Ethical Ai – Adversarial Attacks

Part1.

Functions used:

def workingWithThePartOne(df, model, trainOrTest='train', tfidf=None):

1. This function takes in the df which is dataframe of either the train data or the test data for the part one. The second it takes the model which is LR model. I am passing the model as an argument here so that I can pass the model if the model was trained earlier and return the trained model if it wasn't trained earlier. Then trainortest is used to define if the function will work for the training of the model or testing of the model.
2. It also passes the tf-idf embeddings which are used then we're testing the model, by passing the embeddings which we had trained the model earlier on.
3. This function was very basic, which I've been doing for a while now. In this function I've implemented the part 1 very easily, where I'm transforming and fitting the tf-idf vectorizer model in the same function.
4. The function trains the model with the embedding when we're training the model and passing the trainOrTest as train. The function then calculates and prints the accuracy, precision, recall and f1 score for the model for the train data in the same function.
5. For train the function then returns the LR model and also the embeddings.
6. For the test part of the function, we then pass the LR trained model and embeddings which have been fit earlier.
7. The model then calculates the accuracy, prevision, recall, f1 score and confusion matrix based on the test data we sent through the function argument.
   (FYI – the training data frame and the test data frame I had already declared before the function call, so that is what I'm passing right now).

Part 2.

Functions used;

def character_modify(word,attack='None'):

1. Now this function to be honest was fun to play around with fun to craft as well. In this function I applied all the steps that have been stated in the requirement.
2. Below are the things I'm doing in the function:
    a. The first thing I did was to check if the word is less than two words like a,i, etc, I'm leaving them as it's as I think removing this connecting words will make the text look blatant that it has been attacked.
    b. I then define the functions which takes care of all the tasks. Like add_whitespace(word), swap_characters(w), substitute_char(w), delete_char(w), add_char(w).
        i. As their name suggest, the function works as follows respectively: adding whitespace, swapping characters, substitute characters, delete characters, adding characters.

    c. I then created an array of modification which stores the name of all the functions.
    d. After this I call the random which only picks modification with the 50% probability and if the probability comes more than 50% by the random function it returns the word itself.
        i. In the case probability is less than 50%, then I use the random choices function again to pick any one of the random modification techniques to apply the modification.
        ii. Post that I return the word.
3. When I created the function, I added an attack argument to make the code reusable across all parts. In Part 2, the function can be called standalone, where it includes a 50% chance of modifying the word. However, in the untargeted attack (Part 3), where I'm already selecting 40% of the words to modify inside the function, I don't need that additional 50% check so the attack parameter disables it. The same logic applies to the targeted attack as well.

   Now what the results were and what they meant, I'll share later.

Part 3.

Functions used:

def untargeted_attack(text):

1. This function has a small job of splitting the text or review it gets as an input as the example from a test data from the run_targeted_attack_on_test function.
2. The function than uses the random function and to use only in the time of the 40% of the cases and then calls the character_modify function for each one of the word and stores the word and appends it.
3. It then returns the modified text.

def run_untargeted_attack_on_test(test_df, output_file):

1. This function like the name suggests, does the untargeted attack on the test data and outputs on the file with the file name specified in the output_file argument. I wanted to save all the 5 files which have been generated from the function.
2. In this function I call the def untargeted_attack(text) for each one of the example in the test data and then save that onto the new dataframe and then in turn save it onlt the file.
3. The function then returns the attacked_dataframe which I use to evaluate the result in the evaluate_on_attacked_data(model, tfidf, attacked_df), which I'll describe below.

def evaluate_on_attacked_data(model, tfidf, attacked_df):

1. In this function I take in the model, tf-idf embeddings and attacked_df. Now these arguments are the model which we trained in the part 1 and then the tf-idf embeddings which we created from the part 1 as well. Now the attacked_df is one which we obtained from the run_untargeted_attack_on_test function.

2. This function evaluates the accuracy, precision, recall and f1 score on the attacked_df for the untargeted data and also returns all these values as well.

