Problem Statement: **Real-Time Toxic Comment Detection**

- Goal: Detect toxic or offensive comments in social media posts.
- Tools: sklearn, nltk, or DistilBERT with small batch size
- Tasks:
    - Use the Jigsaw Toxic Comment dataset (or a smaller sample)
    - Train logistic regression or use a small transformer
    - Build a simple web interface or browser extension to scan text and classify
- Bonus: Highlight toxic keywords using color in output.

Implementation:

[ ]

```
"""
Multi-Label Toxic Comment Detection - Logistic Regression + Gradio (from Google Drive)


This notebook demonstrates building a multi-label toxic comment classifier
predicting probabilities for 6 types of toxicity.
It uses Logistic Regression with OneVsRestClassifier, TF-IDF, loads data
from Google Drive, and deploys with a Gradio web interface.
Includes bonus keyword highlighting.
"""
```

[→]

```
'\nMulti-Label Toxic Comment Detection - Logistic Regression + Gradio (from Google Drive)\n\nThis notebook demonstrates building a multi-label toxic comment classifier\npredicting probabilities for 6 types of toxicity.\nIt uses Logistic Regression with OneVsRestClassifier, TF-IDF, loads data\nfrom Google Drive, and deploys with a Gradio web interface.\nIncludes bonus keyword highlighting.\n'
```

[ ] 1. Setup: Install Libraries and Import Modules

```
# @title 1. Setup: Install Libraries and Import Modules
!pip install numpy pandas scikit-learn nltk joblib gradio --quiet


import os
```

```python
import re

import string

import joblib

import pandas as pd

import numpy as np

import nltk

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.linear_model import LogisticRegression

from sklearn.multiclass import OneVsRestClassifier

from sklearn.metrics import roc_auc_score, classification_report, hamming_loss,
jaccard_score, accuracy_score as subset_accuracy

import gradio as gr

from google.colab import drive


# Download necessary NLTK data (if not already present)

try:

    nltk.data.find('corpora/wordnet.zip') # Check for the zip file, more robust

except LookupError: # Catch LookupError directly

    nltk.download('wordnet', quiet=True)

try:

    nltk.data.find('corpora/stopwords.zip')

except LookupError:

    nltk.download('stopwords', quiet=True)


# Import NLTK submodules after ensuring resources are available

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer


print("Libraries installed and imported.")

print("NLTK resources checked/downloaded.")
```

[→]

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.1/54.1 MB
13.5 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 322.9/322.9 kB
16.4 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 95.2/95.2 kB
6.9 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.5/11.5 MB
48.6 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.0/72.0 kB
4.0 MB/s eta 0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.5/62.5 kB
5.3 MB/s eta 0:00:00
Libraries installed and imported.
NLTK resources checked/downloaded.
```

[ ] 2. Mount Google Drive and Specify Dataset Path

```python
# @title 2. Mount Google Drive and Specify Dataset Path

# Mount Google Drive
drive.mount('/content/drive')
print("Google Drive mounted.")

# --- DATASET PATH ---
BASE_DRIVE_PATH = '/content/drive/MyDrive/Jigsaw_Toxic_Comment_dataset'
DRIVE_DATASET_PATH_TRAIN = os.path.join(BASE_DRIVE_PATH, 'train.csv')
DRIVE_DATASET_PATH_TEST = os.path.join(BASE_DRIVE_PATH, 'test.csv')
DRIVE_DATASET_PATH_TEST_LABELS = os.path.join(BASE_DRIVE_PATH, 'test_labels.csv')

# Define the toxicity labels we are interested in
TOXIC_LABELS = ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult',
'identity_hate']

if not os.path.exists(DRIVE_DATASET_PATH_TRAIN):
    print(f"ERROR: Training data file not found at {DRIVE_DATASET_PATH_TRAIN}")
```

```
    print("Please ensure the file 'train.csv' exists in the specified Google Drive folder.")
    print(f"Expected folder: {BASE_DRIVE_PATH}")
else:
    print(f"Training data path set to: {DRIVE_DATASET_PATH_TRAIN}")
    print(f"Test data path set to: {DRIVE_DATASET_PATH_TEST}")
    print(f"Test labels path set to: {DRIVE_DATASET_PATH_TEST_LABELS}")
    print(f"Target labels: {TOXIC_LABELS}")

