SUMMER TRAINING/INTERNSHIP

PROJECT REPORT

(Term June -July 2025)

(TITLE OF THE PROJECT/ INTERNSHIP COMPANY)

Submitted by

Sankarasetty Jaya Sri Ram
Registration Number: 12300855

Course Code : PETV79

Under the Guidance of

Mahipal Singh Papola

School of Computer Science and Engineering

Lovely Professional University, Punjab

BONAFIDE CERTIFICATE

This is to certify that the project entitled **"Legal Document Analyser"** is a bonafide work carried out by **Sankarasetty Jaya Sri Ram**, a student of **B.Tech CSE** at **Lovely Professional University**, under my supervision in partial fulfillment of the requirements for the award of the degree.

The project embodies the original work done by the student during the Summer Internship Course Project in the academic session **2024–2025**.

SIGNATURE

<<<Name of the Supervisor>>>>

Sankarasetty Jaya Sri Ram

SIGNATURE

<<Signature of the Head of the Department>>

SIGNATURE

<<Name>>

HEAD OF THE DEPARTMENT

<<<Signature of the Supervisor>>

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my mentor, Mahipal Singh Papola, for his constant guidance and support throughout this project.

I am also thankful to my family and friends for their encouragement and moral support during the course of this work.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Company Profile:

Lovely Professional University Skill Development Courses

1.2 Overview of Training Domain:

As part of this course, I explored a wide range of topics in the field of Artificial Intelligence and Machine Learning. The training started with the basics of classical machine learning, such as understanding different algorithms like linear regression, decision trees, and clustering. I learned how to evaluate models properly and how to handle challenges like overfitting and bias. Moving forward, the course covered deep learning concepts and NLP, where I studied how neural networks work and how they can be used for different types of data, including images and text and how machines understand natural languages. I also got an introduction to some of the latest advancements in generative AI, like large language models and prompt engineering.

1.3 Objective of the Project:

The objective is to build a clause analyzer that can classify clauses from legal documents into predefined types and highlight named entities. The goal is to apply the knowledge of Machine Learning, Natural Language Processing — like your Legal Document Clause Analyzer — to bridge theory and practice.

CHAPTER 2: TRAINING OVERVIEW

2.1 Tools & Technologies Used:

Technology-Python, Numpy, scikit-learn, spaCy, Streamlit, pandas, seaborn, matplotlib, Pytorch, TensorFlow.

Tools - Jupyter Notebook, Google Colab

2.2 Areas Covered During Training:

Details the 3 layers:

1. Classical ML: linear regression, decision trees, clustering, metrics, math.
2. Deep Learning: neural networks, CNNs, RNNs, Transformers.
3. Generative AI: LLMs, Vision Models, RAG, Prompt Engineering.

2.3 Weekly Work Summary:

2.3.1 Week 1

Artificial Intelligence (AI) is the broad field of making machines act smartly. **Machine Learning (ML)** is a branch of AI that trains machines to learn from data. **Deep Learning (DL)** is a type of ML that uses large neural networks to solve complex tasks like image or speech recognition. **Data Science (DS)** is the process of collecting, analyzing, and interpreting data to find useful patterns and make decisions.

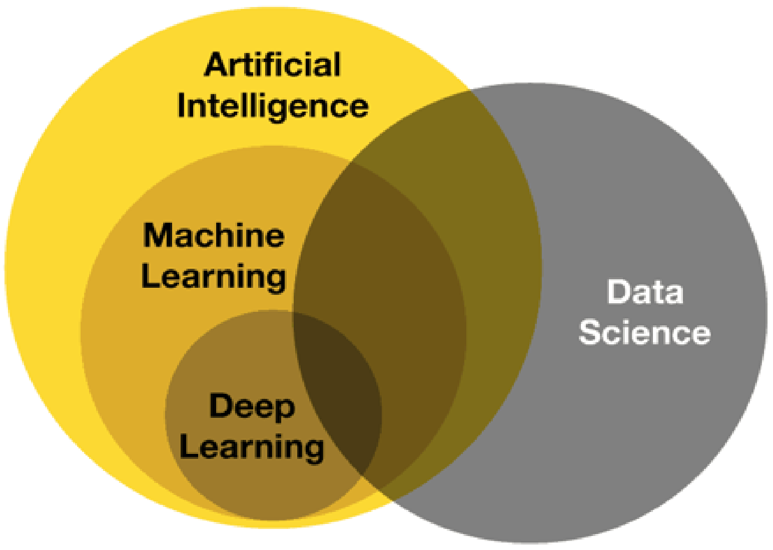


Figure 1: Overview of relationship between AI,ML,DL,DS

Data Cleaning

Data cleaning is the process of fixing or removing incorrect, incomplete, or duplicate data from a dataset to make it reliable for analysis. It includes steps like handling missing values, correcting errors, removing duplicates, and converting data into the right format. Good data cleaning helps improve the quality of the dataset, which leads to better and more accurate machine learning models. Without proper cleaning, models may learn patterns that are misleading or irrelevant.

NumPy

NumPy (Numerical Python) is an open-source Python library used for performing numerical operations on large datasets. It provides support for multidimensional arrays and matrices, making mathematical computations faster and more efficient compared to regular Python lists. NumPy is especially important in Machine Learning and Deep Learning because these fields involve complex matrix and vector operations. By using NumPy, large amounts of data can be processed quickly, which helps speed up model training and testing.