Now to incorporate all these together I created a for loop which runs 5 time and calls the run_untargeted_attack_on_test which helps me save the file for each one of the iteration. I then call the evaluate_on_attacked_data in which I pass this attacked dataframe as the argument to calculate its accuracy, precision and f1 score.
Post this I store them and then print them for each one of the iterations as well.
After this I then print the result of average of all the result.
About the results and what they resemble we'll discuss later.

Part 4.

Now this part was trickiest to implement, as I had to do a lot of coding in an efficient way, in such a way that that the code remains clean and uses the exiting functionality as much as it can, while achieving what I the task needed to be done. I'll try my best to explain how I came to use them and how I did that.
Functions used:

def targeted_attack(text, model, vectorizer, orig_label):

1. In this function the arguments denote text as in the example of the test set, the model which we created in the part 1 i.e. the LR model and then vectorizer, which is our tf-idf embeddings and then the original label of the result.
2. The first step is doing the greedy selection. In this I first store the original vector by transforming the text using the vectorizer argument.
3. I then get the original probability of the correct label using the predict probability function of the model using the original vector and original labels. Now this is something I wrote cleanly in the single line. It picks the probability of the class based on the argument I pass in the original label.
4. I then run the loop through each one of the words in the text example and then calculate the highest probability drop from the text, by removing each word and then storing the probability in the array.
5. I then sort the array with the highest drop in the probability.
6. After then I run the loop again for each one of the word drop by using character_modify function and seeing if the label changes. If it doesn't then I move onto the next word. If it does, I then return the modified text.
7. In the case where the attack isn't successful with our method then I return the original text.
8. This function is called from the

def run_targeted_attack_on_test(test_df, model, vectorizer, output_file):

1. This function uses the heavy lifting done in the previous function. Here the arguments denotes, the test_df which is the data for the test set. The model which we trained in the part 1, vectorizer which is tf-idf we created in the part 1 and the output file as well.

2. Now I think I implemented the below part cleverly as well. So rather than calling the above targeted_attack every time. I only call this function when the labels are predicted correctly as I saw the model itself is predict the test data not very greatly.
    a. So, on the label when it predicts correctly I call the targeted_attack function to attack on that particular text.
    b. I then store the text on the attacked_text array.
3. Post this I then copy the test_df to a new attacked dataframe and I replace the text field with the array of texts I created.

Post this I save the file. This function is called from the run_targeted_5x function which calls this function 5 times. So that it can save file for all the 5 instances and help me calculate the average of all the metrics and produce the result. How I implemented this can be seen in the function below.

def run_targeted_attack_5x(test_df, model, vectorizer):

1. In this function I just pass the test dataframe, model which we trained in the part 1 and also the embeddings gathered from the part 1.
2. Now I run a loop for 5 times, which includes calling run_targeted_attack_on_test which in turns call the targeted_attack and works in cohesion with each other to get me the attacked dataframe, i.e. the attacked texts with the true label.
3. I then use the model to predict on the attacked text and check the result with the true label in the attacked data.
4. I then print the precision, recall, f1 score and accuracy for the texts. I also store this in the array so that I can take average of each one of them.
5. I the print the average accuracy, precision, recall, and f1 score based on the average I took from the arrays.

All of the functions for the targeted attacks, were a bit tough to think through and took a lot of time for me to do it. Making them work in cohesion and keeping the code clean was what I wanted and once it all came alive. It felt amazing.

Result and Analysis for each part:

Part 1.

Part 1 of the homework was pretty straightforward. Since I'm already familiar with Logistic Regression and TF-IDF vectorization, setting this up didn't pose any significant challenges. After training the model and evaluating it, I obtained the following results:

Dataset	Accuracy	Precision	Recall	F1 Score	Confusion Matrix
Train Data	0.97	0.97	0.97	0.97	TN: 4159, FP: 106, FN: 139, TP: 4126
Test Data	0.75	0.73	0.78	0.76	TN: 382, FP: 151, FN: 116, TP: 417

Looking at these results, the Logistic Regression model performed exceptionally well on the training data with near-perfect accuracy, precision, recall, and F1 scores. This indicates the model was effective in learning patterns from the training set.