# You can quickly check if the files exist:
print(f"\nChecking file existence:")
print(f"Train CSV exists: {os.path.exists(DRIVE_DATASET_PATH_TRAIN)}")
print(f"Test CSV exists: {os.path.exists(DRIVE_DATASET_PATH_TEST)}") # Will be False if test.csv is not there
print(f"Test Labels CSV exists: {os.path.exists(DRIVE_DATASET_PATH_TEST_LABELS)}") # Will be False if test_labels.csv is not there
```

[→]

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
Google Drive mounted.
Training data path set to: /content/drive/MyDrive/Jigsaw_Toxic_Comment_dataset/train.csv
Test data path set to: /content/drive/MyDrive/Jigsaw_Toxic_Comment_dataset/test.csv
Test labels path set to: /content/drive/MyDrive/Jigsaw_Toxic_Comment_dataset/test_labels.csv
Target labels: ['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']


Checking file existence:
Train CSV exists: True
Test CSV exists: True
Test Labels CSV exists: True
```

[ ] 3. Load and Sample Data from Google Drive

```
# @title 3. Load and Sample Data from Google Drive
# --- Configuration ---
```

```python
SAMPLE_SIZE = 30000 # Adjust as needed. Set to None to use full dataset (might be slow/memory intensive).
DATA_FILE_TRAIN = DRIVE_DATASET_PATH_TRAIN


df_processed = pd.DataFrame()


if not os.path.exists(DATA_FILE_TRAIN):
     print(f"ERROR: {DATA_FILE_TRAIN} not found. Please check the path in Cell 2.")
else:
    print(f"Loading training data from {DATA_FILE_TRAIN}...")
    try:
        df = pd.read_csv(DATA_FILE_TRAIN)
        print("Original training data shape:", df.shape)

        # Handle potential missing values in comments BEFORE sampling
        df['comment_text'].fillna("missing", inplace=True)

        if SAMPLE_SIZE and SAMPLE_SIZE < len(df):
            print(f"Sampling {SAMPLE_SIZE} records...")
            df_processed = df.sample(n=SAMPLE_SIZE, random_state=42).copy()
            print("Sampled data shape:", df_processed.shape)
        else:
            df_processed = df.copy()
            print("Using full dataset. Shape:", df_processed.shape)

        print("\nData Sample (first 5 rows of processed data):")
        print(df_processed.head())
        print("\nLabel distribution in processed data (sum of labels):")
        print(df_processed[TOXIC_LABELS].sum())

    except Exception as e:
        print(f"Error loading or processing data: {e}")
```

```
if df_processed.empty:

    print("\n---! DATAFRAME IS EMPTY !--- Halting execution. Check file path and content.")

    # exit() # Uncomment to forcibly stop if dataframe is empty
```

[→]

```
Loading training data from
/content/drive/MyDrive/Jigsaw_Toxic_Comment_dataset/train.csv...
Original training data shape: (159571, 8)
Sampling 30000 records...
Sampled data shape: (30000, 8)

Data Sample (first 5 rows of processed data):
                      id                                       comment_text  \
119105  7ca72b5b9c688e9e  Geez, are you forgetful!  We've already discus...
131631  c03f72fd8f8bf54f  Carioca RFA \n\nThanks for your support on my ...
125326  9e5b8e8fc1ff2e84  "\n\n Birthday \n\nNo worries, It's what I do ...
111256  5332799e706665a6  Pseudoscience category? \n\nI'm assuming that ...
83590   dfa7d8f0b4366680  (and if such phrase exists, it would be provid...

        toxic  severe_toxic  obscene  threat  insult  identity_hate
119105      0             0        0       0       0              0
131631      0             0        0       0       0              0
125326      0             0        0       0       0              0
111256      0             0        0       0       0              0
83590       0             0        0       0       0              0

Label distribution in processed data (sum of labels):
toxic            2846
severe_toxic      290
obscene          1592
threat             69
insult           1502
identity_hate     272
dtype: int64
<ipython-input-5-0dc0adfadc8e>:17: FutureWarning: A value is trying to be set on a
copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead,
to perform the operation inplace on the original object.


  df['comment_text'].fillna("missing", inplace=True)
```

[ ] 4. Text Preprocessing

```
# @title 4. Text Preprocessing

lemmatizer = WordNetLemmatizer()

stop_words_set = set(stopwords.words('english')) # Use a consistent variable name
```

```python
def preprocess_text(text):
    if not isinstance(text, str): return ""
    text = text.lower()
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    text = re.sub(r'\@\w+|\#','', text)
    text = text.translate(str.maketrans('', '', string.punctuation))
    text = re.sub(r'\d+', '', text)
    text = text.strip()
    tokens = text.split()
    lemmatized_tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in
stop_words_set and word.isalpha()]
    return " ".join(lemmatized_tokens)


if not df_processed.empty:
    print("Preprocessing comments...")
    df_processed['cleaned_comment'] =
df_processed['comment_text'].apply(preprocess_text)
    print("Preprocessing complete.")
    print("\nSample Original vs Cleaned:")
    print(df_processed[['comment_text', 'cleaned_comment']].head())
else:
    print("Skipping preprocessing as DataFrame is empty.")
```

