KOÇ UNIVERSITY

COMP 430/530: Data Privacy and Security – Fall 2024

Homework Assignment #3

PART 1: IMPLEMENTATION

You are given the Banknote Authentication dataset (**BankNote_Authentication.csv**) which contains data from genuine vs forged banknote specimens. The following 4 features were extracted from the images of the banknotes: **variance**, **skewness**, **curtosis**, **entropy**. They correspond to the first 4 columns of the csv file. The last column, called **class**, is the label. It indicates whether that banknote is genuine or forged.

You are also given skeleton code in Python to read the dataset and split it into training and test sets (70%-30%). We provide sample code for building 3 supervised ML model types: **Decision Tree (DT)**, **Logistic Regression (LR)**, **Support Vector Classifier (SVC)**. The parameters for each of these models are given, e.g., max_depth=5 for DT, penalty=l2 for SVC, and so forth. **In the upcoming questions, you should use these same parameter values when building new models.**

Implement your solutions to the following questions in the Python file. Provide experiment results and discussion in a separate pdf report.

Question 1: Label Flipping Attack [10 pts]

Simulate a label flipping attack by implementing the function:

attack_label_flipping(X_train, X_test, y_train, y_test, model_type, p)

- X_train and X_test: features of the training and test data, respectively
- y_train and y_test: labels of the training and test data, respectively
- model_type: which model will be attacked? one of “DT”, “SVC”, “LR”
- p: percentage of data for which label flipping should take place

Here is a sample function call:

attack_label_flipping(X_train, X_test, y_train, y_test, "DT", 0.05)

In this case, the function should simulate a label flipping attack by randomly flipping the labels of 5% of the training data before building a model, and return the accuracy of the resulting model. To account for the randomness in which labels are flipped, the function should internally repeat the experiment 100 times and average the accuracy results. (The averaged result is returned.)

Using your function, simulate label flipping attacks with p = 5%, 10%, 20%, 40% and for all three ML models: DT, LR, SVC. Example output:

```
################################################
Label flipping attack executions:
Accuracy of poisoned DT 0.05 : 0.9679854368932039
Accuracy of poisoned DT 0.1 : 0.9579368932038835
Accuracy of poisoned DT 0.2 : 0.9292475728155342
Accuracy of poisoned DT 0.4 : 0.7903398058252427
Accuracy of poisoned LR 0.05 : 0.9792233009708738
Accuracy of poisoned LR 0.1 : 0.9740533980582525
Accuracy of poisoned LR 0.2 : 0.970242718446602
Accuracy of poisoned LR 0.4 : 0.9540776699029127
Accuracy of poisoned SVC 0.05 : 0.9535922330097089
Accuracy of poisoned SVC 0.1 : 0.949393203883495
Accuracy of poisoned SVC 0.2 : 0.9458252427184466
Accuracy of poisoned SVC 0.4 : 0.7338349514563106
################################################
```

In your report, provide the experiment results in a table. Briefly interpret and discuss your results: What is the accuracy impact of p? Does the attack affect all 3 ML models equally or is there a model that is more robust to the attack?

Question 2: Defense Against Label Flipping [15 pts]

You would like to explore the effectiveness of an outlier detection-based defense against label flipping attacks by identifying flipped data points. You will implement the function:

label_flipping_defense(X_train, y_train, p)

- **X_train:** Features of training dataset.
- **y_train**: Labels of training dataset.
- **p**: percentage of data for which label flipping should take place.

Within this function, simulate an attack and defense:

1. First perform a randomized label flipping attack (similar to the previous question) using the given **X_train**, **y_train**, and **p.**
2. Then, assuming the role of a defender, design and implement an outlier detection-based defense. You can use an outlier detection algorithm of your choice (e.g., scikit-learn Local Outlier Factor, Isolation Forest) or come up with your own algorithm.

   Hint: You may want to use a heuristic such as: “if the current data point is surrounded by samples of the opposite class, then it is an outlier, therefore probably it was flipped”.

3. Apply your defense against the poisoned training dataset (result of step 1). Your defense correctly identifies how many of the flipped data points? Print this result to the console (standard I/O).

The code that is given to you will call the **label_flipping_defense** function with **p = 5%, 10%, 20%, 40%** and expect it to print a message indicating how many of the flipped data points are correctly identified by the defense.

In your report, explain how you designed your defense, i.e., what is the intuition behind it? How did you choose the parameters of your defense? Is your defense effective? We do not expect

your defense to be always 100% effective (i.e., it correctly identifies ALL flipped points) but it should be reasonably effective, e.g., >40%.

