Problem Framing as a Machine Learning Task

When thinking about how to solve the problem of intent detection, I needed to translate it into a machine learning problem. Here's how I approached it in a practical and relatable way:

Step 1: Define What We're Solving

Imagine someone asking, "How do I return this item?" The goal here is to identify the *intention* behind the question. In this case, the intent is RETURN_EXCHANGE. So, I framed this as a classification task:

- Each query (text) needs to be assigned a single category (intent).
- There are multiple possible intents (e.g., LEAD_GEN, ORDER_STATUS, etc.).

In machine learning terms:

- Input: A user's query (e.g., "Can I return this item?").
- Output: A label representing the query's intent (e.g., RETURN_EXCHANGE).

Step 2: Convert Text into Something a Model Understands

Machines don't understand plain text, so I needed a way to translate words into numbers. This is where BERT comes in. It creates "embeddings," which are essentially numerical representations of words but with an understanding of context. For example:

- The word "return" means something different in "return a product" versus "return to sender."
- BERT helps the model understand this difference.

Using BERT:

1. I tokenized each query, breaking it into smaller pieces and converting it into numbers.
2. I created attention masks to focus on relevant words and ignore padding or extra spaces.

Step 3: Choose the Right Type of Learning

This is a supervised learning problem:

- I already have labeled data (queries paired with their intents).
- The model learns from these examples to predict intents for new queries.

Pros and Cons of Different Approaches

Before finalizing my solution, I considered several approaches. Here's a breakdown of what I explored, what worked, and what didn't:

Approach 1: Traditional Machine Learning Models

- What I Tried:
    - Models like Logistic Regression, Support Vector Machines (SVM), and Random Forests.
    - I converted the text into numerical features using techniques like TF-IDF (Term Frequency-Inverse Document Frequency).
- Pros:
    - Simple to implement and quick to train.
    - Works well for small datasets.
- Cons:
    - These models don't understand context. For example, they might treat "return an item" and "return to sender" as the same thing.
    - Scaling to a large dataset or nuanced intents becomes challenging.

Why I Didn’t Use This: Traditional models struggle with understanding the nuances of language and context, which are critical for intent detection.

Approach 2: Using Pre-trained Word Embeddings (e.g., Word2Vec, GloVe)

- What I Tried:
    - Pre-trained word embeddings to convert words into vectors and then passed these vectors to models like Neural Networks or SVM.
- Pros:
    - Captures some semantic relationships between words (e.g., "king" and "queen" are related).
    - Faster than training embeddings from scratch.
- Cons:
    - Embeddings like Word2Vec don’t consider word order or context effectively.
    - They’re static; the meaning of a word doesn’t change based on its surroundings.

Why I Didn’t Use This: While better than traditional approaches, these embeddings still lacked the contextual understanding I needed.

Approach 3: Using BERT (Final Choice)

- What I Chose:
    - Fine-tuning BERT-base-uncased for intent classification.
- Why I Chose BERT:
    - Contextual Understanding:
        - BERT considers both the word and its context in the sentence. For example, it knows that "return" in "return an item" refers to refunds, while in "return to sender," it means going back.
    - Pre-trained Power:
        - BERT is already trained on a vast amount of text, so it understands grammar, semantics, and common usage patterns.

    - Flexibility:
        - I could fine-tune BERT specifically for my dataset, adapting it to the unique intents in my application.
    - Performance:
        - BERT achieves state-of-the-art results in many NLP tasks, including classification.
- Challenges with BERT:
    - Computationally Expensive: Training and fine-tuning BERT requires a good GPU.
    - Data Requirements: It performs best with a decent-sized dataset for fine-tuning.

Why My Approach Works

1. Understanding Nuance:
    - The difference between "check delivery status" and "delay in delivery" lies in subtle context. BERT captures these nuances better than simpler models.
2. Scalable:
    - Once trained, the model can handle queries from new users or expand to new intents with additional fine-tuning.
3. Proven Effectiveness:
    - Using a pre-trained language model like BERT ensures the system starts with a strong understanding of language, reducing the burden on the dataset.

Summary

In the end, I framed this as a supervised multi-class classification problem. After weighing the pros and cons of various methods:

- I chose BERT for its contextual understanding and flexibility.

- While it required more computational resources, the significant improvement in accuracy and reliability made it the best choice for intent detection.