RIT

Rochester Institute of Technology

CSEC 520/620: Cyber Analytics & Machine Learning

Assignment 3 - Malware Classification

Due Date - October 25, 2021 11:59 PM

Purpose

The purpose of this assignment is to learn about a linear classification algorithm to classify malware samples. Your main task is to implement and train the classifier using an optimization technique to identify if a sample is malware and the malware's attack family.

Description

You and your team will implement

- Support Vector Machine (SVM) and
- Stochastic Gradient Descent (SGD)

You are NOT allowed to use any third-party libraries (e.g. scikit-learn, etc.). You are free to use any external libraries so long as you are not using the libraries' SVM/SGD/Hinge loss/One-vs-All/One-vs-One classification implementation.

Dataset

Get the dataset from here. Read the readme file to learn about the data and familiarize yourself with the dataset.

Steps

1. **Dataset Processing**

You will process each sample into a large vector that identifies whether or not each feature is present in that file.

Vectorizing:

- Build the list of all unique feature strings present in the dataset.
- For each sample, build a vector of 1s and 0s in which a 1 signals that the feature was present in the sample and a 0 signals that it was not.

2. **SVM**

After the data has been loaded and processed, you will build a linear SVM to classify the samples. A linear SVM is a binary classification model that uses a hyperplane to separate two classes of data. This hyperplane is defined by a set of parameters, *w* and b. See *Figure 1* for a helpful visualization of these parameters to form the hyperplane.

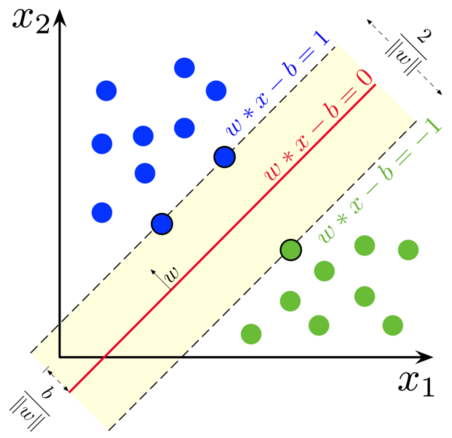

Figure 1: Mathematical definition of the SVM hyperplane (from wiki).

An SVM model predicts using its hyperplane parameters using the following equation:

$$y = w * x - b = w_0 * x_0 + w_1 * x_1 + ... + w_n * x_n - b$$

The signage of the y prediction value indicates which side (i.e. class) the sample vector x lies on. For the SVM to work, the class labels must be either -1 or +1.

3. Finding the Optimal Hyperplane

The core of a SVM is its hyperplane, however finding the optimal hyperplane is a non-trivial problem. There exist several methods for which the SVM's hyperplane can be optimized, but in this assignment we will be using *Stochastic Gradient Descent* with the *Hinge loss* to find the best parameters for our hyperplane.

a. Hinge loss
   i. The Hinge loss is a cost function used for maximum-margin classification. The loss function is defined as follows:
   $$J(y, t) = max(0, 1 - t * y)$$
   where y is the raw prediction from the classifier and t is the true class label for the sample.
   ii. The hyperplane is fit such that it minimizes the value of the cost function for the training samples. For the Hinge loss, this means that most, if not all, of the sample prediction values are greater than the margin distance. This produces a hyperplane with *hard margins* where prediction values are not to fall within the margin of the hyperplane.
   iii. To optimize for *soft margins*, we can add a regularization term to the cost function. Our regularization term needs to be designed to maximize the margin defined as $\frac{2}{||w||}$ (e.g. two divided by the vector norm of the weights). The cost function for soft margins then looks like:
   $$J(y, t) = max(0, 1 - t * y) + C * \frac{||w||}{2}$$
   where $\frac{||w||}{2}$ is the inverse of the margin and C is a tunable parameter that defines the weight of the regularization (e.g. C==0 optimizes for hard margins, C>0 optimizes for soft margins of varying 'softness').

b. Optimizing with SGD

Stochastic gradient descent is a simple algorithm that can be used to optimize our parameters. SGD takes small steps toward the optimal value of a cost function over a number of iterations by using the gradient (i.e. derivative) of the cost function. You will want to use *PyTorch* to automagically calculate the gradients of your Hinge loss calculation for you. This tutorial contains code snippets that may be helpful.

Steps for Implementation:

1. Initialize your parameter values.
2. Define your number of iterations (i.e. epochs) and learning rate.
3. For each iteration:
    a. Loop over all samples in your training data, and for each sample:
        i. Calculate the Hinge loss for the sample using PyTorch functions.
        ii. Multiply the loss gradients by your learning rate, and subtract the result from your parameters, w.
        iii. Reset the PyTorch gradients.

4. Classification

An SVM's hyperplane is limited to separating data into two classes. To address this limitation and build a classifier capable of handling multi-class dataset, there are two options: *One-vs-All* classification and *One-vs-One* classification. Both methods are similar to ensemble classification in that multiple classifiers must be trained to perform the method.

a. One-vs-All:
    i. Train one classifier for each of your classes.
    ii. Each classifier is trained to identify if a sample belongs to either the target class or one of the other classes (labeled as +1 for target class, -1 for any other class).
    iii. To perform multi-class classification, use each model to predict the sample. Use the target class for whichever model has the boundary farthest from the boundary (i.e. the y prediction that is the highest positive value).

b. One-vs-One:
    i. Train $\frac{classes * (classes-1)}{2}$ number of classifiers.
    ii. Each classifier is trained to identify if a sample belongs to one of two particular classes in the multi-class dataset (e.g. spyware or cryptoware).
    iii. To perform multi-class classification, use each model to predict the sample. Use voting to determine which class the sample belongs to.

c. This tutorial also explains the difference between the two techniques.

d. For this assignment, you only need to implement ***one*** of the two techniques.