To download libraries in python – pip install numpy

To use in the code – import numpy as np

Overfitting, Underfitting, and Best Fit

In machine learning, fitting a model means finding the best pattern that represents the data. Overfitting happens when a model is too complex and learns the training data too well, including the noise — this makes it perform poorly on new data. Underfitting occurs when the model is too simple to capture the real pattern, resulting in high errors. A good fit strikes the balance between the two — it accurately captures trends in the training data and generalizes well to new, unseen data.

SQL vs Pandas

SQL stands for Structured Query Language, which is a programming language used to manage and manipulate relational databases. Pandas is an open-source Python library primarily used for data analysis and manipulation. Both SQL and pandas are tools for working with structured data, but they are used differently. SQL is a query language mainly used to manage large databases stored on servers. It is good for tasks like retrieving records, joining tables, and filtering data using queries. Pandas is a Python library that works well for data manipulation and analysis in scripts or notebooks. It allows easy filtering, grouping, merging, and reshaping of data on a local machine. Knowing when to use SQL or pandas helps make data work faster and more efficient.

Machine Learning and Its Types

Machine Learning is a part of Artificial Intelligence that allows computers to learn patterns from data and make predictions or decisions without being explicitly programmed. It can be divided into three main types: **Supervised Learning**, where models learn from labeled data; **Unsupervised Learning**, where models find hidden patterns in data without labels; and **Reinforcement Learning**, where models learn by interacting with an environment and improving through rewards or feedback.

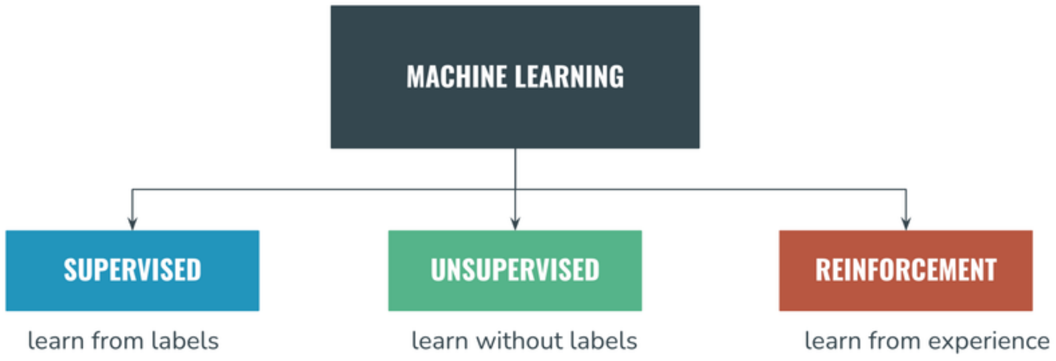


Figure 2: Types of Machine Learning

2.3.2 Week 2

Scikit-learn

Scikit-learn is a popular Python library that provides simple and efficient tools for machine learning. It offers a wide range of algorithms for classification, regression, clustering, and model selection. It also makes it easy to split data, build pipelines, and evaluate models. **Syntax:** from sklearn.model_selection import train_test_split

Linear Regression

Linear Regression is a basic and widely used technique in supervised learning for predicting continuous values. It tries to find the best-fit straight line through the data points that minimizes the difference between actual and predicted values. For example, it can be used to predict house prices based on features like area, location, and number of rooms.

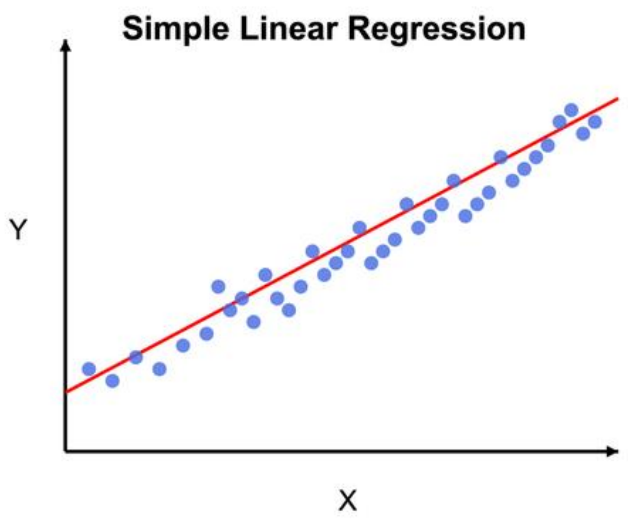


Figure 3: Linear Regression Model Graph

Multiple Linear Regression

1. Uses two or more independent variables to predict a single continuous target.
2. Fits a plane or hyperplane instead of just a line.
3. Example: Predicting house prices using area, number of rooms, and location.

Label Encoding and One-Hot Encoding

In machine learning, algorithms usually work with numerical data. Label Encoding is a method of converting categorical text labels into numbers by assigning a unique integer to each category. However, this can sometimes mislead the model into thinking that the categories have an order. One-Hot Encoding solves this by creating new binary columns for each category, indicating the presence or absence of that category. Both techniques help prepare data for training models.

Bias–Variance Tradeoff

The Bias–Variance Tradeoff explains how a model's complexity affects its performance. Bias is the error that comes from overly simple models that fail to capture the true patterns in the data, leading to underfitting. Variance is the error from overly complex models that learn noise in the training data, causing overfitting. The goal is to find the right balance where the model generalizes well to new, unseen data without being too simple or too complex.

