

**VIT-AP**  
**UNIVERSITY**

**TITLE: -** AI Legal Sentiment Analyzer

**NAME: -** ALUR JAYAPRAKASH

**Roll NO: -** 23BCE9945

**College Name: -** Vellore Institute of  
Technology Andhra Pradesh (VIT-AP),  
Amaravati

**Branch: -** Computer science and Engineering

**Date of Submission: -** 30-06-2025

## Table of Content

S.NO	Topic	Page.NO.
1.	Introduction	3
2.	Objectives	3
3.	Tools	3
4.	Methodology	4-5
5.	Code snippets	6-13
6.	GitHub Link	14
7.	Conclusion	14

## 1. INTRODUCTION

This project is an AI-based tool that analyzes the sentiment of legal documents. It tells whether the text is positive, negative, or neutral. The tool also gives a short summary of the document to help understand it quickly. It's designed to save time for legal teams by reducing manual document review.

## 2. OBJECTIVES

To build an AI system that automatically detects sentiment in legal texts. It also provides summaries to help users quickly grasp the overall meaning of the document.

## 3. TOOLS

Category	Tools/Technologies
Programming Language	Python
Libraries	pandas, sklearn, collections, countvectorizer
AI Model	IBM Watson FLAN-T5-XXL
Platform	IBM Watson Machine Learning
Cloud Storage	IBM Cloud Object Storage (COS)
IDEs	Google Colab

## 4. Methodology:

### 1. Data Collection:

Legal documents or case summaries are collected and stored in IBM Cloud Object Storage (COS) in CSV format.

### 2. Data Preprocessing:

- Dataset is loaded into the project.
- Sentiment labels are mapped to positive, neutral, negative.

### 3. Model Setup:

IBM Watson's FLAN-T5-XXL model is used as the foundation model for generating sentiment predictions.

### 4. Few-Shot Learning:

Example sentences with known sentiments are provided to guide the model (few-shot prompting).

### 5. Prediction Generation:

Sentiment is predicted for each sentence or document using zero-shot or few-shot prompts.

### 6. Summarized Insights:

- Sentiment distribution (% of positive/neutral/negative).
- Frequent negative terms extracted using NLP techniques.

### 7. Output Generation:

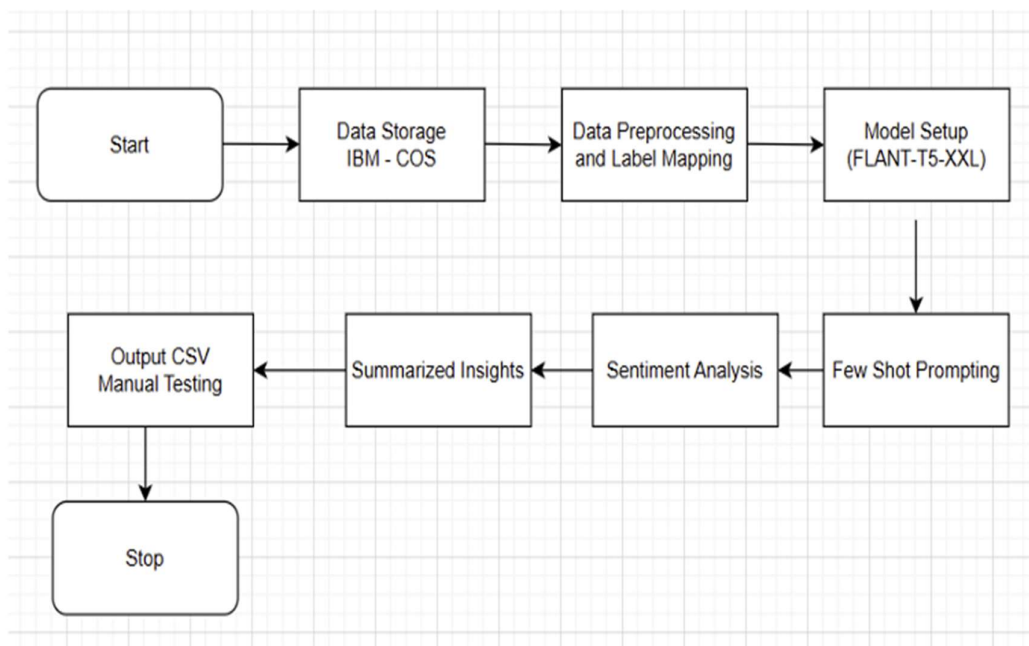
Results are saved into a CSV file for reference.