However, the test data performance was noticeably lower, with accuracy dropping significantly to 0.75. Precision, recall, and F1 scores also decreased, which initially troubled me. The lower metrics for the test data caused some confusion and prompted me to double-check all the implementation details, but after thorough verification, it appears these are indeed the correct results. This significant drop from training to testing might indicate potential overfitting or simply differences in the distribution between the training and test datasets.

Part 2.

For Part 2, I chose all the main character-level modifications to apply randomly:

1. Whitespace Insertion: Randomly inserts whitespace within the word to split it into smaller segments.
2. Character Substitution: Randomly replaces specific characters with visually similar symbols (e.g., 'a' with '@').
3. Character Swapping: Randomly swaps the positions of two adjacent characters.
4. Delete character: Which randomly deletes the character.
5. Add Character: Which randomly adds a character by duplicating a character.

These modifications were implemented in a way that each chosen word had an equal probability of experiencing one type of modification. The random values used were set such that each word had approximately a 50% chance of being modified or left unchanged, ensuring diversity without overly altering the text.

Example outputs from the function are shown below:

Original Word	Modified Outputs	Modification Observed
great	grea t, g reat, gre@t	Whitespace insertion and substitution ('a' → '@')
movie	movie, movie, movei	No modification and character swapping ('i' ↔ 'e')

In the example above, we can see that the word "movie" was unchanged in two out of three cases, demonstrating that the function correctly implements the intended randomness, with a modification rate below 50%. For the word "great," whitespace insertion occurred twice ("grea t", "g reat") and character substitution once ("gre@t"). For the word "movie," character swapping occurred once ("movei").

Overall, these results confirm that the character-level modification functions designed for Part 2 are operating as expected, providing controlled and varied randomization.

Part 3.

In Part 3, I implemented an untargeted attack function (untargeted_attack). This function randomly selects words from the input text, with each word having a 40% chance of undergoing character-level modifications. Once selected, the word is modified using the previously defined function (character_modify), which randomly applies one of the modifications: whitespace insertion, character substitution, character swapping, deletion, or addition.

Below are the comparative results for the original test data and after performing the untargeted attacks:

Dataset	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Original Test Data	0.75	0.73	0.78	0.76
Untargeted Attack Run 1	0.71	0.70	0.73	0.72
Untargeted Attack Run 2	0.72	0.72	0.73	0.72
Untargeted Attack Run 3	0.72	0.71	0.74	0.72
Untargeted Attack Run 4	0.72	0.72	0.73	0.73
Untargeted Attack Run 5	0.72	0.72	0.72	0.72
Average After 5 Attacks	**0.72**	**0.71**	**0.73**	**0.72**

Observations from the above table indicate a moderate but consistent reduction in performance across all metrics compared to the original, unmodified texts. This indicates that random character-level modifications did impact the classifier slightly, but the overall robustness of the Logistic Regression model remained strong.

Below are five randomly chosen examples comparing the original texts and their modified counterparts after the untargeted attack:

Original Text	**Untargeted Attack Example**
a real audience-pleaser that will strike a chord with anyone who's ever waited in a doctor's office , emergency room , hospital bed or insurance company office .	a real audience-pleaser that wil strike a chord wi th anyone wh's ever w@ited i a doctor's office , emregency room , hospital be d or insurance company office .
exposing the ways we fool ourselves is one hour photo's real strength .	exposing the w@ys we fol ourselves s one hour ph oto's real strength .
. . . quite good at providing some good old fashioned spooks .	. . . quite god at providing some ood oold fashioned spooks .
you've already seen city by the sea under a variety of titles , but it's worth yet another visit .	yoou've already seen city by the sea under a variety of titles , but its worth yt another v isit .

this kind of hands-on storytelling is ultimately what makes shanghai ghetto move beyond a good , dry , reliable textbook and what allows it to rank with its worthy predecessors .	this kind 0f hands-on storytelli ng i s ultimately what make s shanghia ghettto move beynod a g0od , dry , reliable textbook @nd what a llows it to rank with it s worthy predeecssors .