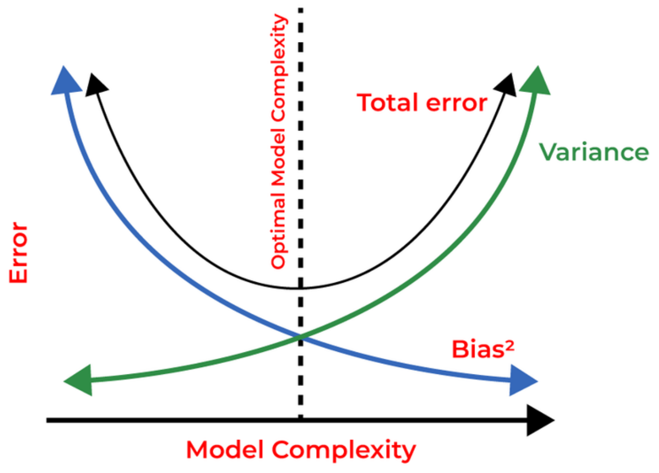


Figure 4: Bias Variance Tradeoff

Evaluation Metrics (Regression and Classification)

For Regression:

1. **Mean Squared Error (MSE):** Average of the squared differences between actual and predicted values. Smaller MSE means better performance.
2. **Mean Absolute Error (MAE):** Average of the absolute differences between actual and predicted values. Shows how far predictions are from real values.
3. **R-squared (R^2):** Shows how well the model explains the variation in the target variable. Value closer to 1 means the model fits the data well.

For Classification:

1. **Accuracy:** Percentage of total predictions that are correct.
2. **Precision:** Out of all items predicted as positive, how many are actually positive.
3. **Recall:** Out of all actual positive items, how many did the model correctly identify.
4. **F1-Score:** Balance between precision and recall; useful when data is imbalanced.

Confusion Matrix

Confusion Matrix is a table used to evaluate the performance of a classification model. It shows the number of correct and incorrect predictions for each class by comparing actual and predicted values. It helps in understanding where the model is making mistakes, such as false positives or false negatives, and gives a clearer picture than just using overall accuracy.

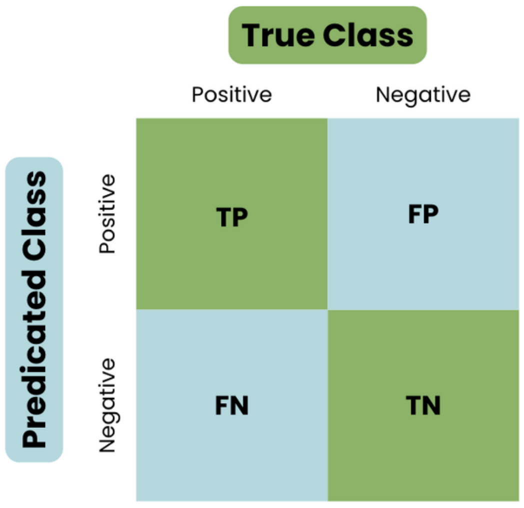


Figure 5: Confusion Matrix

2.3.3 Week 3

AUC-ROC

AUC is a useful measure for comparing the performance of two different models, as long as the dataset is roughly balanced. The model with greater area under the curve is generally the better one.

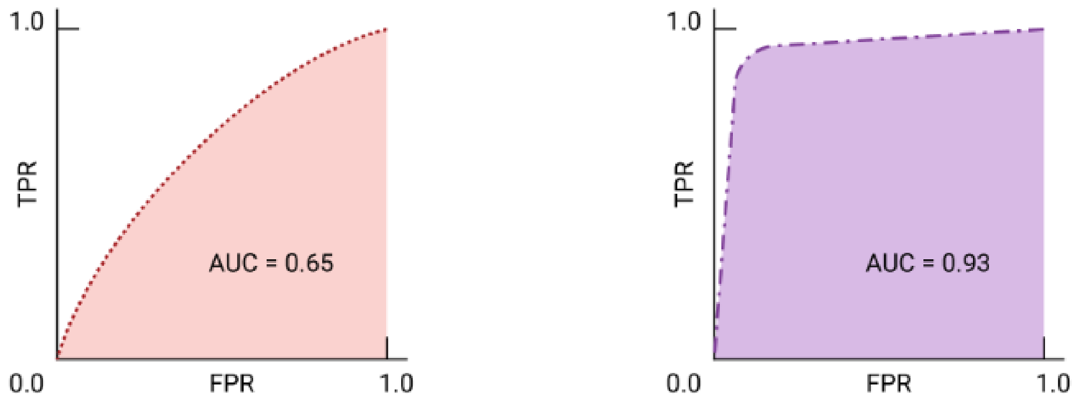


Figure 6: ROC and AUC of two hypothetical models. The curve on the right, with a greater AUC, represents the better of the two models.

Classification Models

Classification models are used to predict categorical outcomes, like whether an email is spam or not. These models learn patterns from labeled data and classify new data points into one of the predefined categories.

Logistic Regression

Logistic Regression is a simple but powerful classification algorithm used for binary or multi-class problems. Instead of predicting a continuous value, it estimates the probability that a given input belongs to a certain class, using the logistic (sigmoid) function.