Manual input analysis is also supported interactively via the console.

## Flowchart:

This flowchart outlines the working of the AI Legal Sentiment Analyzer.

It starts by accessing legal data stored in IBM Cloud Object Storage (COS). The data undergoes preprocessing and label mapping to prepare for analysis. The FLAN-T5-XXL model is then initialized with few-shot prompting for better context. The model performs sentiment analysis, followed by summarized insights generation. Finally, results are saved as CSV and can also be tested interactively by the user.



## 5. Code Snippets

1. The required libraries for data handling, machine learning, and IBM Watson integration were successfully installed. These packages enable connecting to IBM Cloud, running the foundation model, and processing legal text for sentiment analysis.



```

✓ 16s !pip install wget | tail -n 1
!pip install scikit-learn | tail -n 1
!pip install "ibm-watson-machine-learning>=1.0.310" | tail -n 1

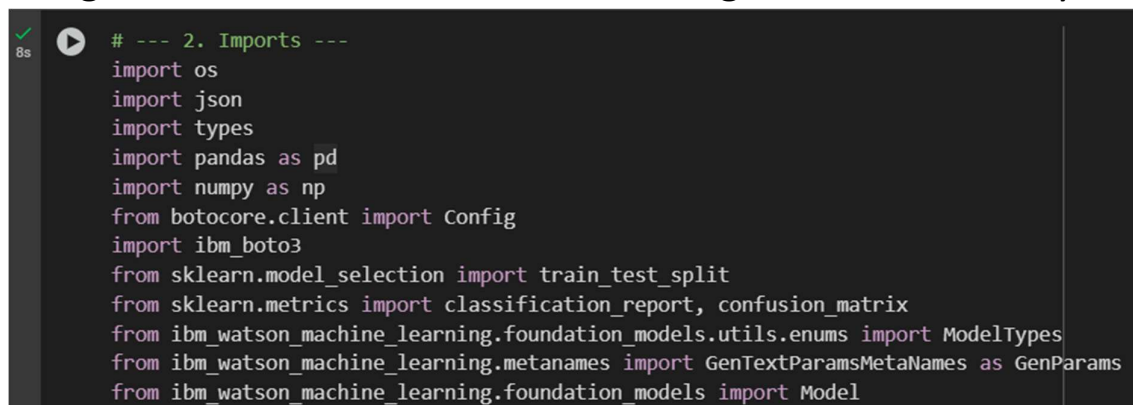
Requirement already satisfied: wget in /usr/local/lib/python3.11/dist-packages (3.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.11/dist-packages (from lomond->ibm-watson-machine-learning>=1.0.310) (1.17.0)

✓ 9s [2] !pip install boto3 | tail -n 1

Successfully installed boto3-1.38.41 botocore-1.38.41 s3transfer-0.13.0

```

2. This section imports all necessary libraries for data handling, model usage, evaluation, and IBM Watson integration to build and evaluate the legal sentiment analyzer.



```

✓ 8s # --- 2. Imports ---
import os
import json
import types
import pandas as pd
import numpy as np
from botocore.client import Config
import ibm_boto3
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from ibm_watson_machine_learning.foundation_models.utils.enums import ModelTypes
from ibm_watson_machine_learning.metanames import GenTextParamsMetaNames as GenParams
from ibm_watson_machine_learning.foundation_models import Model

```

3. This section sets up the **IBM Cloud credentials** (API key and project ID) required to authenticate and connect to **IBM Watson Machine Learning** services for model execution.

```

✓ 20s [4] # --- 3. Set up IBM Credentials ---
      credentials = {
          "url": "https://us-south.ml.cloud.ibm.com",
          "apikey": input("Enter your IBM API Key: ").strip()
      }

      ↵ Enter your IBM API Key: gvDYe1QwNOVzdxcdn5W7inazV8tXkD5JD2ce6KwdbnDo

✓ 10s [5] project_id = os.environ.get("PROJECT_ID", input("Enter your project_id (or press Enter if none): "))

      ↵ Enter your project_id (or press Enter if none): 281483ad-0a47-4482-86f4-c610b9535653