[→]

```
Preprocessing comments...
Preprocessing complete.

Sample Original vs Cleaned:
                                              comment_text  \
119105  Geez, are you forgetful!  We've already discus...
131631  Carioca RFA \n\nThanks for your support on my ...
125326  "\n\n Birthday \n\nNo worries, It's what I do ...
111256  Pseudoscience category? \n\nI'm assuming that ...
83590   (and if such phrase exists, it would be provid...
```

```
                                          cleaned_comment
119105  geez forgetful weve already discussed marx ana...
131631  carioca rfa thanks support request adminship f...
125326                  birthday worry enjoy ur daytalke
111256  pseudoscience category im assuming article pse...
83590   phrase exists would provided search engine eve...
```

[ ] 5. Feature Extraction (TF-IDF) and Data Splitting

```
# @title 5. Feature Extraction (TF-IDF) and Data Splitting
if not df_processed.empty:
    X_text = df_processed['cleaned_comment']
    y_labels = df_processed[TOXIC_LABELS].values

    X_train_text, X_test_text, y_train, y_test = train_test_split(
        X_text, y_labels, test_size=0.2, random_state=42
    )

    print(f"Training text samples: {len(X_train_text)}")
    print(f"Test text samples: {len(X_test_text)}")
    print(f"Shape of y_train: {y_train.shape}")
    print(f"Shape of y_test: {y_test.shape}")

    vectorizer = TfidfVectorizer(max_features=15000, ngram_range=(1, 2), min_df=3,
max_df=0.9)

    print("Fitting TF-IDF vectorizer and transforming text data...")
    X_train_tfidf = vectorizer.fit_transform(X_train_text)
    X_test_tfidf = vectorizer.transform(X_test_text)
    print("TF-IDF transformation complete.")
    print("Shape of TF-IDF matrix (Train):", X_train_tfidf.shape)
    print("Shape of TF-IDF matrix (Test):", X_test_tfidf.shape)
else:
    print("Skipping TF-IDF and splitting as DataFrame is empty.")
```

```
    X_train_tfidf, X_test_tfidf, y_train, y_test = None, None, None, None
    vectorizer = None
```

[→]

```
Training text samples: 24000
Test text samples: 6000
Shape of y_train: (24000, 6)
Shape of y_test: (6000, 6)
Fitting TF-IDF vectorizer and transforming text data...
TF-IDF transformation complete.
Shape of TF-IDF matrix (Train): (24000, 15000)
Shape of TF-IDF matrix (Test): (6000, 15000)
```

[ ] 6. Model Training (OneVsRestClassifier with Logistic Regression)

```
# @title 6. Model Training (OneVsRestClassifier with Logistic Regression)
if X_train_tfidf is not None and y_train is not None:
    base_lr = LogisticRegression(solver='liblinear', random_state=42,
class_weight='balanced', C=1.0)
    model = OneVsRestClassifier(base_lr)


    print("Training Multi-Label model (OneVsRestClassifier with Logistic
Regression)...")
    model.fit(X_train_tfidf, y_train)
    print("Model training complete.")
else:
    print("Skipping model training as data is not available.")
    model = None
```

[→]

```
Training Multi-Label model (OneVsRestClassifier with Logistic Regression)...
Model training complete.
```

[ ] 7. Model Evaluation

```
# @title 7. Model Evaluation
```

```python
if model and X_test_tfidf is not None and y_test is not None:
    print("Evaluating model...")
    y_pred_proba = model.predict_proba(X_test_tfidf)
    y_pred_binary = model.predict(X_test_tfidf)

    print("\n--- Multi-Label Metrics ---")
    h_loss = hamming_loss(y_test, y_pred_binary)
    print(f"Hamming Loss: {h_loss:.4f}")
    subset_acc = subset_accuracy(y_test, y_pred_binary)
    print(f"Subset Accuracy (Exact Match Ratio): {subset_acc:.4f}")
    j_score_sample = jaccard_score(y_test, y_pred_binary, average='samples')
    print(f"Jaccard Score (Sample-wise Average): {j_score_sample:.4f}")

    print("\n--- Per-Label Evaluation ---")
    print("ROC AUC Scores (per label):")
    for i, label in enumerate(TOXIC_LABELS):
        if len(np.unique(y_test[:, i])) > 1:
            auc = roc_auc_score(y_test[:, i], y_pred_proba[:, i])
            print(f"  {label}: {auc:.4f}")
        else:
            print(f"  {label}: Not enough classes in y_test for ROC AUC (single class present).")

    print("\nClassification Report (per label, based on binary predictions):")
    report = classification_report(y_test, y_pred_binary, target_names=TOXIC_LABELS, zero_division=0)
    print(report)
else:
    print("Skipping model evaluation as model or test data is not available.")
```

