CSCE 633 - Homework Assignment 2

Nelson Dsouza - UIN: 129007095

Kiran Kondisetti - UIN: 430000208

Repository: GitHub

Aggie Code of Honor

"An Aggie does not lie, cheat, or steal or tolerate those who do."

Name: Nelson Dsouza UIN: 129007095

Name: Kiran Kondisetti UIN: 430000208

1. Web Material Consulted:
   1.1. https://pypi.org/project/decision-tree-id3/
   1.2. https://scikit-learn.org/stable/modules/cross_validation.html
   1.3. https://machinelearningmastery.com/implement-perceptron-algorithm-scratch-python/
   1.4. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html#sklearn.linear_model.Perceptron
   1.5. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html#sklearn.ensemble.BaggingClassifier
   1.6 https://matplotlib.org/api/pyplot_summary.html

I certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received nor given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment.

Date: 2/11/2020

Signature:
Nelson Michael Dsouza
Kiran Kondisetti

For the entire assignment, the normalization technique from assignment 1 was used.

Task 1 - Decision Tree for Attack Detection

Library Used: ID3 - Link

This library is compatible with sklearn - in that it uses the same method structure for its classifier.
Ex. *fit()*, *predict()* etc.

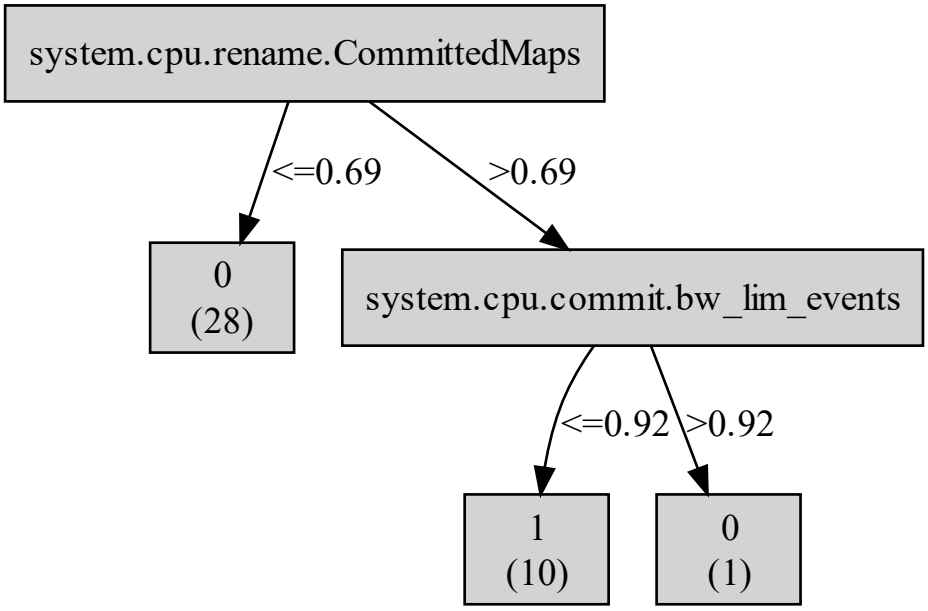
The classification criterion is information gain

This model can be selected by setting the model parameter in dt_plus_ensemble.py to 0 (model = 0)

3 fold cross validation is done by using the StratifiedKFold library in sklearn In contrast to KFold, StratifiedKfold maintains class ratios in the folds created (avoid having folds that only have benign samples in our case). Here 39 samples (30 benign and 9 malignant) would be split so as to have the same number of samples of each class in every fold (3x (10 benign and 3 malignant))

The resultant tree is just 2 features deep and correctly classifies the entire data set, this can be visualized through the tree.dot file. (I did try to do some file processing to output the features in the format suggested in the assignment, but due to a lack of time, I could not complete this bit.)

The tree.dot file is pretty readable and does provide an output which is intuitive to understand.



Where 1 represents malignant and 0 represents benign

Since the trees are really small, we considered splitting the feature set into their base classes, for instance:

```
system.cpu.commit.membars
system.cpu.commit.loads
system.cpu.commit.int_insts
system.cpu.commit.function_calls
system.cpu.commit.committedOps
system.cpu.commit.committedInsts
system.cpu.commit.bw_lim_events
system.cpu.commit.branches
system.cpu.commit.refs
system.cpu.commit.op_class_0::total
system.cpu.commit.op_class_0::MemWrite
system.cpu.commit.op_class_0::MemRead
system.cpu.commit.op_class_0::IntAlu
```

And ran the ID3 algorithm on these splits. This can be done by setting model = 3. Here, the ID3 classifier tries to classify the dataset using these class splits. If the tree cannot correctly classify the entire data set, a prompt is output stating that the feature class does not completely classify the data.

The motivation for doing this is to find simpler trees that can be achieved from statistics collected from just one part of the processor (for instance - commit). As it turns out, a few of these splits do achieve this feat and correctly classify the dataset.