```

4. This section establishes a connection to **IBM Cloud Object Storage (COS)** using the provided API key. It authenticates with IBM's IAM service and specifies the endpoint for the **US-South** region. The user inputs the **COS bucket name** and **dataset file name** to locate the dataset. This enables the program to access the CSV file needed for legal sentiment analysis.

```

▶ # --- 4. Connect to COS & Load Dataset ---
  cos_client = ibm_boto3.client(service_name='s3',
                                ibm_api_key_id=credentials['apikey'],
                                ibm_auth_endpoint='https://iam.cloud.ibm.com/oidc/token',
                                config=Config(signature_version='oauth'),
                                endpoint_url='https://s3.us-south.cloud-object-storage.appdomain.cloud'
                                )

[7] bucket = input("Enter your COS bucket name: ").strip()
      object_key = input("Enter your dataset file name (e.g., legal_sentiment.csv): ").strip()

      ↵ Enter your COS bucket name: bucket-1pem2ak6shsthr1
         Enter your dataset file name (e.g., legal_sentiment.csv): legal_sentiment_dataset (1).csv

```

5. This code fetches the dataset file from IBM COS, reads it into a **Pandas DataFrame**, and displays the first few rows. The

dataset contains **legal sentences** along with their corresponding **sentiment labels**.

```
[14] body = cos_client.get_object(Bucket=bucket, Key=object_key)['Body']
      if not hasattr(body, "_iter_"):
          def _iter_(self): yield from self
          body._iter_ = types.MethodType(_iter_, body)

[15] data = pd.read_csv(body)
      data.head()
```

	ID	Phrase	Sentiment
0	1	The plaintiff's claims are dismissed.	-1
1	2	The contract is deemed valid and enforceable.	1
2	3	The appeal is denied due to lack of merit.	-1
3	4	The legal team submitted additional evidence.	0
4	5	The appeal is under consideration.	0

6. This section cleans the dataset by removing missing entries and mapping sentiment labels (-1, 0, 1) to their respective **negative**, **neutral**, and **positive** text labels. It then displays the count of each sentiment category in the dataset.

```
# --- 5. Data Cleaning ---
label_map = {-1: 'negative', 0: 'neutral', 1: 'positive'}
data = data.dropna(subset=['Phrase', 'Sentiment'])
data['Sentiment'] = data['Sentiment'].astype(int)
data['Sentiment'] = data['Sentiment'].map(label_map)
print(data['Sentiment'].value_counts())
```

```
Sentiment
positive    184
negative    167
neutral     149
Name: count, dtype: int64
```

7. Splitting the Dataset for training and testing.

```
[17] # --- 6. Train-Test Split ---
      data_train, data_test = train_test_split(data, test_size=0.3, random_state=42, stratify=data['Sentiment'])
```

8. This section initializes the **IBM FLAN-T5-XXL** foundation model with specified parameters for text generation. It



connects the model to the project using the provided credentials and project ID for performing sentiment analysis.

```

0s # --- 7. Setup IBM Foundation Model ---
    parameters = {
        GenParams.DECODING_METHOD: "greedy",
        GenParams.RANDOM_SEED: 42,
        GenParams.REPETITION_PENALTY: 1,
        GenParams.MIN_NEW_TOKENS: 1,
        GenParams.MAX_NEW_TOKENS: 5
    }

2s [19] model = Model(
        model_id=ModelTypes.FLAN_T5_XXL,
        params=parameters,
        credentials=credentials,
        project_id=project_id
    )
    model.get_details()