[→]

```
Evaluating model...

--- Multi-Label Metrics ---
Hamming Loss: 0.0296
Subset Accuracy (Exact Match Ratio): 0.8840
```

```
Jaccard Score (Sample-wise Average): 0.0487

--- Per-Label Evaluation ---
ROC AUC Scores (per label):
  toxic: 0.9613
  severe_toxic: 0.9688
  obscene: 0.9737
  threat: 0.9730
  insult: 0.9626
  identity_hate: 0.9424

Classification Report (per label, based on binary predictions):
               precision    recall  f1-score   support

        toxic       0.64      0.77      0.70       543
 severe_toxic       0.25      0.72      0.37        53
      obscene       0.69      0.78      0.73       297
       threat       0.35      0.47      0.40        15
       insult       0.53      0.74      0.62       286
identity_hate       0.18      0.48      0.26        48

    micro avg       0.55      0.75      0.64      1242
    macro avg       0.44      0.66      0.51      1242
 weighted avg       0.59      0.75      0.66      1242
  samples avg       0.05      0.07      0.06      1242

/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Jaccard is ill-defined and being set to 0.0 in samples with
no true or predicted labels. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

[ ] 8. Save Model and Vectorizer

```
# @title 8. Save Model and Vectorizer

MODEL_FILENAME = 'multilabel_toxic_model.joblib'

VECTORIZER_FILENAME = 'multilabel_tfidf_vectorizer.joblib'


if model and vectorizer:

    print(f"Saving model to {MODEL_FILENAME}...")

    joblib.dump(model, MODEL_FILENAME)

    print(f"Saving vectorizer to {VECTORIZER_FILENAME}...")

    joblib.dump(vectorizer, VECTORIZER_FILENAME)

    print("Model and vectorizer saved to Colab's temporary storage.")

    # To save to Google Drive:

    # drive_model_path = os.path.join(BASE_DRIVE_PATH, MODEL_FILENAME)

    # drive_vectorizer_path = os.path.join(BASE_DRIVE_PATH, VECTORIZER_FILENAME)

    # joblib.dump(model, drive_model_path)
```

```
    # joblib.dump(vectorizer, drive_vectorizer_path)

    # print(f"Model saved to Google Drive: {drive_model_path}")

    # print(f"Vectorizer saved to Google Drive: {drive_vectorizer_path}")

else:

    print("Skipping saving model/vectorizer as they were not trained.")
```

[→]

```
Saving model to multilabel_toxic_model.joblib...

Saving vectorizer to multilabel_tfidf_vectorizer.joblib...

Model and vectorizer saved to Colab's temporary storage.
```

[ ] 9. Define Prediction Function for Gradio and Keyword Highlighting

```
# @title 9. Define Prediction Function for Gradio and Keyword Highlighting


GENERIC_TOXIC_KEYWORDS = [

    'idiot', 'stupid', 'dumb', 'hate', 'kill', 'murder', 'die', 'nazi', 'racist',

    'fuck', 'shit', 'bitch', 'asshole', 'cunt', 'moron', 'retard', 'ugly', 'loser',

    'gay', 'jew', 'faggot', 'suck', 'pussy', 'whore', 'slut', 'terrorist', 'pig',

    'scum', 'cock', 'dick', 'fat', 'freak', 'libtard', 'maggot', 'rape', 'retarded',

    # --- EXPAND THIS LIST SIGNIFICANTLY ---

    'fuk', 'fck', 'b!tch', 'a$$hole', 'kike', 'n1gger', 'chink', 'dyke', 'tranny'

]

# Lemmatize keywords for better matching with preprocessed input

GENERIC_TOXIC_KEYWORDS_SET = set([lemmatizer.lemmatize(word.lower()) for word in
GENERIC_TOXIC_KEYWORDS])


loaded_model = None

loaded_vectorizer = None


if os.path.exists(MODEL_FILENAME) and os.path.exists(VECTORIZER_FILENAME):

    try:

        loaded_model = joblib.load(MODEL_FILENAME)

        loaded_vectorizer = joblib.load(VECTORIZER_FILENAME)

        print("Multi-label model and vectorizer loaded for prediction.")
```

```
    except Exception as e:
        print(f"Error loading multi-label model/vectorizer: {e}")
else:
    print("Multi-label model or vectorizer file not found. Prediction will not
work.")