Detailed Observations of Changes:

Sentence 1: "will" → "wil" (Deletion), "with" → "wi th" (Whitespace insertion), "who's" → "wh's" (Deletion), "waited" → "w@ited" (Character substitution), "emergency" → "emregency" (Character swap), "bed" → "be d" (Whitespace insertion)

Sentence 2: "ways" → "w@ys" (Character substitution), "fool" → "fol" (Deletion), "photo's" → "ph oto's" (Whitespace insertion)

Sentence 3: "good" → "god" (Deletion), "good" → "ood" (Deletion), "old" → "oold" (Character addition)

Sentence 4: "you've" → "yoou've" (Character addition), "yet" → "yt" (Deletion), "visit" → "v isit" (Whitespace insertion)

Sentence 5: "of" → "0f" (Character substitution), "storytelling" → "storytelli ng" (Whitespace insertion), "is" → "i s" (Whitespace insertion), "makes" → "make s" (Whitespace insertion), "shanghai" → "shanghia" (Character swap), "ghetto" → "ghettto" (Character addition), "beyond" → "beynod" (Character swap), "good" → "g0od" (Character substitution), "and" → "@nd" (Character substitution), "allows" → "a llows" (Whitespace insertion), "its" → "it s" (Whitespace insertion), "predecessors" → "predeecssors" (Character swap)

Readability Observations:

The readability after untargeted modifications notably declined but remained mostly comprehensible. The random modifications sometimes disrupted the coherence of sentences but did not consistently target key words crucial to the text's overall meaning.

Word Selection Observations:

Because the modifications were random, many significant words likely relied upon by the classifier (such as "audience-pleaser", "emergency", "strength", "good", "storytelling") remained partially intact or were only slightly modified, resulting in modest performance degradation. Thus, while random modifications reduced accuracy slightly, they didn't precisely disrupt the classifier's reliance on critical semantic features.

In conclusion, the untargeted attack moderately reduced classifier performance, reflecting the classifier's resilience against minor, random perturbations.

Part 4.

For Part 4, I implemented a targeted attack function (targeted_attack) utilizing classifier feedback to precisely select words to modify. The targeted attack consists of two main components:

1. **Greedy Word Selection**: This involves iteratively removing each word from the text to measure the reduction in the classifier's confidence in predicting the correct class. The words that caused the greatest drop in prediction confidence were prioritized for modification.
2. **Word Modification**: The selected words were individually modified using the character-level modification function (character_modify) until the classifier's prediction changed.

Below are the comparative results for the original test data, untargeted attacks, and targeted attacks:

Dataset	Accuracy	Precision	Recall	F1 Score
Original Test Data	0.75	0.73	0.78	0.76
Avg Untargeted Attack	0.72	0.71	0.73	0.72
Targeted Attack Run 1	0.21	0.20	0.20	0.20
Targeted Attack Run 2	0.21	0.19	0.18	0.19
Targeted Attack Run 3	0.21	0.20	0.19	0.20
Targeted Attack Run 4	0.20	0.18	0.17	0.18
Targeted Attack Run 5	0.22	0.21	0.20	0.21
Average Targeted Attack	**0.21**	**0.20**	**0.19**	**0.19**

As evident from the results, the targeted attack substantially reduced all metrics compared to both the original and untargeted texts. This considerable decrease in performance demonstrates that systematically targeting words critical to the classifier's decisions effectively disrupted its predictions more than random modifications.

Below is a comparison of original texts, untargeted attacked texts, and targeted attacked texts:

Original Text	Untargeted Attack Example	Targeted Attack Example
a real audience-pleaser that will strike a chord with anyone who's ever waited in	a real audience-pleaser that wil strike a chord wi th anyone wh's ever w@ited i a	a real audeince-pleaser that will strike a chord with anyone who's ever waited in