```

Showing all the details of the model FLAN\_T5\_XXL

```

{'model_id': 'google/flan-t5-xxl',
 'label': 'flan-t5-xxl-11b',
 'provider': 'Google',
 'source': 'Hugging Face',
 'functions': [{'id': 'text_generation'}],
 'short_description': 'flan-t5-xxl is an 11 billion parameter model based on the Flan-T5 family.',
 'long_description': 'flan-t5-xxl (11B) is an 11 billion parameter model based on the Flan-T5 family. It is a pretrained T5 - an encoder-decoder model pre-trained on a mixture of supervised / unsupervised tasks converted into a text-to-text format, and fine-tuned on the Fine-tuned Language Net (FLAN) with instructions for better zero-shot and few-shot performance.',
 'terms_url': 'https://huggingface.co/google/flan-t5-xxl/blob/main/README.md',
 'input_tier': 'class_2',
 'output_tier': 'class_2',
 'number_params': '11b',
 'min_shot_size': 0,
 'task_ids': ['question_answering',
 'summarization',
 'retrieval_augmented_generation',
 'classification',
 'generation',
 'extraction'],
 'tasks': [{'id': 'question_answering', 'ratings': {'quality': 4}},
 {'id': 'summarization', 'ratings': {'quality': 4}},
 {'id': 'retrieval_augmented_generation', 'ratings': {'quality': 3}},
 {'id': 'classification', 'ratings': {'quality': 4}},
 {'id': 'generation'}],

```

9. This part prepares **few-shot examples** by selecting sample sentences for each sentiment category from the training data.

These examples help guide the foundation model during prediction for better accuracy.

```

0s # --- 8. Prepare Few-Shot Examples ---
few_shot_examples = []
for sentiment in data_train['Sentiment'].unique():
    samples = data_train[data_train['Sentiment'] == sentiment].sample(2)
    for _, row in samples.iterrows():
        few_shot_examples.append(f"sentence: {row['Phrase']}\nsentiment: {row['Sentiment']}")

few_shot_context = "\n".join(few_shot_examples)

[23] print(few_shot_context)

```

```

↵ sentence: The case was transferred to another jurisdiction.
    sentiment: neutral
    sentence: The court reviewed the documentation.
    sentiment: neutral
    sentence: The contract is deemed valid and enforceable.
    sentiment: positive
    sentence: The court finds in favor of the plaintiff.
    sentiment: positive
    sentence: The plaintiff's claims are dismissed.
    sentiment: negative
    sentence: The contract is declared void.
    sentiment: negative

```

10. This block generates sentiment predictions by giving the model an instruction along with each input phrase. The model outputs 'positive', 'negative', or 'neutral', which is then cleaned and stored for evaluation.

```

0s instruction = "Determine the sentiment of the following sentence (as 'positive', 'negative', or 'neutral'). Use the examples below as reference:\n" + few_shot_context + "\n"

# --- 9. Generate Predictions ---
results = []
for text in data_test['Phrase']:
    prompt = instruction + f"\nsentence: {text}\nsentiment:"
    result = model.generate(prompt)[0]['generated_text']
    results.append(result.strip().lower())

```

11. This code compares the predicted sentiments (`y_pred`) with the actual labels (`y_true`) and prints a **classification report** using `classification_report()` from `scikit-learn`. The classification report shows **precision, recall, F1-score, and support** for each class, helping evaluate how well the model predicts each sentiment category. The model achieved **high accuracy (93%)**, perfect precision on neutral/positive, but slightly lower recall for positive class — suggesting some positive examples were misclassified.

```
[30] # --- 10. Evaluation ---
      y_true = data_test['Sentiment'].values
      y_pred = results

      print("\nClassification Report:")
      print(classification_report(y_true, y_pred))
```

	precision	recall	f1-score	support
negative	0.83	1.00	0.91	50
neutral	1.00	1.00	1.00	45
positive	1.00	0.82	0.90	55
accuracy			0.93	150
macro avg	0.94	0.94	0.94	150
weighted avg	0.94	0.93	0.93	150

12. This code generates a **confusion matrix** to show how many samples were correctly or incorrectly predicted for each sentiment class. From the output, the model perfectly predicted all *negative* and *neutral* samples.

```
[27] print("\nConfusion Matrix:")
      print(confusion_matrix(y_true, y_pred))
```

Confusion Matrix:		
[50	0	0]
[ 0	45	0]
[10	0	45]]

13. This code calculates the **percentage distribution** of each predicted sentiment by counting how many times each label

appears in `y_pred`. The output shows that 40% of the predicted sentences were negative, while neutral and positive sentiments each made up 30%

```

0s from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer
# --- Sentiment Distribution ---
sentiment_counts = Counter(y_pred)
total = sum(sentiment_counts.values())
summary_distribution = {sentiment: f"{(count/total)*100:.2f}%" for sentiment, count in sentiment_counts.items()}
print("\nSentiment Distribution (%):")
print(json.dumps(summary_distribution, indent=2))