5. Performance metrics

To analyze the performances of your implemented algorithm, you will need to include the following metric in your code:

- Accuracy = C / (C + W)
    - C = Correctly predicted if a sample is malware or not.
    - W = Incorrectly predicted if a sample is malware or not.

You should train a binary classifier (e.g. using only two labels, Malware vs Benign) when evaluating accuracy. *The OvA or OvO technique is not necessary here.*

- Include a normalized confusion matrix of your multi-class accuracy for the malware families.
    - When you train the multi-class classifier, you can exclude the benign samples and only compare malware families.
    - When presenting your confusion matrix, only include the top 20 malware families (by sample count).

***Tip**:* There is a very large number of benign samples in the dataset. You should reduce the number of samples you use to save time. *Be sure to report the number of benign samples you decided to use, in addition to the ratio between benign and malware samples.*

6. Hyperparameter Tuning

You should consider the following parameters in your tuning:

- C value
- number of iterations
- learning rate

Deliverables

1. All source code of your implementations.
    - Document your code! Your code documentation should demonstrate you understand what your code is doing (and that you did not just copy-and-paste from an external source).
2. A readme file that should contain:.
    - A clear description of the directions to set up and run your code.
    - If your code requires external libraries, or if not written in Python, provide the additional references/direction.

   *If your instructions are not clear, your work will **NOT** be graded!*
3. A short report describing your experiments and the results of your evaluations (~3 pages).

Expected Report Elements:

1. Report the number of benign samples you used (step 5), in addition to the ratio between benign and malware samples.
2. Performance
    a. Report the performance of your classifier based on its accuracy.
    b. Provide the confusion matrix (only include the top 20 malware families) that you generated in step 5.
    c. Discuss the results in your confusion matrix.
3. Hyperparameter tuning
    a. Describe your hyper-parameter tuning process and show figures (with proper labeling) describing interesting results.
4. Feature importance
    a. Feature importance can be ranked by using the value of the w vector for each feature. Show the 10 most and 10 least important features.
    b. Do you notice any trends or characteristics in the most important features?
    c. Would it be good to normalize the features?
5. SVM

    a. SVMs can be non-linearized with the use of kernel functions. Speculate on the possible advantages and disadvantages of using non-linear kernel functions, such as the radial basis function or polynomials, on this dataset.
    b. Use the sklearn implementations for LinearSVC and SVC to compare the real performance of a linear SVM and SVM with the 'rbf' kernel. *Use a smaller data set.*
    c. What does it mean if the magnitude of the *y* prediction value for a sample is < 1? **Hint**: there is a particular term used to describe these data points in a SVM.
    d. What feature about the loss function makes these samples particularly important?
6. **Bonus:**
    a. Implement both One-vs-One and One-vs-All classification.
    b. How does the performance between One-vs-One and One-vs-All classification compare based on your implementation?
    c. Speculate as to why one may work better than the other on this problem.
7. Include any citations in an appropriate and consistent format.
8. ***Not Expected*:** Background on the SVM, SGD, or the dataset. Especially if it leads you to copy other material (see "A Note on Plagiarism" below!).

Grading Rubric

Criteria	1 Poor	2 Basic	3 Proficient	4 Distinguished
Implementation (40%)	No or non-functioning code for anomaly identification techniques.	Code for one or both classifiers are functional, but one (or both) implementations have significant errors.	Minor errors or significant inefficiencies exist in implementation, but are otherwise mostly correct.	Techniques are perfectly correct. Code is cleanly written and handles most edge cases.
Experiments & Results (20%)	Minimal results are reported.	Results are incomplete and/or wrong, or are not presented in a way where its meaning is clear.	A reasonable set of results showing expected performance. Some presentation issues such as confusing graphs/tables or unnecessary detail.	Detectors achieve expected performance and all experiments are presented clearly.
Analysis & Questions (20%)	No or nonsensical analysis. Missing most questions.	Analysis is inaccurate or hard to understand. Some, but not all, questions are addressed.	Analysis is sensible and covers questions well. May be missing appropriate references or is weak in some areas.	Clear and accurate analysis that is backed up with appropriate references.
Writing Quality (10%)	Very poorly written and hard to follow.	Major points are visible, but writing may include many errors and/or lack focus and is disorganized.	Writing is clear enough to be understood, but some points may lack focus. Relatively few writing errors.	The paper is clear and well organized. Writing is smooth and polished with very few errors.
Code Documentation (10%)	Inadequate documentation provided both in the code and with the code.	Acceptable documentation for running the code, but lacking in-code documentation and helpful descriptions to demonstrate understanding.	Code includes some documentation (e.g. function docstrings, inline comments), however quality may be weak or unclear.	High quality documentation is provided. The purpose and function of all segments of code can easily be understood.

A: 3.25 average or higher
B: 2.5 - 3.25 average.
C: 1.75 - 2.5 average.

A Note on Plagiarism

When writing, you must include in-line citations whenever you are reporting on information that is not "general knowledge," i.e. anything you learned for this project and didn't know in advance. This is **NOT** just for quoted information. Failure to do this is plagiarism.

This article on plagiarism is good and covers the line between common knowledge and other material: https://writingcenter.unc.edu/tips-and-tools/plagiarism/. Also: https://www.plagiarism.org/ has a ton of additional information.

I have had students fail to follow these guidelines and get caught nearly every time I've taught my research seminar. These students get put on probation and have even been suspended from the university for this serious academic violation. *Please do not be the next!*