SSRI - Finetuning LLMs in Python for Survey Response Classification

Nick Hemauer
Ph.D. Candidate - Political Science
& Social Data Analytics

Overview

Today, we will discuss the “full-stack” process of LLM text classification. The goal is to provide a general understanding of the models being used and to inspire potential research applications.

GitHub Repository: https://github.com/nhemauer/SSRI_LLM_Finetune

The slides will follow the following topic order:

1. Definitions and Data
2. LLMs for Research
3. Neural Networks - The Big Picture
4. Transformer/LLM Architectures
5. BERT Architecture
6. Model Evaluation
7. Application to Binary Sentiment Analysis

Definitions and Data

There are two terms that are often applied to neural networks:

1. **Pretrained**
    a. The models parameters are trained on an (often) general corpus of knowledge.
2. **Finetuned**
    a. The pretrained model is taken and provided with additional training to perform better on a specific task. Finetuning allows for “personalization.”

Almost any application in social science will be finetuning.

Data

- Data is incredibly important, both the quality and quantity.
- Complex tasks with lower signal or tasks with more classes will require exponentially more data.
- For binary classification <500 samples may be sufficient.
- For 5+ classes the sample required may be several thousand depending on how distinct the classes are.

LLMs for Research

Learning to finetune personal models has advantages over other approaches.

OpenAI (ChatGPT), Anthropic (Claude), Google (Gemini), all offer paid APIs that can be applied to data for classification.

- These are good options IF the data you want to classify is openly available.

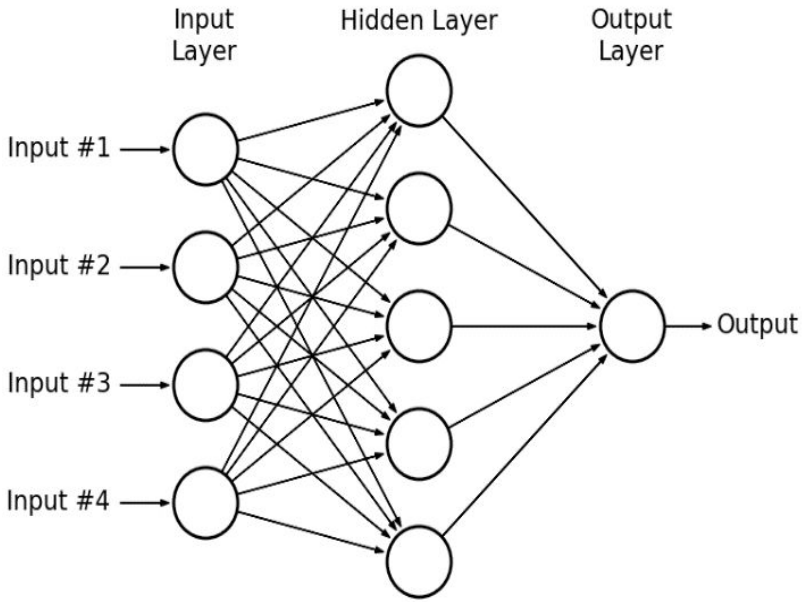
However, in the case of IRB approved surveys, this approach breaches respondent confidentiality, as the data is being given directly to these companies.

- This is the benefit of finetuning a small personal model.

Neural Networks - The Big Picture

Machine learning is the process of giving data to a model to predict a behavior.

Neural networks exist within the overarching idea of machine learning and are designed to artificially mimic the human brain.



This example visualizes how we can input data to receive an output prediction:

1. The input layer nodes could be thought of as variables in a dataframe.

2. As the data goes through the model, each of the nodes in the hidden layer connect and learn from the data given.

3. The output layer takes this signal and produces a prediction.

Transformer/LLM Architectures

The architecture of the modern LLM (post-2017) is defined by an **encoder** and/or a **decoder** block. These blocks may be used separately or together.

Decoder only models are those such as ChatGPT, Claude, or Gemini. These models excel at generating text autoregressively.

Encoder models, in contrast, excel at classifying text. Unlike decoder models, encoder models look both before and after each token in the input sequence. Examples of these are BERT, RoBERTA, DeBERTa, etc.