a doctor's office , emergency room , hospital bed or insurance company office .	doctor's office , emregency room , hospital be d or insurance company office .	a doctor's office , emergency room , hospital bed or insurance c ompany office .
exposing the ways we fool ourselves is one hour photo's real strength .	exposing the w@ys we fol ourselves s one hour ph oto's real strength .	exposing the ways we fool ourselves is one hour pho+o's real stre ngth .
. . . quite good at providing some good old fashioned spooks .	. . . quite god at providing some ood oold fashioned spooks .	. . . quite god at providing some ood oold fashioned spooks .
you've already seen city by the sea under a variety of titles , but it's worth yet another visit .	yoou've already seen city by the sea under a variety of titles , but its worth yt another v isit .	you've already seen city b y teh sea underr a varieety o title s , bt iit's worth ye a nother vvisit .
this kind of hands-on storytelling is ultimately what makes shanghai ghetto move beyond a good , dry , reliable textbook and what allows it to rank with its worthy predecessors .	this kind 0f hands-on storytelli ng i s ultimately what make s shanghia ghettto move beynod a g0od , dry , reliable textbook @nd what a llows it to rank with it s worthy predeecssors .	this kind of hands-on storytelling is ultimately what makes shanghai ghetto move beyond a goodd , dry , reliable textbook and what allosw it to rank with its worthy predecessors .

Detailed Observations of Changes (Targeted Attack):

- **Sentence 1:** "audience" → "audeince" (Character swap), "company" → "c ompany" (Whitespace insertion)
- **Sentence 2:** "photo's" → "pho+o's" (Character substitution), "strength" → "stre ngth" (Whitespace insertion)
- **Sentence 3:** "good" → "god" (Character deletion), "good old" → "ood oold" (Character deletion and addition)
- **Sentence 4:** "by" → "b y" (Whitespace insertion), "the" → "teh" (Character swap), "under" → "underr" (Character addition), "variety" → "varieety" (Character addition), "of" → "o" (Character deletion), "titles" → "title s" (Whitespace insertion), "but" → "bt" (Character deletion), "it's" → "iit's" (Character addition), "yet" → "ye" (Character deletion), "another" → "a nother" (Whitespace insertion), "visit" → "vvisit" (Character addition)
- **Sentence 5:** "good" → "goodd" (Character addition), "allows" → "allosw" (Character swap)

Readability Observations:

- Readability drastically deteriorated due to targeted modifications affecting critical words, making sentences notably more challenging to understand compared to the untargeted attack.

Word Selection Observations:

- The targeted attack precisely identified and altered words likely critical for classifier prediction (such as "audience", "photo's", "good", and "visit"), thereby more effectively compromising classifier performance compared to the random untargeted approach.

Overall, the targeted attack significantly outperformed the untargeted attack in reducing classifier accuracy by methodically altering key influential words.

Conclusion:

In this task, I initially trained a Logistic Regression model using TF-IDF embeddings, achieving solid performance in terms of accuracy, precision, recall, and F1 score. After introducing various character-level modifications such as inserting whitespace, substituting characters, swapping letters, adding or deleting characters I noticed moderate disruptions in readability but minimal impact on model predictions.

When applying these modifications randomly in an untargeted attack, the classifier's accuracy decreased slightly but not significantly. The main drawback was the randomness, as many altered words were not crucial to the classifier's decision-making, limiting the effectiveness of the attack. However, readability remained relatively intact, making the untargeted attack less impactful but more practical.

However, in the targeted approach, specifically selecting words based on their importance to the classifier's predictions. By identifying and modifying the most influential words, this targeted attack considerably reduced the classifier's accuracy and overall performance metrics. It effectively disrupted the model by altering key words, making it significantly more effective than the random method.

Nevertheless, the targeted attack greatly reduced readability, making texts notably harder to understand compared to the untargeted modifications. Thus, while the targeted attack was highly effective in misleading the classifier, it compromised the practical readability of the texts.

Overall, the untargeted attack maintained better readability but lacked precision and effectiveness, whereas the targeted attack succeeded in strategically undermining classifier performance, though at the cost of significantly reduced readability.

Problems encountered:

- Initially, understanding and correctly implementing the greedy selection approach took some extra time and effort.
- I had to significantly restructure the code by creating reusable functions to avoid repetition.
- Integrating these functions within other functions increased the overall complexity of the implementation.
- Despite these challenges, the entire process provided an excellent and valuable learning experience.