Theoretical Tasks

Task 1.1 Ethics

Theoretical Background

Bias in training data is a pertinent issue that demands the attention of machine learning practitioners and data scientists. It pertains to the presence of skewed, unrepresentative, or unfair elements within the data used to train machine learning models. When present, bias can lead to inaccurate or unfair predictions and decisions made by the model. To gain a deeper understanding of bias in training data, consider the following points:

Skewed Representation: Overrepresentation or underrepresentation of certain groups or types of data in the training dataset can result in biased outcomes. For example, if a facial recognition system is trained mostly on images of lighter-skinned individuals, it may not perform well in accurately identifying people with darker skin tones.

Unfair Treatment: Bias in training data can lead to unfair treatment of certain groups. This is particularly concerning in areas such as credit scoring, hiring processes, and law enforcement.

Impact on Model Performance: Biased training data can significantly impact the performance of machine learning models, leading to inaccurate predictions and decisions.

Ethical Considerations: Addressing bias in training data is an integral part of ethical AI development, as it aims to ensure that AI systems make fair and unbiased decisions for all individuals and groups.

You can ask any LLM of your choice and they will tell you why they are biased! It all comes down to the training data fed. Here we are trying to show you how they display their bias.

First take any LLM you want and feed them some professions prompts: like the doctor, nurse, the lawyer, the office worker, the janitor, the construction worker, etc. translate them into non-neutral (like english) into another with genders (Swedish, Spanish, etc.) and try to get it to use stereotypes.

Present your findings with examples.

Metrics:

Theoretical Background

As a fundamental tool for evaluating classification models, the confusion matrix, also known as an error matrix, provides a square table that displays the number of correct and incorrect predictions made by a model for each target class. The matrix is structured with rows representing the actual class and columns representing the predicted class.

The confusion matrix represents four key metrics: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). These metrics are useful in measuring the model's performance and can be used to calculate performance measures like accuracy, precision, recall, and F1-score.

The confusion matrix offers several advantages in evaluating classification models. Firstly, it provides a clear and interpretable visualization of the model's performance, allowing for quick identification of strengths and weaknesses in classifying specific classes. Additionally, it facilitates error analysis, enabling researchers to pinpoint areas for improvement and refine the model's training process. The matrix is also particularly beneficial when dealing with imbalanced class distributions, where one class might have significantly more instances than others.

In conclusion, the confusion matrix is a valuable tool for evaluating classification models because it provides a clear and detailed representation of the model's performance. Its metrics enable researchers to identify areas for improvement and refine the model's training process, making it an essential element of the machine learning toolkit.

Confusion Matrices Examples:

		Predicted	
		Negative	Positive
Actual	Negative	TN	FP
	Positive	FN	TP

		Predicted	
		Negative	Positive
Actual	Negative	990	10
	Positive	20	30

Task:
Fill out all the missing values, and put an explanation of why accuracy may not be the best metric.

- True Negatives (TN):

- False Positives (FP):
- False Negatives (FN):
- True Positives (TP):

Calculate precision, recall, and F1 score:

$$\text{Precision} =$$

$$\text{Recall} =$$

$$\text{Accuracy} =$$

$$\text{F1 Score} =$$

Now with this:

		Predicted	
		No	Yes
Actual	No	9000	50
	Yes	100	850

$$\text{Precision} =$$

$$\text{Recall} =$$

$$\text{Accuracy} =$$

$$\text{F1 Score} =$$

Practical 1: Chatbot – Simple ANN & Tranformers

Task 1.1

Create a simple chatbot that analyzes text responses typed by a user using an artificial neural network (ANN).

The minimum requirement is that the bot prompts a response from the user (with various possible prompts). When the user has typed the answer, the bot should analyze the text (using an ANN) and formulate a response based on the analysis.

A small dataset of product reviews that have been labelled as negative (0) or positive (1) is provided in the Files - Exercises - Lab 1 folder, along with some code needed to extract information.

A suggested approach is first to try to train a network on the given data.

When that task has been concluded, the model can be improved by finding more data, using a dataset with a broader range of labels, using word embeddings to create unique sentence embeddings, making the bot capable of extended dialogue or any other extension you want to pursue.

1 Task 1.2 Transformers Implementation

For this task, you will implement your transformer in PyTorch. You are instructed to follow this link: Tranformers in Pytorch.

2 Task 1.2 (Alternative)

If you find any problems with the previous code, links, or set-up (due to Amazon or anything else), we offer you another alternative to developing your own transformer.

You can follow Andrej Karpathy tutorial for a NanoGPT here: YouTube and use his GitHub with the code here: NanoGPT or you can develop your own code if you want.

The only requirement is to standardise your training data. Make it the same across your implementations.

3 Task 1.3

Comparison

Here, it would be best if you did a comparison of both models; you are requested to modify your Chatbot to use the same data as the transformer and answer the following:

- Compare the performance of the two models and explain in which scenarios you would prefer one over the other.
- How did the two models' complexity, accuracy, and efficiency differ? Did one model outperform the other in specific scenarios or tasks? If so, why?
- What insights did you obtain concerning data amount to train? Embeddiutilizedised? Architectural choices made?