When we combine both blocks into one model, we get models such as BART, which excel at summarizing, translating, or rewriting text.

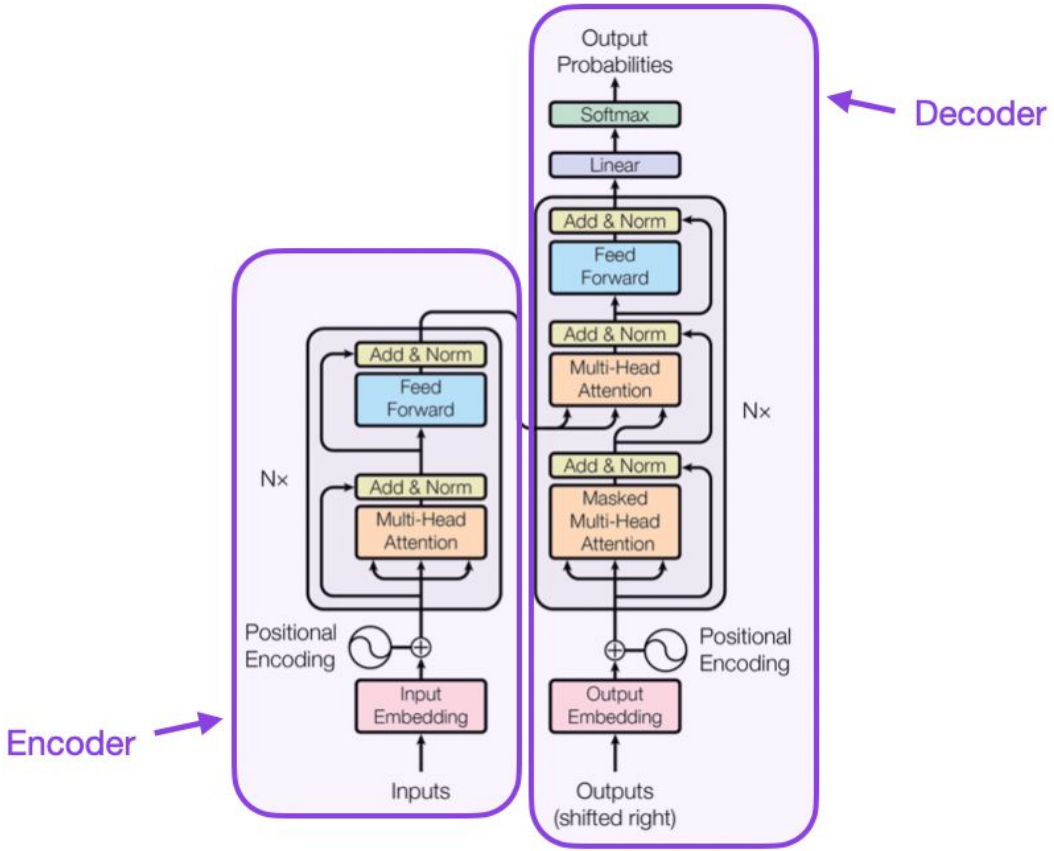


Figure 1: The Transformer - model architecture.

https://www.capellasolutions.com/blog/understanding-llm-agent-architectures

BERT Models - Forward Pass

For the purposes of this presentation, we will focus on encoder models, and specifically: **Bidirectional Encoder Representations from Transformers (BERT)**.

In the case of classification using BERT, the "forward pass" (sending the data through for an output) works as follows:

1. **Tokenizing**
    a. Input text is broken into subwords and mapped to unique token IDs.
2. **Embedding**
    a. Each token is embedded to a vector of numbers of n-dimensions (BERT uses 768 dimensions).
    b. The model accounts for the unique token and the position of the token.
3. **Encoding**
    a. The model uses self-attention to learn context across and between all words.
4. **Classification Head**
    a. Extract final model state and produce logits for each class.
5. **Activation Functions**
    a. The model applies sigmoid or softmax functions to return class probabilities.