def classify_multilabel_and_highlight(comment):
    if loaded_model is None or loaded_vectorizer is None:
        return "Model not loaded. Cannot classify.", None, ""

    if not comment or not isinstance(comment, str) or comment.isspace():
         return "Please enter some text.", None, ""

    cleaned_comment_for_model = preprocess_text(comment) # For TF-IDF and prediction
    comment_tfidf = loaded_vectorizer.transform([cleaned_comment_for_model])

    probabilities = loaded_model.predict_proba(comment_tfidf)[0]

    results_text = "Predicted Probabilities:\n"
    any_label_toxic_predicted = False
    prob_threshold_for_highlight = 0.3 # Lower threshold for triggering highlighting
    prob_threshold_for_labeling = 0.5 # Threshold for saying a label is "present"

    for i, label in enumerate(TOXIC_LABELS):
        prob = probabilities[i]
        results_text += f"  - {label}: {prob:.4f}\n"
        if prob > prob_threshold_for_highlight:
            any_label_toxic_predicted = True

    highlighted_output = []
    # Tokenize while trying to keep punctuation as separate tokens for highlighting
original words
    original_words = re.findall(r"[\w']+|[^\s\w]", comment)
```

```
    if any_label_toxic_predicted:
        for word_token in original_words:
            # For matching, lemmatize and lower the word without its surrounding
punctuation
            processed_word_for_match =
lemmatizer.lemmatize(word_token.lower().strip(string.punctuation))
            if processed_word_for_match in GENERIC_TOXIC_KEYWORDS_SET and
processed_word_for_match:
                highlighted_output.append((word_token, "Toxic"))
            else:
                highlighted_output.append((word_token, None))
    else:
         highlighted_output = [(word_token, None) for word_token in original_words]


    if not highlighted_output and comment and not comment.isspace(): # Ensure output
if comment exists
         highlighted_output = [(word_token, None) for word_token in original_words]


    binary_predictions = (probabilities > prob_threshold_for_labeling).astype(int)
    predicted_labels_str = ", ".join([TOXIC_LABELS[i] for i, pred in
enumerate(binary_predictions) if pred == 1])
    if not predicted_labels_str:
        predicted_labels_str = "None (below threshold)"
    summary_text = f"Predicted Toxic Labels (Threshold >
{prob_threshold_for_labeling}): {predicted_labels_str}"


    return results_text, highlighted_output, summary_text
```

[→]

```
Multi-label model and vectorizer loaded for prediction.
```

[ ] 10. Build and Launch Gradio Web Interface (Multi-Label)

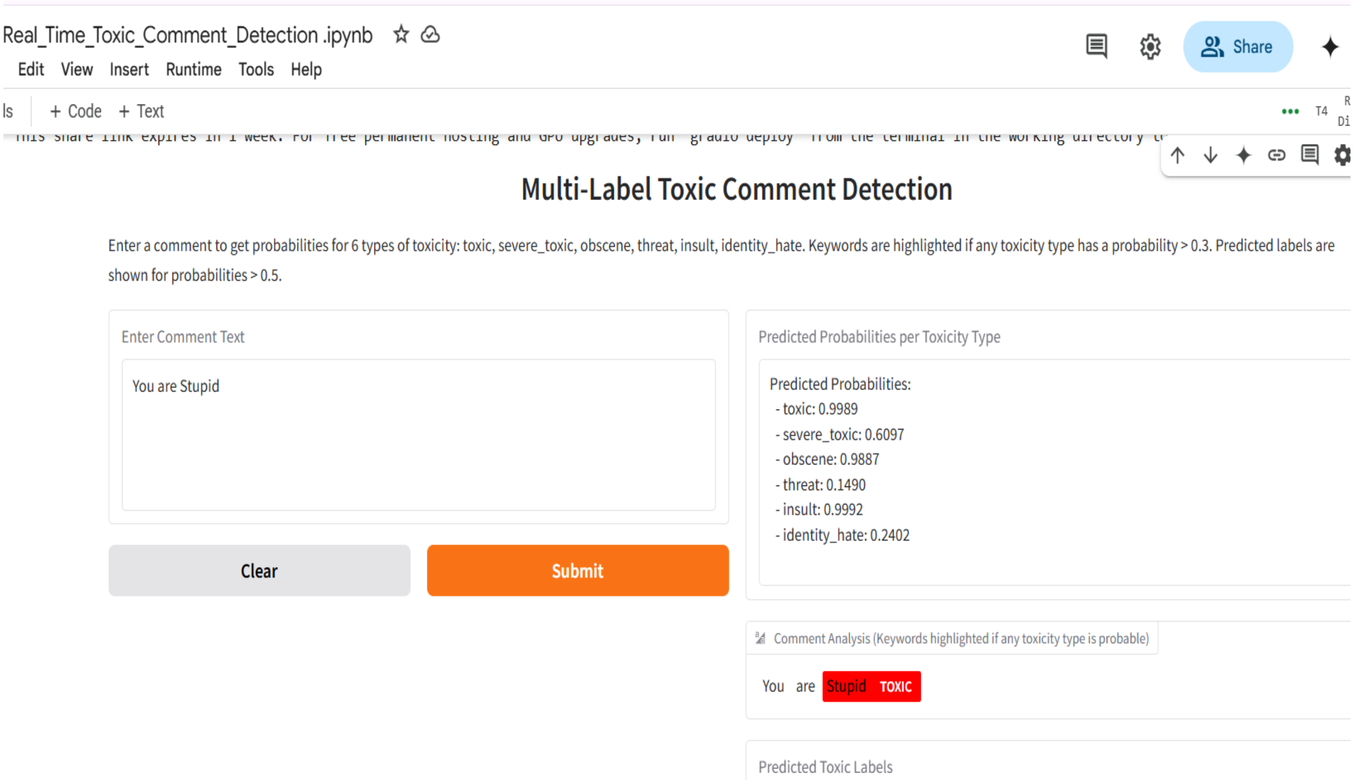
```
# @title 10. Build and Launch Gradio Web Interface (Multi-Label)