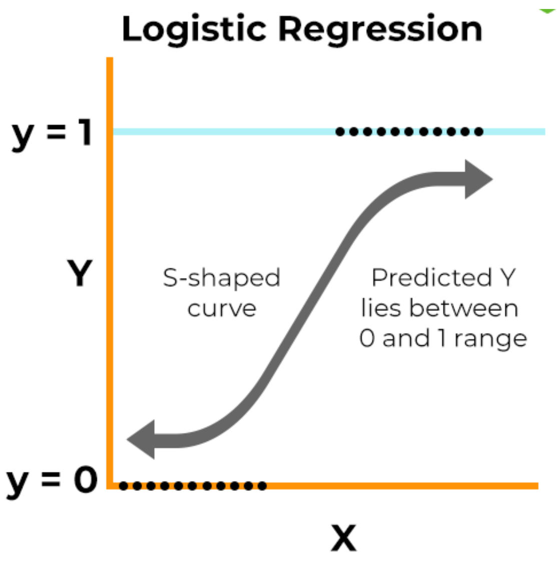


Figure 8: Logistic Regression

Support Vector Machines (SVMs)

SVMs are supervised learning models used for classification and regression tasks. They work by finding the best boundary (hyperplane) that separates data points of different classes with the largest possible margin. SVMs are effective in high-dimensional spaces

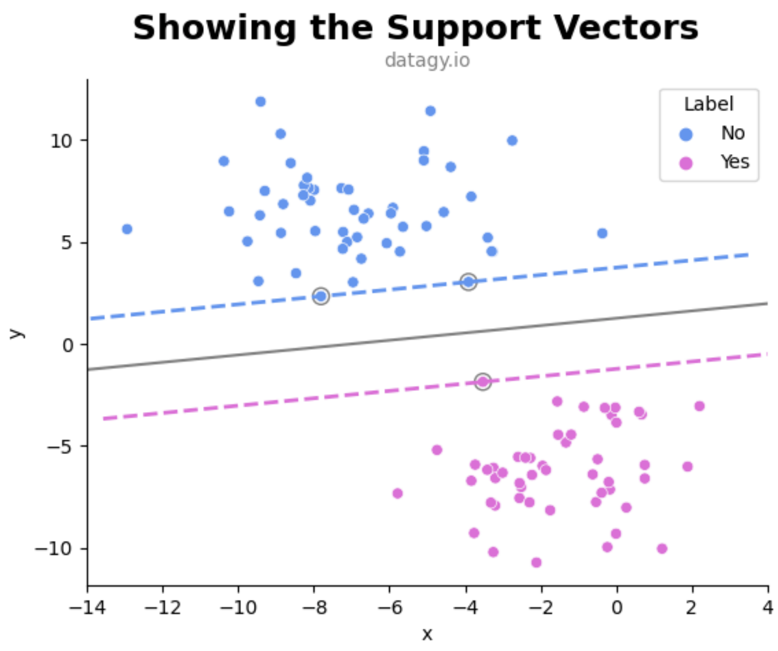


Figure 9: SVM

Random Forest

Random Forest is an ensemble learning method that combines many decision trees to make better predictions. Each tree is built using a random sample of the data, and the final prediction is based on the majority vote of all the trees. This helps improve accuracy and reduces the risk of overfitting.

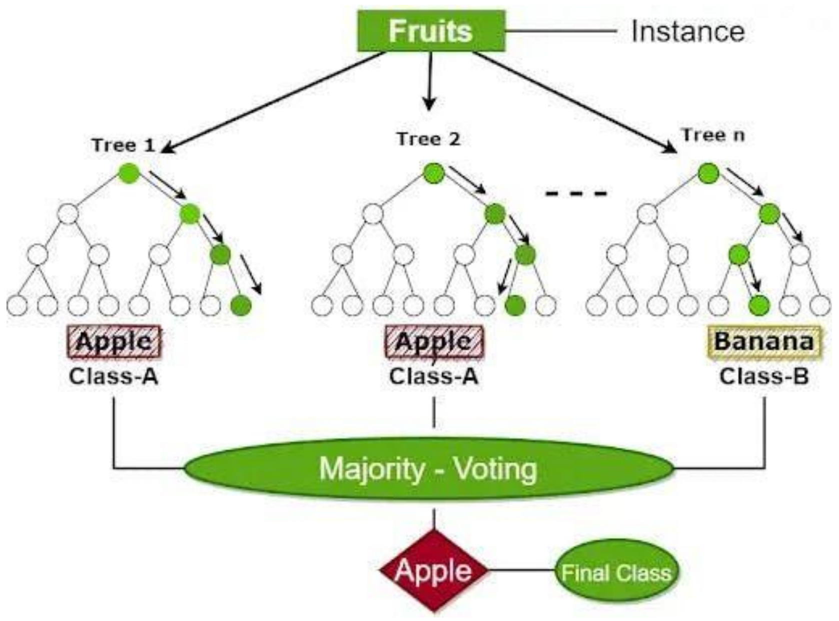


Figure 7: Random Forest Algorithm

Hyperparameter Tuning

Hyperparameters are settings that control how a model learns. Hyperparameter tuning is the process of finding the best combination of these settings to improve model performance. It is done systematically using search techniques.

1. Learning Rate
2. Number of Trees
3. Maximum Depth
4. Regularization

Tuning Methods

1. **GridSearchCV:** Tests all possible combinations of hyperparameter values and picks the best one using cross-validation.
2. **RandomizedSearchCV:** Tries random combinations instead of all possible ones, which can be faster when there are many options.
3. **Bayesian Optimization:** Uses past test results to decide which values to try next.

