**Technical Assignment Report**

**Author:** Amin Khodamoradi  
**Date:** 14/06/2025

**Objective**

The objective of this assignment was to build a text classification pipeline that:

* Classifies texts into one of the following categories: cult, paranormal, or dramatic (using task.csv).
* Offers a demo web interface for prediction using Gradio.
* Provides a REST API for programmatic access using FastAPI.
* Is containerized via Docker for consistent deployment.
* Optionally, extends to a multi-label classification using bonus\_task.csv.

**Step 1: Data Investigation**

The dataset task.csv contained movie metadata with the following columns:

* Title: Movie title
* Synopsis: Movie description (used for classification)
* Tag: Ground-truth category

Basic statistics and value counts were extracted to understand class distribution.

**Step 2: Preprocessing**

* Converted all text to lowercase.
* Removed special characters and punctuation using regex.
* Created a clean\_synopsis column for the cleaned text.
* Encoded the Tag labels using LabelEncoder into numeric format.

**Step 3: Train-Test Split**

Used an 80/20 split on the cleaned synopsis and encoded labels:

* X\_train, X\_test: Vectorized text features
* y\_train, y\_test: Encoded class labels

Stratification was used to preserve label distribution.

**Step 4: Feature Engineering (TF-IDF)**

Used TfidfVectorizer to convert cleaned text into numerical feature vectors:

* Max features: 5000
* Fitted on training data, transformed both train and test sets

**Step 5: Model Training and Evaluation**

Trained a logistic regression model (LogisticRegression) as a baseline classifier.  
Other models such as RandomForestClassifier and GradientBoostingClassifier were explored.

Evaluation metrics:

* Precision, Recall, F1-score (using classification\_report)
* Label decoding used to interpret predictions

**Step 6: Gradio Demo App**

Developed a user-friendly demo with Gradio:

* Allows users to input/paste text
* Displays predicted category in real-time
* Bound to 0.0.0.0:7860 for Docker networking

**Step 7: FastAPI REST API**

Built a REST endpoint /predict using FastAPI:

* Accepts JSON payload: { "text": "your input here" }
* Returns: { "prediction": "cult" }

Tested with Swagger UI (/docs), Postman, and PowerShell curl.

**Step 8: Dockerization**

Created a Dockerfile to package the app:

* Python 3.11.3 base image
* Installs dependencies via requirements.txt
* Exposes ports 8000 (API) and 7860 (Gradio)
* Entrypoint defaults to FastAPI, but Gradio can be run with CMD override

**Step 9: Docker Compose (Run Both Services Together)**

Used docker-compose.yml to run both FastAPI and Gradio apps concurrently:

* fastapi\_service at port 8000
* gradio\_service at port 7860
* Shared image and codebase

**Bonus (Not Implemented Yet)**

The bonus task involves using bonus\_task.csv for **multi-label classification**.  
The plan is to:

* Use transformers to embed the text
* Create a RAG to tag the input text

**Tools & Libraries Used**

* Python 3.11
* pandas, scikit-learn
* Gradio
* FastAPI
* Docker & Docker Compose

**Conclusion**

The project demonstrates a complete NLP deployment pipeline:

* Preprocessing & training
* Evaluation
* Real-time demo (Gradio)
* API integration (FastAPI)
* Scalable deployment (Docker Compose)

This solution is modular and extensible — ready to integrate bonus multi-label support or scale with larger models like BERT and Transformers.