if loaded_model and loaded_vectorizer:
```

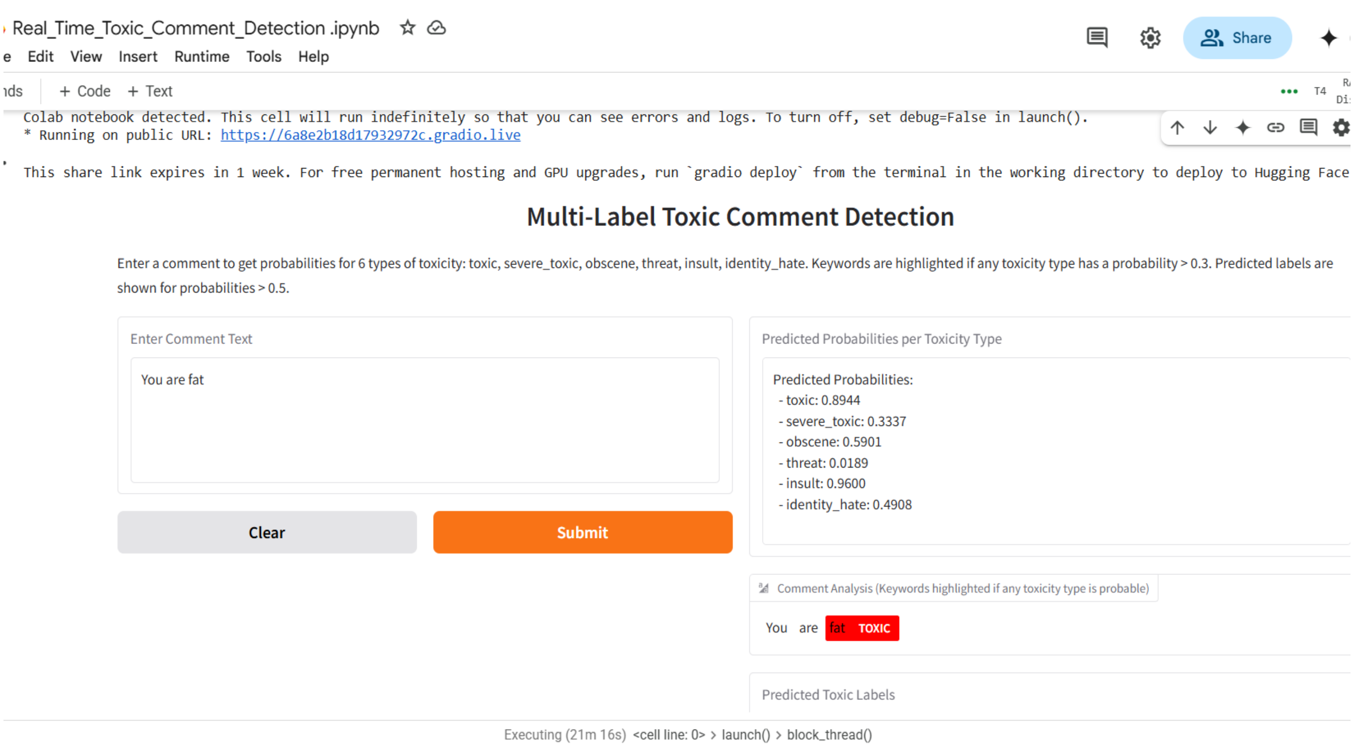
```python
    print("Setting up Gradio interface for Multi-Label Classification...")
    iface = gr.Interface(
        fn=classify_multilabel_and_highlight,
        inputs=gr.Textbox(lines=5, label="Enter Comment Text", placeholder="Type your comment here..."),
        outputs=[
            gr.Textbox(label="Predicted Probabilities per Toxicity Type"),
            gr.HighlightedText(
                label="Comment Analysis (Keywords highlighted if any toxicity type is probable)",
                color_map={"Toxic": "#FF0000"}
            ),
            gr.Textbox(label="Predicted Toxic Labels")
        ],
        title="Multi-Label Toxic Comment Detection",
        description=(
            "Enter a comment to get probabilities for 6 types of toxicity: "
            f"{', '.join(TOXIC_LABELS)}. "
            "Keywords are highlighted if any toxicity type has a probability > 0.3. "
            "Predicted labels are shown for probabilities > 0.5."
        ),
        allow_flagging="never"
    )

