CSCI 181 / E-181 Spring 2014 Practical 3

Kaggle Team "Capt. Jinglehiemer"

David Wihl
davidwihl@gmail.com

Zack Hendlin
zgh@mit.edu

March 12, 2014

Warm-Up

Warmup Topic 1

Warmup Summary

Classifying Malicious Software

Preliminary Data Analysis

This problem was significantly more complex than the previous practicals due to a number of factors: unbalanced distribution of the data, sparsity of the data, conversion of hierarchical XML data into a myriad of choices for vectorization.

Significant experimentation was required to iterate over feature engineering, classifier choice, permutations of ensembles of classifiers and hyper parameter optimization.

The first step was a significant refactoring of the sample code in order to separate vectorization, classification, validation and submission.

Using Cross-validation

Very early we created a test harness that enabled testing of n iterations (default 5) over cross-validation sets consisting of 30%, 20% and 10% of the data. This proved very valuable as we experimented. In total, we ran over 200 cross-validation tests each consisting of 15 different passes.

Per File Feature Engineering

The first vectorizer took simple counts of each different type of system call as well as per-process (e.g. 'startreason', 'terminationreason', 'username', 'executionstatus', 'applicationtype') and per-file metrics coupled with an unoptimized logistic regression. This enabled us to achieve a 75% accuracy on the Kaggle submission within just a day or two.

Comparing Classifiers and Tuning Hyperparameters

We then attempted a number of different classifiers (SVM, kNN, Gradient Boosting). Results are summarized in TODO.

By making multiple cross-validation runs, we tuned each classifiers hyper parameters.

Pruning the Dimensions

We took two complementary approaches to prune the dimensions. 1. We examined the weights of each of the classifiers to see which dimensions were not producing a signal. Using the mean and standard deviation of the resulting matrix, we noted any features where abs(mean) < 0.001 and std <0.01. 2. We exported our Design Matrix to a CSV and then used tools in R to run a linear regression and see the predicted values of the coefficients.

We then pared back approximately 20 features that provided no value (and ran cross validation again to recheck of course).

Combining Classifiers

We examined and experimented with several options for combining the classifiers (AdaBoost, Random Forests). We wanted to attempt a novel approach to combining classifiers (using predict_proba to find what level is >98% is appropriate for a given classifier. Then if the classifier, such kNN, had a high confidence level, it was given the vote. If one classifier had a significantly higher confidence level (determined empirically to be 0.4), it was given the vote. If no classifier had a high confidence, it was given to Logistic Regression as it generally had the lowest error rate.

TODO Combined LR and kNN. Chose kNN only when it's confidence was 0.4 greater than LR's.

Per System Call Feature Engineering

We created second, separate model of per system-call metrics, consisting of items such as 'successful' attributes. Attributes of type string that had potential value were hashed into a numeric form. This resulted in a design matrix of over 3000000 rows $\times$ 14 features.

We then applied Logistic Regression to this new design matrix and rechecked, as always with multiple cross-validation passes and again pared back any features that did not have a significant signal.

Quality Check prior to Submission

After a couple of poor submissions due to bugs in our algorithms, we implemented a sanity checked for submissions. This would useful in real world scenarios to ensure that the predictions are within a reasonable tolerance prior to putting new models into production. The sanity check ensured that all test data had a predictions within the possible values, and then displayed a distribution of distribution to ensure that no unusual skews were occurring.

Exotic Classifiers

We examined and did preliminary experiments with two different approaches: Exemplar SVM and Theano / Deep Learning. While the reading was interesting, in both cases, the quality of the code and library precluded us from pursuing it further. We hit a bug in Theano when using the Scipy CSR array. We intend to continue exploring Theano outside the scope of this exercise in the future.

Conclusion

This was a valuable exercise in understanding the state of the art in multi-class classification with real-world, sparse hierarchical data. Clearly, *there is no free lunch.*[1]

A more general approach to attacking this type of problem requires more research and investigation. The permutations of features, classifiers and hyper parameters grow geometrically without many significant tools to provide automated means of attacking the problem. This is an area we would be interested in pursuing further research.

[1]Wolpert, Macready, *No Free Lunch Theorems for Optimization* http://ti.arc.nasa.gov/m/profile/dhw/papers/78.pdf

We were surprised how little guidance there was in the literature for this type of problem. We scanned through several books, journals, articles and web sites, but found little external guidance.

Background Reading

[1] Multiclass SVM classification http://svmlight.joachims.org/svm_multiclass.html

[2] Exemplar SVM on github (octave / matlab) https://github.com/quantombone/exemplarsvm

[3] One against all or One against All http://hal.archives-ouvertes.fr/docs/00/10/39/55/PDF/cr102875872670.pdf

[4] Classifier comparison: http://scikit-learn.org/stable/auto_examples/plot_classifier_comparison.html