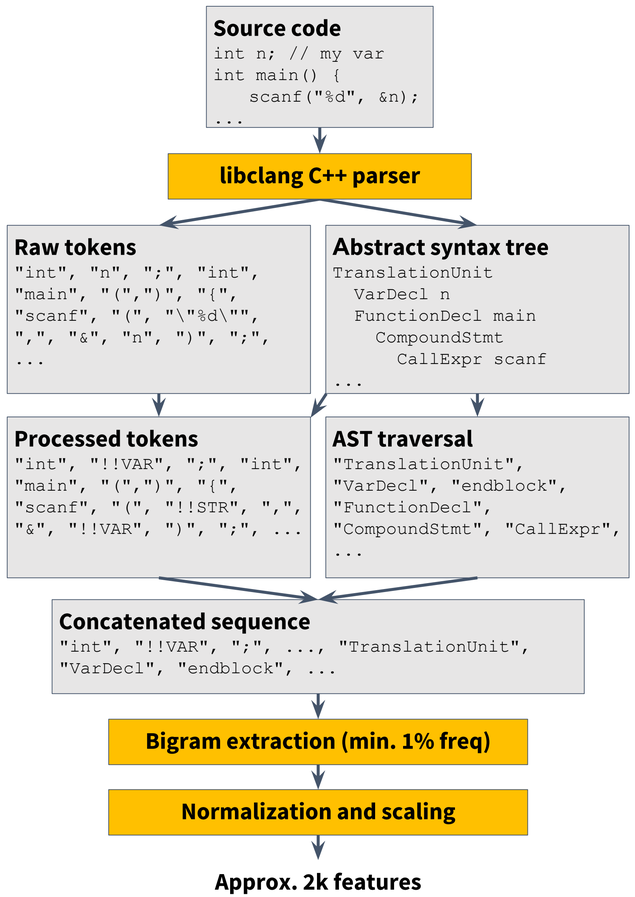
- Predict the **rank** (±1) and **country** of a competitive programmer given only a single C++ code sample.
- Focus on coding style: Only consider working solutions, and not consider comments or formatting.

Codeforces Rank	Rating Range	Codeforces Rank	Rating Range
Legendary Grandmaster	3000+	Candidate Master	1900-2099
International Grandmaster	2600-2999	Expert	1600-1899
Grandmaster	2400-2599	Specialist	1400-1599
International Master	2300-2399	Pupil	1200-1399
Master	2100-2299	Newbie	0-1200

Data Set

- 10 Codeforces contests, Aug-Nov 2018.
- Selected contests are open to all ranks.
- Scraped with custom scraper.
- Only consider last passing submission.
- Only consider C++ (~70% of total).
- ~6k submissions per contest, total ~60k.
- For country analysis, only participants in the top 10 countries (~70% of total).

Preprocessing



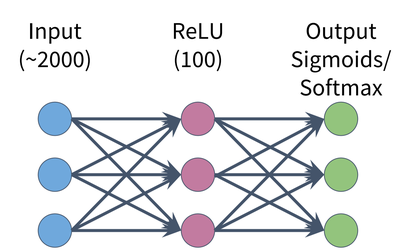
Models

Linear regression (for rating)
- Loss for single example: $-w^{(i)}(\theta^T x^{(i)} - y^{(i)})^2$
- Weights are inverse of class size.

Gaussian discriminant analysis (GDA)
- $p(y = k) = 1 / (\#\text{ classes})$ (Forced uniform)
- $p(x \mid y = k) \sim N(\mu_k, \Sigma)$

Logistic regression
- For **country prediction**, standard weighted softmax.
- For **rank prediction,** train 10 independent logistic models. Each example is part of ranks r-1, r, r+1.
- Weights are inverse of class size.

Neural network
- Output layer and weighted loss functions are the same as logistic regression.
- 50% dropout for hidden nodes in training.

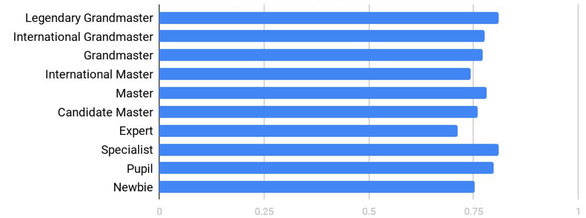


Results (10-fold cross validation)

- Training on 9 contests (~54k examples) and testing on 1 contest (~6k examples) to test generalization to unseen problems.
- Accuracy is the weighted accuracy where the weight of each example is the inverse of the class size. This shows how well the model can predict all classes. (Unweighted accuracies are generally higher.)

Model	Accuracy (Rank±1)		Accuracy (Country)	
	Train	Test	Train	Test
Random/constant	30.0%	30.0%	10.0%	10.0%
Linear regression	69.6%	60.1%	N/A	N/A
GDA	75.7%	67.2%	75.0%	65.0%
Logistic regression	86.1%	71.6%	92.2%	68.4%
Neural network	94.4%	77.2%	97.0%	72.5%

The model is not heavily biased towards larger classes:



Discussion

- Neural network achieves the best test accuracy of 77.2% for rank (±1) and 72.5% for country.
- The neural network is probably able to learn more complex relationships between the features compared to the other models.
- Predicting the rank using classification worked better than predicting the rating using regression. This may be because classification optimizes what we actually care about, which is predicting the correct rank.
- GDA works surprisingly well, almost as well as logistic regression. The normalized/scaled feature space seems Gaussian to some degree.
- There may be some overfitting.

Interpretation of GDA model

For International Grandmaster vs. Pupil, here are the features where the class means differ the most.

Strongest indicators of high skill level
- Use of #ifdef, assert, and function templates.

`ifdef`	`# ifdef`	`assert`	`endif`
`# endif`	`assert (`	`( ...`	
`FunctionTemplate TemplateTypeParameter`			
`__VA_ARGS__`	`FunctionTemplate`		
`#ifdef LOCAL`		`LOCAL`	

Strongest indicators of low skill level
- Use of cin/cout instead of scanf/printf, or perhaps cin >> x; cin >> y; instead of cin >> x >> y;
- It seems that the model uses some features to favor shorter solutions. TranslationUnit is always present exactly once, so the normalized value will be higher in shorter solutions.

`cin >>`	`cin`	`>> !!VAR`	`>>`
`cout <<`	`cout`		
`TranslationUnit InclusionDirective`			
`TranslationUnit`	`std ;`		
`IfStmt BinaryOperator`	`main`	`main (`	

Future Work

- More data will likely help. Going from 5 to 10 contests increased the accuracy significantly.
- Try a recurrent neural network, e.g. LSTM.
- Improve token processing, e.g. also replace class and macro names with placeholders.
- Interpretation of the neural network model.

References

Prior work with similar ideas:

1. Alsulami, Bander, et al. "Source Code Authorship Attribution Using Long Short-Term Memory Based Networks." European Symposium on Research in Computer Security. Springer, Cham, 2017.
2. Burrows, Steven, and Seyed MM Tahaghoghi. "Source code authorship attribution using n-grams." Proceedings of the Twelth Australasian Document Computing Symposium, Melbourne, Australia, RMIT University. 2007.
3. Ugurel, Secil, Robert Krovetz, and C. Lee Giles. "What's the code?: automatic classification of source code archives." Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2002.

Clang, Julia, Scikit-learn and TensorFlow were used to implement this project.