```

```

Sentiment Distribution (%):
{
  "neutral": "30.00%",
  "positive": "30.00%",
  "negative": "40.00%"
}

```

14. This code uses `CountVectorizer` to extract the **top 10 most frequent unigrams and bigrams** from the predicted negative sentences. The result shows common legal terms like “*claims dismissed*”, “*void*”, and “*denied*”, which frequently appear in negative sentiment phrases.

```

0s # --- Top Words/Phrases for Negative Sentences ---
negative_texts = data_test['Phrase'][np.array(y_pred) == 'negative']
vectorizer = CountVectorizer(stop_words='english', ngram_range=(1,2), max_features=10)
X = vectorizer.fit_transform(negative_texts)
top_negative_words = vectorizer.get_feature_names_out()
print("\nTop Words/Phrases in Negative Sentences:")
print(top_negative_words)

```

```

Top Words/Phrases in Negative Sentences:
['claims' 'claims dismissed' 'contract declared' 'declared void' 'denied'
 'dismissed' 'motion' 'plaintiff' 'plaintiff claims' 'void']

```

15. This code generates a **summary statement** that identifies the most common sentiment in the document and its

percentage. It also lists the **frequent negative phrases**, giving a concise overview of the dominant tone and legal themes in the analyzed text.

```
[38] # --- Summary Statement ---
majority_sentiment = max(sentiment_counts, key=sentiment_counts.get)
summary_statement = (
    f"The overall sentiment of this document is predominantly\n **{majority_sentiment}** "
    f"({summary_distribution[majority_sentiment]} of sentences).\n "
    f"Frequent negative themes include: {' '.join(top_negative_words)}."
)
print("\nSummary Statement:")
print(summary_statement)
```

Summary Statement:  
 The overall sentiment of this document is predominantly  
 \*\*negative\*\* (40.00% of sentences).  
 Frequent negative themes include: claims, claims dismissed, contract declared, declared void, denied, dismissed, motion, plaintiff, plaintiff claims, void.

16. Output Saved to the file and manual testing is done.

```
[23] # --- 11. Output Results to File ---
output_df = data_test.copy()
output_df['Predicted Sentiment'] = y_pred
output_df.to_csv("legal_sentiment_results.csv", index=False)
print("Results saved to legal_sentiment_results.csv")
```

Results saved to legal\_sentiment\_results.csv

```
# --- 12. Interactive Testing ---
while True:
    custom_input = input("\nEnter a legal sentence to analyze sentiment (or type 'exit'): ").strip()
    if custom_input.lower() == 'exit':
        break
    custom_prompt = instruction + f"\nsentence: {custom_input}\nsentiment:"
    response = model.generate(custom_prompt)['results'][0]['generated_text']
    print(f"Predicted Sentiment: {response.strip().lower()}")
```

Enter a legal sentence to analyze sentiment (or type 'exit'): the food is very good  
 Predicted Sentiment: positive

Enter a legal sentence to analyze sentiment (or type 'exit'): the food is good  
 Predicted Sentiment: positive

Enter a legal sentence to analyze sentiment (or type 'exit'): the food is ok  
 Predicted Sentiment: neutral

Enter a legal sentence to analyze sentiment (or type 'exit'): the food is bad  
 Predicted Sentiment: negative

Enter a legal sentence to analyze sentiment (or type 'exit'): exit

## 6. GitHub link

<https://github.com/Jayaprakash9945/IBM-FINAL-PROJECT>

## 7. Conclusion

The AI Legal Sentiment Analyzer successfully automates the task of identifying sentiment in legal documents. It classifies text into positive, negative, or neutral, reducing the need for manual review by legal teams. The tool also generates summarized insights, including sentiment distribution and key negative terms. This helps users quickly understand the overall tone and focus of legal texts. By leveraging IBM Watson's FLAN-T5-XXL model, the system ensures high-quality predictions. The integration with IBM Cloud Object Storage makes handling datasets efficient and secure. Interactive input support further enhances its usability for analyzing custom legal sentences. Overall, the project provides a smart, time-saving solution for sentiment analysis in legal workflows.