Example output of the function:

```
##################################################
Label flipping defense executions:
Results with p= 0.05 :
Out of 48 flipped data points, 24 were correctly identified.
Results with p= 0.1 :
Out of 96 flipped data points, 54 were correctly identified.
Results with p= 0.2 :
Out of 192 flipped data points, 86 were correctly identified.
Results with p= 0.4 :
Out of 384 flipped data points, 196 were correctly identified.
```

Question 3: Evasion Attack [20 pts]

Implement an evasion attack in the following function:

evade_model(trained_model, actual_example)

- **trained_model**: a model that is already trained and available to the attacker, white-box (e.g., one of myDEC, myLR, mySVC)
- **actual_example**: modify the features of this data point so that it is misclassified by the trained_model

If **actual_example** is originally classified as **1** by **trained_model**, then **evade_model** should modify it such that the modified version is classified as **0**. If **actual_example** is originally classified as **0**, then **evade_model** should modify it such that the modified version is classified as **1.** The return value of **evade_model** is the modified version.

While achieving evasion, you should aim to minimize the amount of perturbation (difference between **actual_example** and **modified_example**). The amount of perturbation is calculated by the **calc_perturbation** function given to you; please inspect it before formulating your evasion attack strategy.

Two important rules:

1. Your evade_model function must **guarantee successful evasion**, i.e., given an actual example, it should **always** be able to find an example that evades successfully.
2. There are multiple possible attack strategies and no single correct answer. All feasible attack strategies will be accepted, as long as their average perturbation amount (computed across 40 examples by the **main** function) remains lower than **3** for each individual model (DT, LR, SVC).

In your report:

- Briefly describe your attack strategy. Figures or images are welcome.
- State the average perturbation amount that is printed out to the console by the code

given to you for all models. For example, here are the results for our own evasion attack implementation:

```
Avg perturbation for evasion attack using DT : 0.7737499999999999
Avg perturbation for evasion attack using LR : 0.844375
Avg perturbation for evasion attack using SVC : 0.8756249999999998
```

Since all perturbation amounts are lower than 3 and successful evasion is achieved for all examples, this attack would receive full points.

Question 4: Evasion Attack Transferability [10 pts]

Your goal is to measure the cross-model transferability of the evasion attack you implemented in the previous question. To that end, implement the following function:

evaluate_transferability(DTmodel, LRmodel, SVCmodel, actual_examples)

- DTmodel, LRmodel, SVCmodel: the three ML models
- actual_examples: 40 actual examples (not adversarial) that will be used in your transferability experiments

Inside this function, you should design and perform the necessary experiments to answer the following questions (use your **evade_model** function):

- Out of 40 adversarial examples you craft to evade DT, how many of them transfer to LR? How many of them transfer to SVC?
- Out of 40 adversarial examples you craft to evade SVC, how many of them transfer to LR? How many of them transfer to DT?
- Out of 40 adversarial examples you craft to evade LR, how many of them transfer to DT? How many of them transfer to SVC?

Include the results of your experiments in your report. Based on your results, do you think your evasion attack has high cross-model transferability? Discuss briefly.

Example output:

```
##################################################
Transferability of evasion attacks:
Out of 40 adversarial examples crafted to evade DT :
-> 20 of them transfer to LR.
-> 19 of them transfer to SVC.
Out of 40 adversarial examples crafted to evade LR :
-> 22 of them transfer to DT.
-> 18 of them transfer to SVC.
Out of 40 adversarial examples crafted to evade SVC :
-> 23 of them transfer to DT.
-> 19 of them transfer to LR.
##################################################
```

Question 5: Backdoor Attack [15 pts]

Implement a backdoor attack in the function:

backdoor_attack(X_train, y_train, model_type, num_samples)

- X_train: features of the training data
- y_train: labels of the training data
- model_type: which model will be attacked? one of “DT”, “SVC”, “LR”
- num_samples: number of backdoored samples to inject

Internally, this function should perform the following steps.