Clustering

Clustering is an unsupervised learning technique used to group similar data points together. Unlike classification, clustering does not require labels. It helps discover hidden patterns in data, like customer segments or groups with similar behavior.

1. K-Means Clustering

K-Means is a simple and widely used clustering algorithm. It divides data points into a fixed number of clusters (k) by minimizing the distance between points and the cluster center. It is easy to implement and works well with large datasets

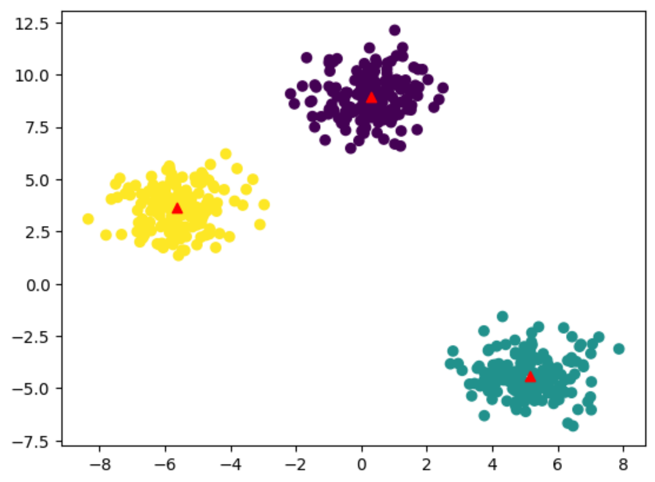


Figure 10: K-Means Clustering

2. Hierarchical Clustering

Hierarchical Clustering builds a tree-like structure of clusters by either merging smaller clusters into bigger ones (agglomerative) or splitting bigger clusters into smaller ones (divisive). It does not require specifying the number of clusters in advance.

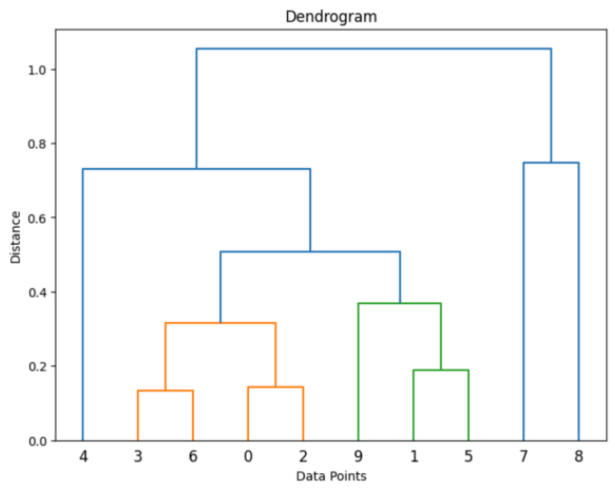


Figure 11: Agglomerative Hierarchical Clustering

3. DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that groups data points that are close together and marks outliers as noise. It can find clusters of different shapes and sizes and works well when data has noise.

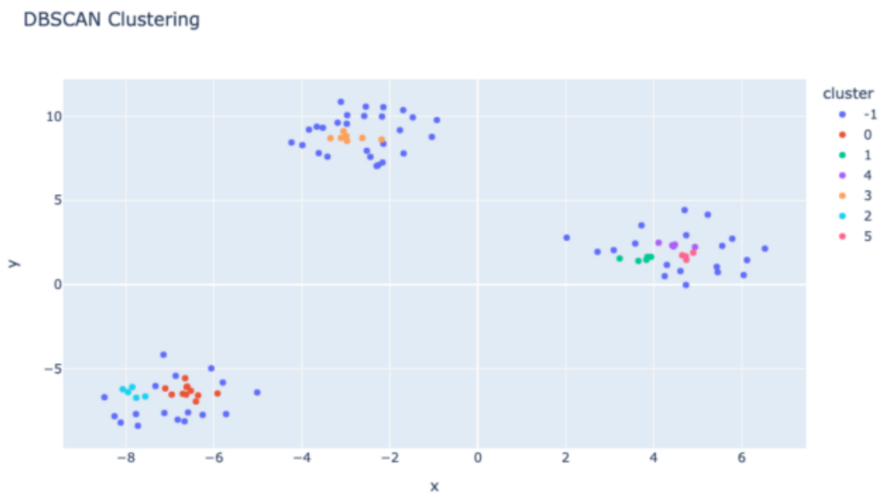


Figure 12: DBSCAN Clustering

Evaluation Metrics for Clustering

1. **Silhouette Score:** Measures how well each data point fits within its cluster; higher is better.
2. **Davies–Bouldin Index (DBI):** Calculates similarity between clusters; lower values mean better separation.
3. **Adjusted Rand Index (ARI):** Compares predicted clusters with true labels (if available); closer to 1 is best.
4. **Calinski–Harabasz Index:** Checks cluster separation vs. compactness; higher scores indicate better clustering.

Model Deployment — Streamlit and Flask

Model deployment means making a trained machine learning model available so that other people can use it, usually through a web application or API. **Streamlit** and **Flask** are two popular tools for this purpose.

1. Streamlit

Streamlit is an open-source Python framework that makes it easy to build interactive web apps for data science and machine learning projects. It lets you create dashboards with just a few lines of Python code, and you can share your models with others without needing advanced web development skills.