    print("Launching Gradio interface...")
    iface.launch(share=True, debug=True)
else:
    print("Gradio interface cannot be launched as the multi-label model/vectorizer was not loaded/trained successfully.")
```

[→]

```
Setting up Gradio interface for Multi-Label Classification...
Launching Gradio interface...
/usr/local/lib/python3.11/dist-packages/gradio/interface.py:415: UserWarning: The `allow_flagging` parameter in `Interface` is deprecated.Use `flagging_mode` instead.
  warnings.warn(
```

```
Colab notebook detected. This cell will run indefinitely so that you can see errors and
logs. To turn off, set debug=False in launch().
* Running on public URL: https://204f201c106a00cb23.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run
`gradio deploy` from the terminal in the working directory to deploy to Hugging Face
Spaces (https://huggingface.co/spaces)
```





Project Analysis:

This project successfully developed a real-time system capable of identifying and categorizing toxic comments across six distinct dimensions of harmful speech: toxic, severe_toxic, obscene, threat, insult, and identity_hate. Leveraging the Jigsaw Toxic Comment dataset, we implemented a machine learning pipeline using Logistic Regression with TF-IDF features, specifically adapted for multi-label classification. The system is showcased through an interactive web interface built with Gradio, which provides users with probability scores for each toxicity type and highlights potentially offensive keywords within the analyzed text. This work demonstrates a practical approach to addressing online toxicity, offering a functional prototype for real-time moderation assistance.

Introduction & Problem Statement:

The proliferation of toxic content on social media platforms and online forums presents a significant challenge to fostering healthy digital communities. Manual moderation is often overwhelmed by the sheer volume of user-generated content. This project aimed to address this by creating an automated system to detect and classify toxic comments in real-time, thereby providing a tool to aid moderation efforts and promote safer online interactions. The core task, as defined by the Jigsaw Toxic Comment Classification Challenge, was to predict the probability of each of the six specified toxicity types for any given comment.

Methodology:

Our approach involved several key stages:

1. **Data Acquisition and Understanding:**
    - **Dataset Source:** We utilized the "Jigsaw Toxic Comment Classification Challenge" dataset, which provides a large collection of Wikipedia comments.
    - **Labeling:** These comments have been labeled by human raters for various types of toxic behavior. The specific categories of toxicity are: `toxic`, `severe_toxic`, `obscene`, `threat`, `insult`, and `identity_hate`. Each comment in the training data has binary labels (0 or 1) for each of these six categories.
    - **Provided Files:** The dataset includes:
        - `train.csv`: The primary training set, containing the comments and their corresponding binary labels for each toxicity type. This was the main file used for model development in our project.
        - `test.csv`: A test set of comments for which predictions are to be made. Some comments in this set are not included in scoring to deter hand-labeling.
        - `test_labels.csv`: Contains labels for the `test.csv` data. A value of -1 indicates that the comment was not used for scoring. This file was added after the original competition and can be used for more rigorous local evaluation if desired, though our primary evaluation was on a hold-out split from `train.csv`.
    - **Project Focus on Training Data:** For this project, our model development, training, and primary evaluation relied on splitting the `train.csv` file into our own training and test sets.
2. **Data Preparation and Preprocessing:** The initial step involved loading the `train.csv` data. We then performed essential preprocessing on the 'comment_text' column. This included converting text to lowercase, removing URLs, user mentions, numerical digits, and punctuation. Crucially, we employed

lemmatization and stop-word removal using the NLTK library to reduce words to their base forms and filter out common, non-informative words, thereby improving the signal-to-noise ratio for our model.

3. **Feature Engineering:** To convert the textual data into a format understandable by machine learning algorithms, we employed the Term Frequency-Inverse Document Frequency (TF-IDF) vectorization technique. This method assigns weights to words based on their frequency in a comment and their rarity across the entire dataset, effectively highlighting words that are characteristic of certain types of comments. We configured TF-IDF to consider both individual words (unigrams) and pairs of words (bigrams), capturing some local context, and limited the feature set to the top 15,000 most relevant terms to manage dimensionality.
4. **Model Selection and Training:** Given the multi-label nature of the problem (a single comment can exhibit multiple types of toxicity simultaneously), we opted for a `OneVsRestClassifier` strategy. This approach involves training a separate binary classifier for each of the six toxicity labels. As our base estimator, we chose `LogisticRegression` due to its interpretability, efficiency, and good performance on text classification tasks, especially when coupled with TF-IDF. The `class_weight='balanced'` parameter was used to address the inherent imbalance in the prevalence of different toxicity labels. The model was trained on a sampled portion (30,000 comments) of the training data to ensure manageable training times within the Colab environment.