1. Inject **num_samples** number of samples to the training data containing the trigger pattern of your choice.
2. Train a backdoored model depending on the **model_type**, i.e., DT or SVC or LR.
3. Design an appropriate experiment to measure the Success Rate of your backdoor attack. You must decide what experiment is suitable and how Success Rate should be defined and measured.
4. Return the Success Rate of the attack:

```
Success rate of backdoor attack: 0.0 model_type: SVC num_samples: 0
Success rate of backdoor attack: 0.3 model_type: SVC num_samples: 1
Success rate of backdoor attack: 0.6 model_type: SVC num_samples: 3
Success rate of backdoor attack: 0.9 model_type: SVC num_samples: 5
Success rate of backdoor attack: 1.0 model_type: SVC num_samples: 10
```

In your report, provide the following:

- A table that summarizes Success Rate of your backdoor attack for the three model types (DT, SVC, LR) and for num_samples = 0, 1, 3, 5, 10.
- Explain what your trigger pattern is and how you injected it. [up to -5 pts if not answered]
- Explain how you mathematically define the **Success Rate** metric for your backdoor attack and how you design the experiment for measuring it. [up to -10 pts if not answered or answered incorrectly]

Question 6: Model Stealing [10 pts]

Simulate a model stealing attack by implementing the function:

steal_model(remote_model, model_type, examples)

- **remote_model:** Assume this is the remote model that the attacker is trying to steal. Attacker queries this model and obtains responses.
- **model_type**: “DT” for decision tree, “LR” for logistic regression, “SVC” for support vector classifier. For simplicity, assume that the attacker knows the parameters of the models:
  - For DT: max_depth = 5, random_state=0
  - For LR: penalty = ‘l2’, tol=0.001, C=0.1, max_iter=100
  - For SVC: C=0.5, kernel=’poly’, random_state=0
- **examples**: Attacker uses this list of unlabeled examples for querying the model and steals the model based on the responses to these examples.

The code that is given to you will call the **steal_model** function and expect it to return the stolen model. As the number of examples grows, we expect the stolen models to become more accurate in general. For example:

```
Number of queries used in model stealing attack: 5
Accuracy of stolen DT: 0.493
Accuracy of stolen LR: 0.862
Accuracy of stolen SVC: 0.641
Number of queries used in model stealing attack: 50
Accuracy of stolen DT: 0.898
Accuracy of stolen LR: 0.964
Accuracy of stolen SVC: 0.755
Number of queries used in model stealing
attack: 200 Accuracy of stolen DT: 0.961
Accuracy of stolen LR: 0.981
Accuracy of stolen SVC: 0.782
```

Conduct an experiment to measure the accuracies of stolen DT, LR and SVC models for varying numbers of examples: [5, 10, 20, 30, 50, 100, 200]. Include a table of your results in your report. Briefly discuss your results - which model is easiest or hardest to steal? Under what conditions?

PART 2: READING [total: 20 pts, each question worth 5 pts]

Read the paper "Exploiting Machine Learning to Subvert Your Spam Filter" by Nelson et al. This is one of the very first works on attacking machine learning-based spam filters. Answer the following questions based on this paper.

(a) As stated in Section 3.4, an enabling observation behind the attack is that: *"the spam scores of distinct words do not interact; that is, adding a word w to the attack does not change the score f(u) of some different word u≠w"*. Based on the explanation of the learning method in Section 2.3 and the equations therein, explain how the authors arrive at this observation.

(b) Describe how the above observation is used when generating an attack payload/strategy.

(c) The authors propose two defenses against their attack: RONI and Dynamic Threshold. We learned about RONI in the lectures, but not the Dynamic Threshold defense. Explain how this defense works and why it is effective.

(d) In the lectures, we discussed that "outlier detection" is also a potential defense strategy. How can outlier detection be used to defend against this paper's attack? In particular, the authors admit what aspect/feature of the attack payload can make it possible for the attack to be detected (which you can use in your outlier detection defense)?

SUBMISSION

When you are finished, submit your assignment via LearnHub.

- Move all of your relevant files into a folder named `**your KU Login**`.
- **Compress this folder into a zip**. (Do not use compression methods other than zip.)
- Upload your zip file to LearnHub.

Notes and reminders:

- After submitting, download your submission and double-check that: (i) your files are not corrupted, (ii) your submission contains all the files you intended to submit, including all of your source code and your report. If we cannot run your code because some of your code files are missing, we cannot give you credit!
- This homework is an **individual assignment**. All work needs to be your own. Submissions will be checked for plagiarism (including comparing to previous years' assignments).
- **Your report should be a pdf file.** Do not submit Word files or others which may only be opened on Windows or Mac (or opening them may remove table/figure formatting).
- Only LearnHub submissions are allowed. Do not e-mail your assignment to the instructor or TAs.
- Do not change the names or parameters of the functions that we will grade.
- If your code does not run (e.g., syntax errors) or takes an extremely long amount of time (e.g., it takes multiple hours whereas our reference implementation takes 2-3 minutes), you may get 0 for the corresponding part.

GOOD LUCK!