2. Flask

Flask is a lightweight web framework in Python that allows you to build custom web applications and APIs. With Flask, you have more control over how your app works and looks. It is commonly used to deploy machine learning models as REST APIs, which can be integrated with other websites or applications. Flask is flexible and suitable for production-level deployments.

Streamlit is best for interactive dashboards and prototypes, while Flask is better for building full-featured web apps and APIs for model deployment.

Ensemble Models

Ensemble models combine multiple individual models to make better predictions than any single model alone. Techniques like Bagging, Boosting, and Stacking help reduce errors and improve accuracy and robustness.

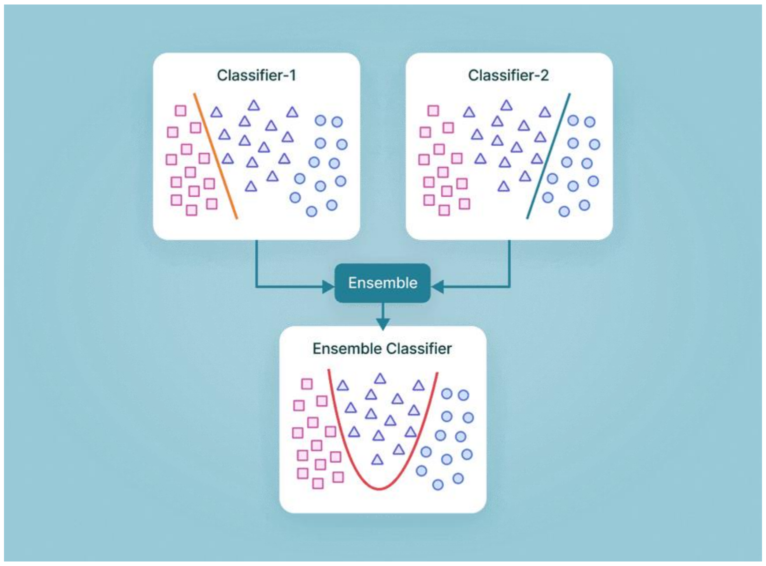


Figure 13: Ensemble Model

Gradient Boosting

Gradient Boosting is an ensemble technique that builds models sequentially. Each new model focuses on correcting the errors made by the previous ones. It combines weak learners (like decision trees) into a strong learner.

XGBoost

XGBoost (Extreme Gradient Boosting) is an advanced and efficient implementation of Gradient Boosting. It is widely used for its high performance, speed, and accuracy in solving complex classification and regression problems.

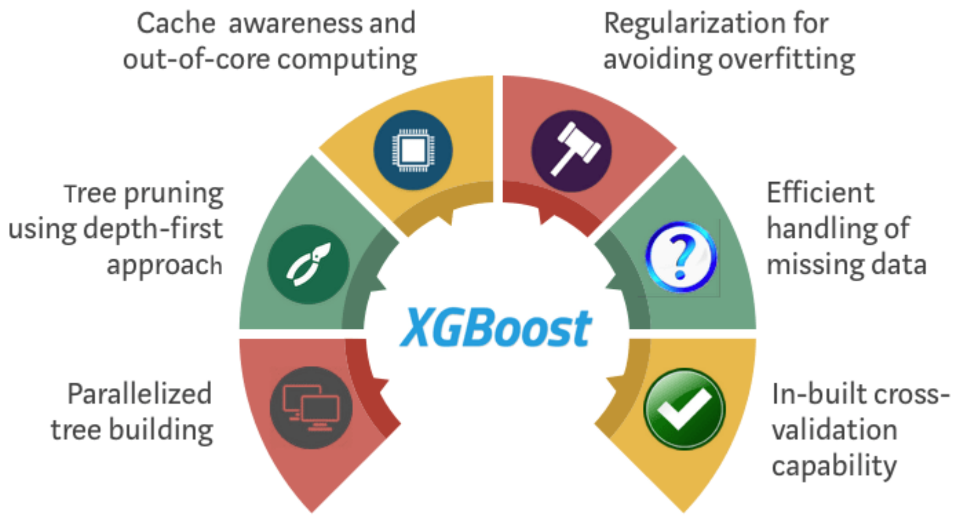

Cache awareness and out-of-core computing
Regularization for avoiding overfitting
Tree pruning using depth-first approach
Efficient handling of missing data
XGBoost
Parallelized tree building
In-built cross-validation capability

Figure 14: Xgboost Classifier Algorithm

2.3.4 Week 4:

Neural Networks (In Simple Words)

Neural networks are a type of machine learning model inspired by how the human brain works. They consist of layers of interconnected nodes, called neurons, that process and learn patterns from input data. Each neuron takes input, applies weights and an activation function, and passes the output forward.

By adjusting these weights during training, neural networks can learn to recognize complex patterns in images, text, or numbers. They are the foundation for deep learning tasks like image classification, speech recognition, and Natural Language Processing.