The trees thus created can be found in the 'data' folder where this script is run. The cross validation scores are then calculated for these trees and printed onto the console

Accuracy: 0.90 (+/- 0.04)

Task 2 - Perceptron for Attack Detection

We used 2 flavors of perceptron to experiment with different perceptron algorithms

1. Perceptron - Link
2. Sklearn Perceptron

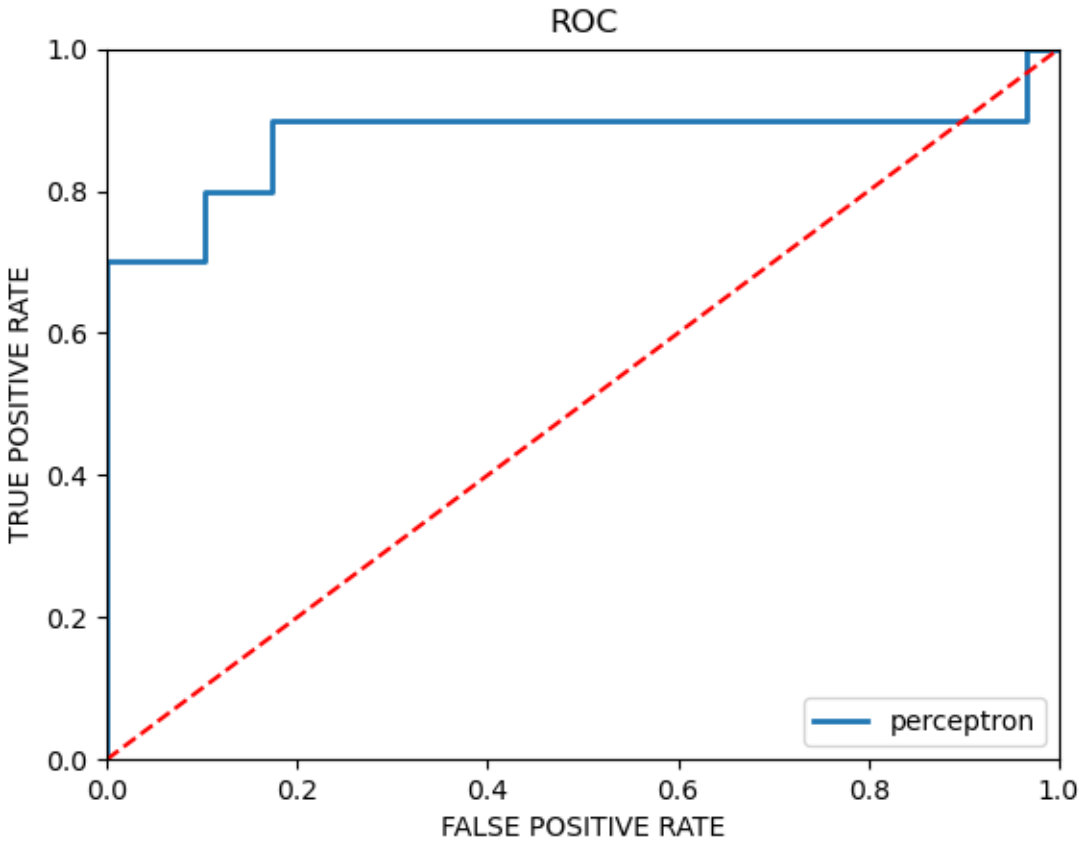
For the sake of better transparency and understanding of the underlying code, we have worked more on the first code base (set model = 1). Alternatively the sklearn perceptron can also be used (set model = 2). The functions were modified to work for our needs. The perceptron takes 2 input parameters: a learning rate (l_rate) and the number of epochs to use during training (n_epoch). After some experimentation and tuning, in our case, we got the used the following parameters: l_rate = 1 and n_epoch = 11.

The weights were then printed in the following files:

1. Perceptron-sorted.txt
2. Perceptron-absolute-sorted.txt

Accuracy: 0.90 (+/- 0.07)

To plot the ROC curve, we modified the predict function to also take a threshold parameter. Using this function we calculated the confusion matrix of the data set for the given threshold and calculated the true positive rate and the false positive rate. After some tweaking get the following ROC curve was achieved.



We used the matplotlib library to plot this curve. Link

The k-fold crossvalidation performed is similar to the one in ID3.

The perceptron was then re-trained with the threshold value corresponding to the elbow of the ROC curve (thresh = -9.428)

The crossvalidation metrics remained the same. The new weights were then printed in the following files:

1. Perceptron-sorted_roc.txt
2. Perceptron-absolute-sorted_roc.txt

Accuracy: 0.87 (+/- 0.10)

Task 3 - Ensemble Methods for attack detection

The primary idea for this stemmed out of the different types of statistics that the data provided, as an extension of the aforementioned split decision trees for task 1, we used the all of the resultant trees from the splits considered; including the ones that did not completely classify the data. We effectively have 18 splits.

The split categories are:

```
cc_regfile
decode
int_regfile
misc_regfile
branchPred
committed
fetch
rob
icache
workload
dtb
rename
memDep0
commit
dcache
iq
iew
```

Each of these will create a tree that can be seen in the data/ directory. The prediction of these trees is used to train the perceptron.

For each of these trees, we formed the dataset to include only the features that belong to the set, in our layout, this involved retaining only those columns. With these new data sets, 3 fold cross validation was performed - for each of the folds, the estimator was trained with 2 folds and and the predictions were used to create a new data set to train a perceptron as a means to aggregate the results of the individual trees. The training data thus formed contains 39 rows and 18 columns (1 for each tree). Note that since the outputs of the decision trees is either a 1 or a 0, the contents of the new matrix are binary.

The trained perceptrons accuracy was then evaluated with the remaining test fold and the accuracy numbers are as follows:

Accuracy: 0.95 (+/- 0.04)

The accuracy achieved in this case is better than both the individual decision trees and the perceptron models by themselves.

As an additional experiment, we also did some experiments with bagging which is available as a library in sklearn. This can be selected by setting model = 4. This function only used id3 and not the perceptron.

Accuracy: 0.87 (+/- 0.04)

This is worse than the id3 without bagging.

Github link- https://github.com/kirankondisetti/CSCE633_S20-KIRAN_HW2