Corey Schulz

12.10.19

CS5350

Professor Srikumar

Machine Learning

Machine Learning – *Project Final Report*

For my final project, I decided to do the competitive competition. Below are my results and small analyses for each of the six algorithms used to run the Android Malware dataset: Perceptron, LIBSVM's SVM, our class SVM, Naïve Bayes, Random Forest, and SVM Over Trees.

- Perceptron

Submission Name: decaying_perceptron_90000_epochs.csv

Submission Public Score: 0.68232

First, because the Perceptron algorithm we wrote uses labels in {-1, 1}, I changed all the zeros to negative one in the data, then did cross validation on the learning rate and Perceptron variant. I determined that (surprisingly) Decaying Perceptron was the best here, with a learning rate of 0.01. (In fact, averaged perceptron only got a score of about .49!) So, I ran that for the anon data with 1000 epochs and printed out my results then submit to Kaggle. It got an fscore of about .67. Then, I tried with 90,000 epochs and an hour and a half of compute time later, the fscore went up a bit to .68232. Not really a jump in score worth the time spent computing the classifier. I'm surprised that Averaged Perceptron did so poorly, though! It just goes to show the efficacy of cross validation.

- *LIBSVM's SVM Implementation*

Submission Name: libsvm_solution_invert.csv

Submission Public Score: 0.48492

I'm using this as my external library submission for the final project. This classifier got about the same as my implementation of Averaged Perceptron, which is a bit disappointing. This is reasonable, as in the end these two algorithms are doing a lot of the same work. Though since this data isn't very well linearly separable, the SVM implementation didn't do a very good job. The classifier was quite easy to train, however. I just downloaded the library from GitHub, then ran `python3 libsvm_train inputs.file` basically and it generated a classifier that could be ran against the anon data. The accuracy for the first run was incredibly low – about 0.09, but when I inverted the answers the final fscore jumped to *0.48492,* which is actually worse than just always predicting the majority element. Not very effective overall, but this could be effective in training on other, more linearly separable data sets. During this run, I also did no feature transforms for the data so that also leads to poorer results. Further, this implementation of the algorithm was hard to control because the version of it that I used did not support choosing your own hyperparameters.

- *Support Vector Machine*

Submission Name: svm_predictions_2.csv

Submission Public Score: 0.59833

It was surprising that my implementation got more than the stock LIBSVM's score on this data. This is because of a few reasons: I transformed the data to be discretized instead of the stock data. I performed a threshold on the median value of the features for this, where values that fell below the threshold got assigned a -1. After cross validation, the best hyperparameters reported for my run was a learning rate of 1, a regularization of 10, and I gave it some extra compute time with 1000 epochs. This data did better than the LIBSVM version because I had control of the hyperparameters and could do transforms on the data in my code. Had I not done cross validation on the data, my SVM would have performed worse than LIBSVM's variant on the algorithm (and it did the first time I ran it).

- Naïve Bayes

Submission Name: naive_bayes_2.csv

Submission Public Score: 0.64285

For this submission, I discretized the data in the same way that I did for SVM, thresholding on the median feature value. I cross-validated and ended up with a smoothing term of 0.5 being the best hyperparameter for the dataset. That got a .619 score. I undid the discretization and kept the default data values, and the score went up to 0.64285. I learned from this that, though a feature transformation works for some algorithms, it may not work for all of them. It makes sense that Naïve Bayes did better on this dataset than the other algorithms discussed thus far given that it has access to all the information from the outset of running the algorithm and it uses a more probabilistic approach to solving the problem.

- Random Forest

Submission Name: random_forest_2.csv

Submission Public Score: 0.71217

Random forest is an improvement in prediction over using straight decision trees of depth 2 (a value gotten by cross validation) of about 10%. I discretized the data, thresholding on the median, and originally ran the algorithm with a forest size of 500 trees. As, in general, increasing the size of the forest should usually average out to an increased accuracy, for this submission I increased the size of the forest to 5,000 trees. It performed better than anything else I had done to this point in the project.

- SVM Over Trees

Submission Name: svm_over_trees_3.csv

Submission Public Score: 0.72483

Unsurprisingly, SVM Over Trees performed a bit better than simply using random forest. In fact, out of all the runs I have done with different algorithms, this has performed the best thus far. I discretized the data on a threshold on the median of the feature values. This time, I used 100 trees in the forest, and for the SVM's parameters a learning rate of 1 and a 40 regularization with 1,000 epochs. I believe that this classifier did a good job of classifying the data without using outside libraries! I used cross validation on the learning rate and the regularization, and it was surprising to see such a high regularization value come out of that validation. Indeed, the values in the data do vary wildly when it's not discretized so the result of this cross validation is reasonable.

Closing: What I Learned

I believe that this project *did* indeed help me learn about ML and get better at it in a practical sense! The project really nailed down the important of cross validation when choosing hyperparameters, and the hyperparameters change depending on the data set. It was also interesting to find out that the way you change your data has important impacts on the results of the learning algorithms you run on the data. While researching which outside library to use for my prediction (which, due to the limited control available in the implementation I used turned out to be a poor decision in terms of prediction accuracy), there are so many tools out there to do the job! I would like to experiment with a lot more of them in the future to solve problems! However, I *did* find it hard to reason about what constitutes a good feature transform. The data presented to us was just numbers, and I would have liked to have known which feature corresponds to which system call to actually draw conclusions about the data and the classifier in general. I also learned how much of a balancing act machine learning is. It's a balancing act between compute time, resources, hyperparameters and data organization. There's a lot to manage and a lot more things I want to learn about it.

If I had more time to work on the project, I would have liked to implement a simple neural network (and then use PyTorch or Tensorflow's implementation of it!) These tools are rather powerful, and I would really like to see how they perform at classifying this data set.

All in all, this class was a very good introduction to the discipline of machine learning and I'm very glad that I took it!