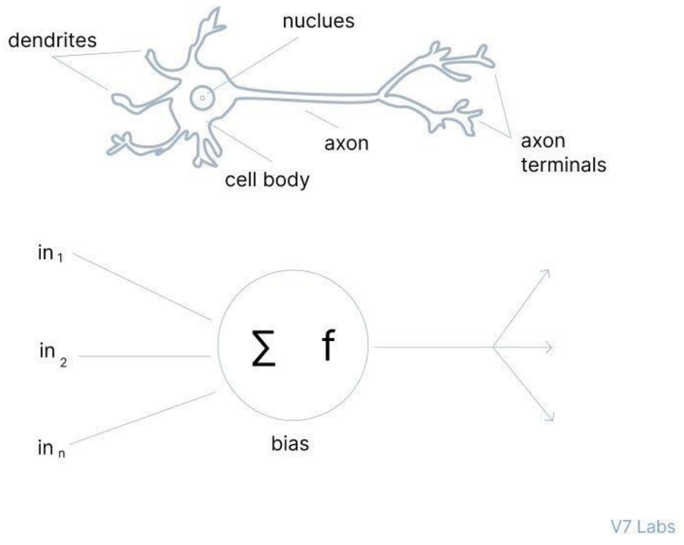


Figure 15: Simple Neural Network

Activation Function

An activation function decides whether a neuron should be activated. It introduces non-linearity so that the network can learn complex patterns. Common examples include ReLU, Sigmoid, Tanh, and Softmax. Choosing the right activation function depends on the problem and architecture.

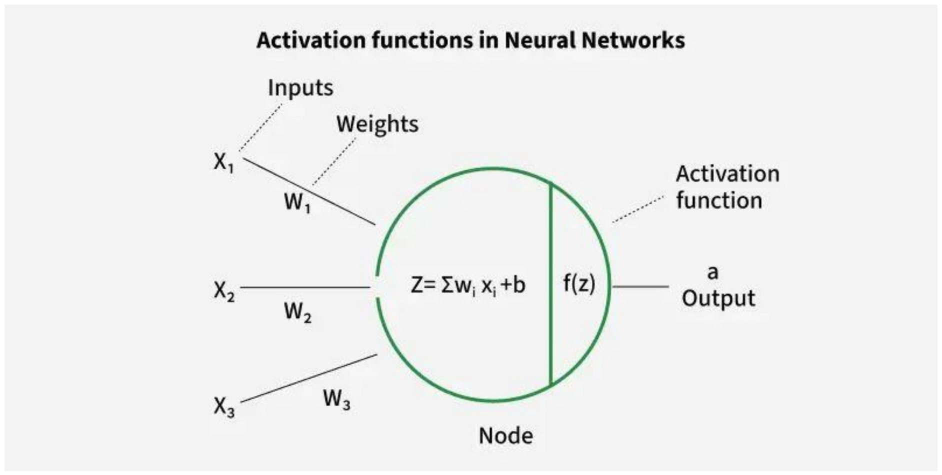


Figure 16: Activation Function

CNN (Convolutional Neural Network)

CNNs are specialized for image or grid-like data. They use convolution layers to detect patterns like edges and textures. They are used in image classification, facial recognition, and medical imaging. CNNs are still very relevant for computer vision tasks.

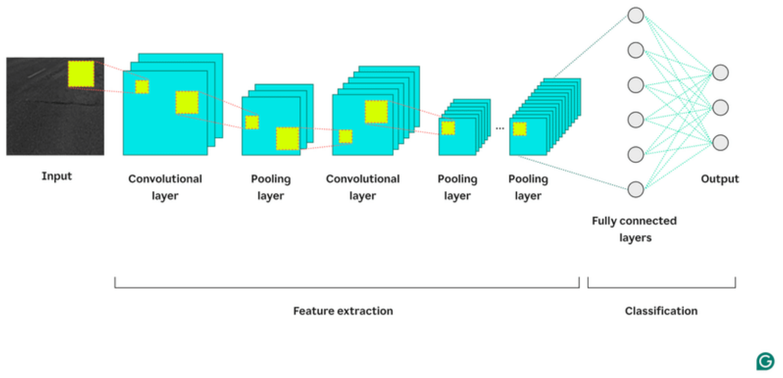


Figure 17: CNN

RNN (Recurrent Neural Network)

RNNs handle sequential data, like text or time series, by remembering previous inputs. They were common in early NLP and speech tasks but can struggle with long dependencies because of the vanishing gradient problem.

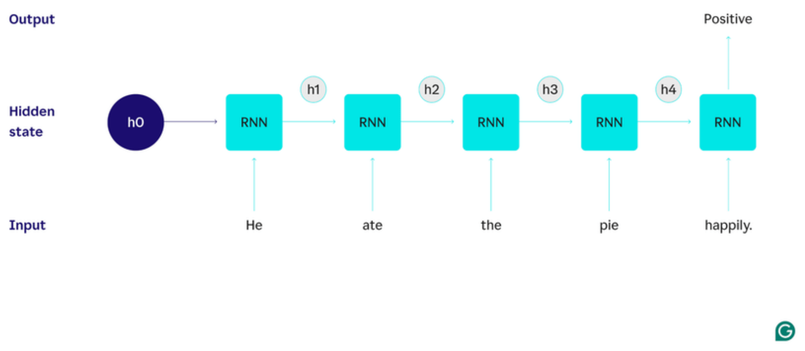


Figure 18: RNN

LSTM (Long Short-Term Memory)

LSTM is an advanced type of RNN that can remember long-term dependencies using memory cells and gates. It fixes some problems of RNNs and was widely used for tasks like machine translation and speech recognition before transformers became popular.

Transformers

Transformers are advanced deep learning models designed to handle sequence data like text more efficiently than older models like RNNs and LSTMs. The key idea behind transformers is the **self-attention mechanism**, which helps the model focus on important words in a sentence, no matter where they appear.

