**Eth. AI, HW4, Adversarial Attacks**

**Due Week 11, Thursday, 11:59 PM**

**Goal:** Create both an untargeted and targeted character level adversarial attack and test them on a trained model.

**Resources:** Tutorials on python and sklearn library can be found both in [W3Schools](https://www.w3schools.com/python/default.asp) as well as [here (Python course Slides](https://ind657-my.sharepoint.com/:f:/g/personal/jrusert_pfw_edu/Ei_EBaJU-IJBgpwM2A7mmvUB-GrO9jUSvSq0CgMgwZgwYw?e=DJcAgX)). Also, sklearn has demonstrations of models and functions on the specific model pages as well (see below links).

**Models:** You will be working with 1 type of model for this homework, 1 from the sklearn python library:

[Logistic Regression](https://scikit-learn.org/1.5/modules/generated/sklearn.linear_model.LogisticRegression.html)

This homework consists of 4 parts, but all parts use the same dataset for attack and testing provided on brightspace. Even though there are 4 parts, there still should only be 1 final report and 1 .py file handed in.

## Part 1: Prepare the Classification Model

**Goal:** This part is the simplest of the homework, as you are simply training a Logistic Regression model and Vectorizer on the given Rotten Tomatoes training dataset.

**Coding goals:**

* Use the sklearn library to train a logistic regression model on the rotten tomatoes training set (train\_rotten-tomatoes.csv).
  + You should use the [tfidf-vectorizer](https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) to automatically convert the text to a vector form.
* Produce a score on the training data.
* Test and produce a score on the test dataset (test\_ rotten-tomatoes.csv).
* This should all be contained in its own function which returns the classifier and vectorizer.

**To be added to the report:**

* Describe the simple process used to train/test the logistic regression models.
* Report the scores for both the training data and the test data in a small table.
* Describe any difficulties faced during the implementation of the model.

## Part 2: Prepare Character Level Modification

**Goal:** This part creates a function which will be used in both the untargeted and targeted attack. Simply put, you are creating a function which takes in a word and applies character level modifications to it and returns it. For example, if the word is “great” then it may return “gr3@t” or “gre at” or “graet”. You will be choosing what character level modifications will be made.

**Coding Goals:**

* Create a function **character\_modify(word)** which takes in a single word and modifies it at the character level and then returns that modified word.
* The character level modifications are up to you, but should involve at least 2 different types of modification. The function should randomly choose between these modifications for the given word (only 1 per word, but a random chance of any of them being chosen).
  + Some character level modifications are:
    - White space addition, “great” -> “gre at”, “g r e a t”, “g r eat”, etc.
    - Character swapping, “great” -> “graet”, “gerat”, etc.
    - Character substitution, “great” -> “gr3@t”, “gr3a+”, etc.
    - Character deletion, “great” -> “gret”, “geat”, etc.
    - Character addition, “great” -> “greeat”, “grrrrreat”, etc.
* The character level modifications should also involve some randomness. For example, if you swap characters in a text, do so randomly (utilizing the random library). Or if you insert white space, do randomly. This random rate is up to you, but should be no more than 0.5 as to not modify too much of the word.

**To be added to the report:**

* Description of the 2 or more character level modifications chosen and how they were implemented.
* The random values used and how you came to the decision to use those values.
* 3 Example outputs of your function run on the following words (note if all 3 outputs look the same, then the randomness was not implemented correctly):
  + great
  + movie

## Part 3: Untargeted Attack

**Goal:** This part is simple now that you have completed Part 2. You will create an untargeted attack function, **untargeted\_attack(text)**, which takes in a single text or document and randomly applies the character level modification to each word in it. Each word should have a 40% chance of modification, if it is chosen for modification, it should be passed to the previous function and the return value replaces that word in the original text.

**Coding Goals:**

* Create the above function which takes in a text, splits it by whitespace, and randomly applies the character level modification function (at a rate of 40%), finally returning the modified text.
  + This will be a simple function overall, as the previous function will be doing most of the heavy lifting.
* In a different function, run the untargeted attack on the test data.
  + For each text, pass it to the untargeted attack function.
  + Write out the modified texts into a new file: **untargeted-attack\_rotten-tomatoes.csv**
    - This should have the same format as the test file, however, the text column should now contain the modified text (label column stays the same).
* Test the Part 1 model on the modified test examples and obtain the score. Run the attack 5 times and take an average of the scores.

**To be added to the report:**

* Brief description of your implementation of each function implemented.
* A table of the models scores on the original, umodified texts and the (attacked) modified texts.
* Observations on how the attack performed given the above table.
* Randomly choose 5 texts from the modified texts (don’t just take the first 5) and add these to the report.
* Make observations on:
  + Readability of the texts
  + How well they chose the “correct” words during the attack (correct here means the words you believe the classifier would be relying on to make its classification)

## Part 4: Targeted Attack

**Goal**: Your final goal is to create a version of the previous attack which is more precise in choosing which words to modify. Rather than randomly choosing words (as above), this attack will use the classifier itself for feedback to find the words it is relying on. This function, **targeted\_attack(text, model, vectorizer, orig\_label)**, will function the same as the untargeted and take a text and modify and return it. However, it also needs the target model, vectorizer, and original label to be passed into it.

