KOÇ UNIVERSITY

COMP 430/530: Data Privacy and Security – Fall 2022

Homework Assignment #3

PART 1: READING [total: 40 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)?

Next, read the paper "Functionality-Preserving Black-Box Optimization of Adversarial Windows Malware" by Demetrio et al. This is a much more recent work (published last year); it is about creating adversarial examples against ML-based Windows malware detectors. Answer the following questions based on this paper.

(e) The following optimization problem is presented as the attack formulation:

$$\underset{s \in \mathcal{S}}{\text{minimize}} \;\; F(s) = f(\boldsymbol{x} \oplus \boldsymbol{s}) + \lambda \cdot C(s),$$
$$\text{subject to } q \leq T.$$

In your own words, explain what this optimization is trying to achieve. You must explain the meanings of the terms and notation (such as s, S, F(.), f(.), x, $\oplus$, ƛ, C(.), q, T) in plain English if necessary in your answer.

(f) What is meant by “query-efficiency” and “functionality-preserving” in the context of this paper? Why are these two properties important and desired?

(g) According to the results in Figure 4, under the same attack size, higher number of queries decreases detection rate. Why is this an intuitive result? Explain by discussing: (i) what detection rate, attack size, number of queries mean, and (ii) how you would expect attack size and number of queries to affect detection rate in the context of this paper’s attack.

(h) Do the attacks on commercial antivirus software leverage the concept of transferability? Why or why not?

PART 2: IMPLEMENTATION [total: 60 pts]

You are given the Forest Fires dataset (**forest_fires.csv**) which contains data from possible forest fire instances. The following 10 features were extracted from the collected data:

- Temperature : temperature noon (temperature max) in Celsius degrees: 22 to 42
- RH : Relative Humidity in %: 21 to 90
- Ws :Wind speed in km/h: 6 to 29
- Rain: total day in mm: 0 to 16.8
- Fine Fuel Moisture Code (FFMC) index: 28.6 to 92.5
- Duff Moisture Code (DMC) index: 1.1 to 65.9
- Drought Code (DC) index: 7 to 220.4
- Initial Spread Index (ISI) index: 0 to 18.5
- Buildup Index (BUI) index: 1.1 to 68
- Fire Weather Index (FWI) index: 0 to 31.1

They correspond to the first 10 columns of the csv file. The last column, called **class**, is the label. It indicates whether a forest fire has happened or not.

You are also given skeleton code in Python to read the dataset and split it into training and test sets (80%-20%). 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 LR, 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, n)

- 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”
- n: 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 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 n = 5%, 10%, 20%, 40% and for all three ML models: DT, LR, SVC. In your report, provide the experiment results in a table. Briefly interpret and discuss your results: What is the accuracy impact of n? Does the attack affect all 3 ML models equally or is there a model that is more robust to the attack?

Question 2: Membership Inference Attack [10 pts]

Simulate a membership inference by implementing the function:

inference_attack(trained_model, samples, t)

- trained_model: in this question, only SVC model is used
- samples: samples which are actually from the training dataset
- t: threshold probability value

For each sample, this function gets the confidence scores of the prediction (use *predict_proba* method). The confidence value of the predicted class is compared against *t* to make a decision about whether the sample was included in the training data. The function measures and returns the **recall** of inference, calculated as follows.

Define:

- TP = Sample is a member of the training set, the attack correctly infers that it is a member of the training dataset.
- FN = Sample is a member of the training set, the attack incorrectly infers that it is not a member of the training dataset.

Then, recall is defined as:

$$\text{Recall} = \frac{TP}{TP + FN}$$

In your report, provide the following:

- The recall values for t = [0.99,0.98,0.96,0.8,0.7,0.5] in a table or graph.
- Explain what recall means for this attack and why the recall values are changing in the trend that you observe.

Question 3: Backdoor Attack [10 pts]

Simulate a backdoor attack by implementing 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:

- Inject **num_samples** number of samples to the training data containing the trigger pattern of your choice.
- Train a backdoored model depending on the **model_type**, i.e., DT or SVC or LR.
- 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.
- Return the Success Rate of the attack.

Here is the result of a sample execution (with SVC):

```
Success rate of backdoor: 0.16 model_type: SVC num_samples: 1
Success rate of backdoor: 0.56 model_type: SVC num_samples: 3
Success rate of backdoor: 0.99 model_type: SVC num_samples: 5
Success rate of backdoor: 0.99 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. [-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. [-5 pts if not answered or answered incorrectly]

Question 4: Evasion Attack [10 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 (called **adversarial_example** in the code given to you) would be classified as **0**. If **actual_example** is originally classified as **0**, then **evade_model** should modify it such that the modified version would be 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 **adversarial_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 evasion, i.e., given an actual example, it should **always** be able to find an adversarial example.
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) remains lower than the following values:
    - Decision Tree: 2
    - Logistic Regression: 3
    - SVC: 4

In your report:

- Briefly describe your attack strategy. Figures and/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 evasion attack implementation:

```
Avg perturbation for evasion attack using DT : 0.8555750000000154
Avg perturbation for evasion attack using LR : 1.2536750000000205
Avg perturbation for evasion attack using SVC : 1.48690000000003
```

Question 5: Transferability of Evasion Attacks [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 (using 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.

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=1000
  - For SVC: C=0.5, kernel=’poly’, random_state=0
- **examples**: Attacker uses this list of unlabeled samples for querying the model and steals the model based on the responses to these samples.

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.

Conduct an experiment to measure the accuracies of stolen DT, LR and SVC models for varying numbers of examples: [8, 12, 16, 20, 24]. Include a table containing your results in your report. Briefly discuss what you observe from your results.

SUBMISSION

When you are finished, submit your assignment via Blackboard:

- Move all of your relevant files and your report into a folder named `**your KUNet ID**`.
- **Compress this folder into a single zip file**. (Don't use compression methods other than zip.)
- Upload your zip file to Blackboard.

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!
- You must upload your code in py files**, we do not accept Python notebooks (.ipynb extension)**.
- 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 Blackboard submissions are allowed. Do not e-mail your assignment to the instructor or TAs.
- **Do not submit any data files**.
- 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!