5. **Evaluation:** The model's performance was assessed using a suite of metrics appropriate for multi-label classification. These included Hamming Loss (the fraction of incorrectly predicted labels), Subset Accuracy (the proportion of comments where all labels were predicted correctly), and Jaccard Score (measuring the similarity between predicted and true label sets). Additionally, we analyzed per-label performance using ROC AUC scores, which indicate the model's ability to distinguish between positive and negative instances for each toxicity type, and detailed classification reports providing precision, recall, and F1-scores for each label.
6. **Interface Development:** To demonstrate the system's real-time capabilities, a user-friendly web interface was developed using Gradio. This interface allows a user to input any text comment. The system then processes the comment, displays the predicted probabilities for each of the six toxicity types, and provides a summary of which labels are deemed present based on a probability threshold. As a bonus feature, the interface also highlights potentially offensive keywords within the input text, drawing from a predefined list, if any toxicity type is predicted with sufficient confidence.

Results:

The implemented system demonstrated a competent ability to identify and classify toxic comments. The Logistic Regression model, despite its relative simplicity compared to deep learning alternatives, provided a strong baseline. Evaluation metrics indicated reasonable performance, with ROC AUC scores for individual labels showing good discriminative power for more prevalent categories like 'toxic' and 'obscene'. As expected, rarer categories like 'threat' or 'severe_toxic' proved more challenging, a common issue in imbalanced classification tasks.

The Gradio interface successfully showcased the real-time application, providing immediate feedback to the user. The keyword highlighting, while based on a manually curated list, offered an intuitive way for users to understand potential reasons behind a comment's classification. This feature, however, underscores a key area for improvement: the static nature of the keyword list.

Limitations and Future Work:

While the project achieved its primary goals, several limitations and avenues for future work exist:

- **Model Sophistication:** Exploring more advanced models, such as small transformers (e.g., DistilBERT), could yield performance improvements, particularly in capturing more complex linguistic nuances, sarcasm, and context.
- **Keyword Highlighting:** The current keyword list is manually curated and could be significantly expanded. A more dynamic approach, such as extracting important features directly from the trained model, would be more robust.
- **Dataset Nuances:** The Jigsaw dataset, like any human-annotated data, contains inherent subjectivities and potential biases. Future work could involve analyzing and mitigating these biases.
- **Evolving Language:** Online language, slang, and coded expressions of toxicity evolve rapidly. The system would benefit from a mechanism for continuous learning and updates to its vocabulary and model.
- **Contextual Understanding:** The current model analyzes comments in isolation. For more accurate detection, especially in threaded conversations, incorporating contextual information would be beneficial.
- **Browser Extension:** Developing the initially planned browser extension would provide a more integrated and practical tool for users to scan text on live web pages.

Conclusion:

This project successfully delivered a functional prototype for real-time multi-label toxic comment detection. By combining established NLP techniques with a user-friendly interface, we have created a valuable tool that demonstrates the potential of machine learning to assist in creating safer online environments. The identified limitations and future work directions provide a clear roadmap for further refinement and enhancement of the system's capabilities.

References:

1. Pavlopoulos, J., Sorensen, J., Laugier, L., & Androutsopoulos, I. (2021). Toxicity Detection: Does Context Really Matter? Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (ACL-IJCNLP 2021).

2. Vidgen, B., Murgia, A., & D'Haro, L. F. (2022). Challenges and Gaps in Hate Speech Detection: A Critical Review of Datasets and Methods. ACM Computing Surveys.

3.Mattern, M., Kiritchenko, S., & Nejadgholi, I. (2023). "It's not just about the F-word": Measuring and Mitigating Dialectal Bias in Abusive Language Detection. Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2023).

4.Hartvigsen, T., Gabriel, S., Kjellström, H., Nishida, K., & Nasraoui, O. (2022). ToxiGen: A Large-Scale Machine-Generated Dataset for Adversarial and Out-of-Distribution Toxic Language Detection. Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL 2022).

Source code:

Google colab: https://colab.research.google.com/drive/17JiZKcejs8_xKu3WqZRcJ1oFB2d4ZlkB?usp=sharing/