* The targeted attack will consist of 2 main parts, word selection and word modification.
* Word selection should be done via greedy select (algorithm below). Simply put, greedy select using the classifier to retrieve probabilities of when a word is left out of the text. The word which causes the classifier’s probability to drop the most, is seen as the first word to modify. The second highest would be second and so on.

|  |
| --- |
| Greedy Select Original\_probability = classifier(original\_text)[original label]  For each word in the target text:    Text\_minus\_word = text with current word removed  New\_probability = classifier(text\_minus\_word)[original label]  Probability\_drop = original\_probability – new\_probability    Add probabilty\_drop (linked with position) to list or dictionary  **Sort obtained list/dictionary by largest probability drop, pass to modification step** |

* For word modification, the attack should simply sort and use the list (or dictionary) given from the selection step to start modifying words. That is, the word that caused the highest probability drop should be modified via the character level modification function (Part 2) and replace the original word in the original text. This new text should then be checked against the classifier. If the classifier fails (no longer predicting the correct, original label), then the attack ends and the text is returend. If not, the next word is modified and text is checked and so on until the classifier fails.

**Coding Goals:**

* Implement the above function as described. Clearly make each part (selection and modified step) via comments.
* In a different function, run the targeted attack on the test data.
  + For each text, first verify that the LR model is predicting it correctly. If not, do not pass it to the targeted attack function (simply write it to the output file as is.)
  + Otherwise, pass the text to the targeted attack function, along with the LR model, vectorizer, and original label.
  + Write out the modified texts into a new file: **targeted-attack\_rotten-tomatoes.csv**
    - This should have the same format as the test file, however, the text column should now contain the modified text (label column stays the same).
* Test the Part 1, LR, model on the modified test examples and obtain the score. Run the attack 5 times and take an average of the scores.

**To be added to the report:**

* Brief description of your implementation of each function implemented.
* A table of the models scores on the original, umodified texts and the (attacked) modified texts as well as the average score from the untargeted attack.
* Observations on how the attack performed given the above table and why one attack may have performed better than the other.
* Randomly choose 5 texts from the modified texts (don’t just take the first 5) and add these to the report.
* Make observations on:
  + Readability of the texts
  + How well they chose the “correct” words during the attack (correct here means the words you believe the classifier would be relying on to make its classification)

## To Turn In:

* Python file (**USERNAME\_HW4.py**.) which contains functions related to the above parts. You should include comments and useful function names to differentiate the testing portions.
* Report which contains:
  + The above requirements for each part. Make sure you make it clear (via titles or subheadings) which part you are describing.
  + Discussion of issues or difficulties encountered.
  + A note of any AI tools used in coding or writing.
* Your two produced adversarial attack files. (Even though you will be running the functions multiple times, you only need to hand in 1 for each).

**Additional Rules (MUST BE FOLLOWED):**

1. All Homeworks should follow the overall [homework guidelines](https://ind657-my.sharepoint.com/:p:/g/personal/jrusert_pfw_edu/EZxf1ZsRXjBEkQLNhEhdTgUBt6U64KiT1DJ1YHtkARgKwA?e=bwHrbf)
2. The code should be written in python 3.
3. If noted, the functions must follow the naming and number of arguments as demonstrated.
4. You should make your code modular to the different steps. (You may have more functions to help your main functions)
5. You should be adding comments to document your code. **If I can’t understand why you perform an action, then I can’t credit you for performing that action.**
6. The report should be readable and reference your code, **without explicitly including code.**
7. You should include your name and homework number in the comments at the beginning of the python file.

**Report**

The reports for the homeworks are necessary to communicate your learning and thinking through of the material. Examples of good reports can be found on brightspace under Additional Resources/Guides. Note that your report style may differ, but it is a good reference to start with.

**Grading**

Assignment will be graded as follows:

|  |  |
| --- | --- |
| **Description** | **Points** |
| **(.py file)** Part 1 and 2 coding requirements | 5 |
| **(.py file)** Part 3 and 4 coding requirements | 15 |
| **(Report)** Part 1 and 2 report requirements | 5 |
| **(Report)** Part 3 and 4 report requirements | 15 |
| **(Report)** Other Report Requirements | 5 |
| **(.py file)** Documentation/Code (Comments, functions, etc) | 5 |
| **Total:** | **50** |

* **If the code does not run, it cannot be graded well.** (Many points can be lost if the code cannot be run, as I will not be able to fully test the implementation of the functions).
* **Breaking of the additional rules can result in applied penalties.** (Always make sure you are checking against the rules)

**Suggestions**

* **Documentation is key for showing your effort in this homework.** Make sure you are noting why you make certain decisions all throughout your code.
* The slides for previous classes are posted, so please refer to these and the book for ideas during implementation.
* Start simple, build up complexity. You should always make sure your new ideas being added do not cause your program to crash. So starting simple is the best way to a) maintain the ability to keep your code running, b) add in comments for documentation and thought process as you add more code.
* Work through the homework yourself, rather than sharing ideas (especially not code) with other students. **As a reminder, plagiarism (or sharing) of code is strictly prohibited.** This assignment is complex enough that significant overlap between students will be suspicious.
* If you have not worked with python before, w3schools can help you translate your previous coding experience to python (<https://www.w3schools.com/python/default.asp>)
* Stop by office hours to discuss ideas. I am always happy to help you think through your process!