BERT Models - Backpropagation (Training)

To train, the model is “backpropagated”:

1. **Forward Pass**
    a. Calculate the final output using training data.
2. **Compute the Loss (or “Error”)**
    a. Compute the loss between the observed target and prediction.
3. **Backpropagation and Gradient Descent**
    a. Compute the derivative of the output loss and backpropagate, applying gradient descent.
    b. Here, “backpropagate” means moving backwards through the network.
4. **Update the Neuron Weights**
    a. Update the neurons (or parameters) to improve prediction.

These 4 steps are equivalent to 1 epoch. Models are then trained for several epochs, and hopefully, the model continues to improve.

Model Evaluation

The evaluation metric will differ depending on the data and task.

<u>Balanced Data (Samples are Not Dominated by One Class)</u>

1. **Accuracy**
    a. Divide correct predictions over all results.
    b. Not very informative.
2. **Receiver Operating Characteristic - Area Under the Curve (ROC-AUC)**
    a. For binary classification, but there are multiclass extensions.
    b. Plot false positive rate against the true positive rate.
    c. 0-1 Score
3. **F1 Score**
    a. A single metric (0-1) metric that balances precision and recall.
    b. **Precision = True Positives / (True Positives + False Positives)**
        i. "Of all positive predictions how many were correct?"
    c. **Recall = True Positives / (True Positives + False Negatives)**
        i. "Of all actual positives, how many were caught?"
    d. There are multiclass extensions (F1-Micro and F1-Macro).

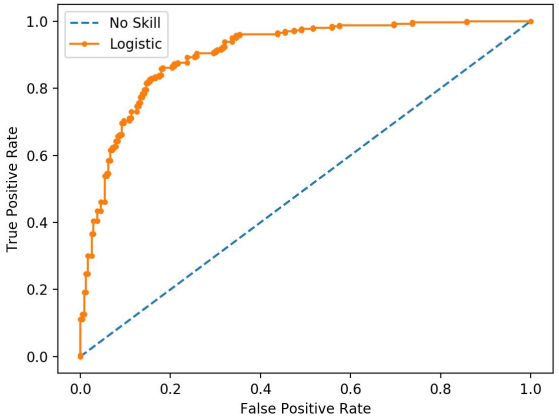


Figure 2. ROC-AUC

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Model Evaluation Cont.

Imbalanced Data (e.g., 95% of Samples in One Class)

1. **F1 Score**
    a. F1 can work well for both imbalanced and balanced data.
2. **Balanced Accuracy**
    a. Weights each class equally.
3. **AUC-PR**
    a. Similar to ROC-AUC, but focuses on both positive and negative classes.
    b. 0-1 Score
    c. X-Axis is Recall and Y-Axis is Precision.
        i. Precision = True Positives / (True Positives + False Positives)
        ii. Recall = True Positives / (True Positives + False Negatives)

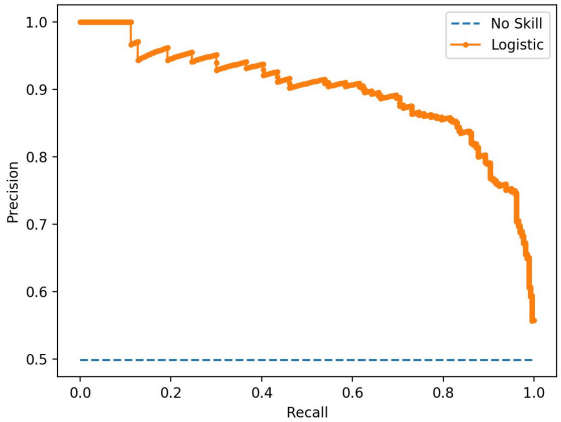


Figure 3. AUC-PR

An Application: Binary Sentiment Analysis

With a general understanding of language models, we can now go to a Python coded application.

I will be presenting a model I trained on a dataset of App Store reviews for Spotify.

- https://www.kaggle.com/datasets/alexandrakim2201/spotify-dataset

GitHub Repository: https://github.com/nhemauer/SSRI_LLM_Finetune