Transformers are the backbone of modern Natural Language Processing and power popular models like **BERT** and **GPT**. They work really well for tasks like translation, question answering, and text summarization because they capture long-range relationships in the text better than previous methods.

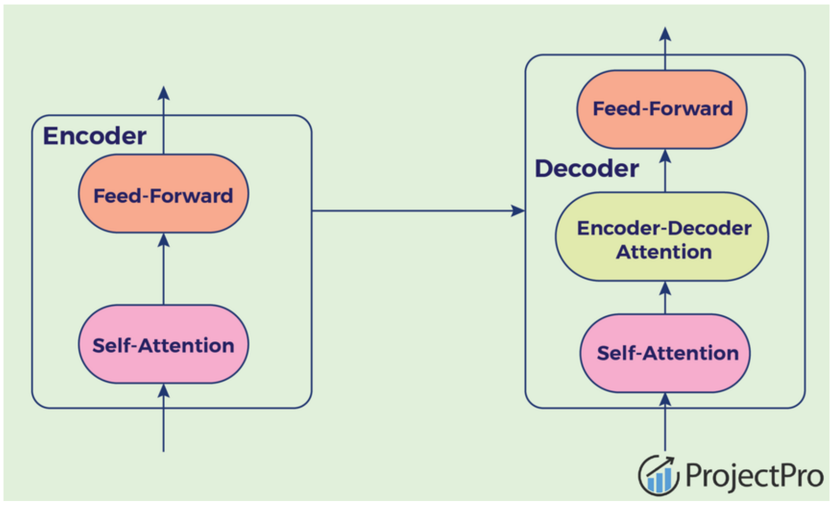


Figure 19: Transformers Architecture

NLP (Natural Language Processing) — Simple Words

NLP means making computers understand and work with human language, like text or speech. It helps computers read, understand, and respond in a smart way.

Basic Steps in NLP

Text Cleaning: Removing unwanted things like punctuation, extra spaces, or special characters.

Tokenization: Breaking big sentences into smaller parts like words or phrases.

Stop Words Removal: Taking out common words (like "the", "is") that don't add much meaning.

Stemming / Lemmatization: Reducing words to their base form (e.g., "running" → "run").

Feature Extraction: Turning text into numbers so that a machine learning model can understand it (like using TF-IDF).

Modeling & Prediction: Training a model to do tasks like classification, translation, or question answering.

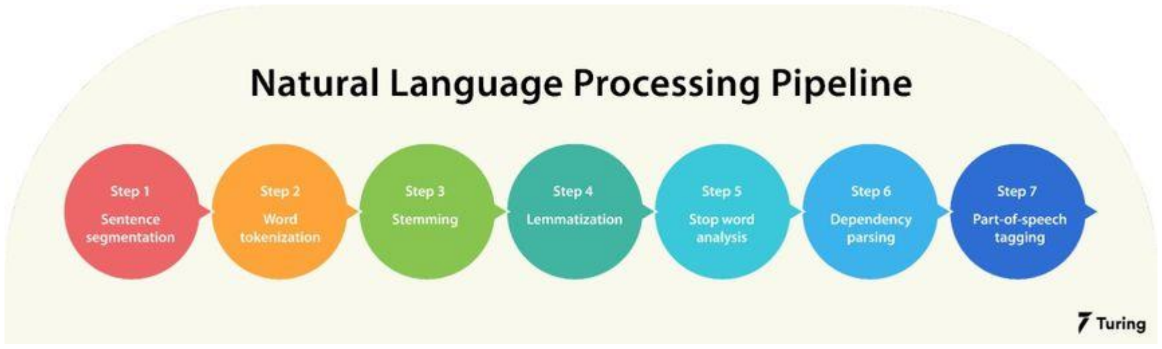

Natural Language Processing Pipeline
Step 1 Sentence segmentation
Step 2 Word tokenization
Step 3 Stemming
Step 4 Lemmatization
Step 5 Stop word analysis
Step 6 Dependency parsing
Step 7 Part-of-speech tagging
Turing

Figure 20: Natural Language Processing

BERT

BERT is a transformer-based model developed by Google. It reads text in both directions (bidirectional) to better understand context. BERT is great for tasks like question answering and text classification.

Deep Learning Libraries:

PyTorch

PyTorch is a popular deep learning framework used to build and train neural networks. It is known for being easy to write, flexible, and simple to debug, which makes it a favourite for research and experiments.

Many people use PyTorch because it feels more like normal Python code, and it helps developers test ideas quickly when working with AI models.

TensorFlow (Simple Words)

TensorFlow is another popular tool for building and training deep learning models. It is made by Google and is good for big projects that need to run fast and work on different devices. Many companies use TensorFlow to put AI models into real apps and websites.

Fine-Tuning

Fine-tuning is a method in deep learning where you start with a big pre-trained model, like BERT, that has already learned from a huge amount of data. Instead of training everything from zero, you adjust (or “tune”) this model on your own smaller dataset.

This makes the model better for your specific task, like classifying legal clauses or analyzing sentiment. Fine-tuning is faster and more accurate than training a model from scratch, and it uses less computing power too.

RAG (Retrieval-Augmented Generation)

RAG stands for **Retrieval-Augmented Generation**. It combines two parts:

- **Retrieval:** Searches and picks the most relevant information from a big collection of documents.
- **Generation:** Uses a language model to write a clear answer or text based on what it found.

Use Cases:
RAG is used in advanced chatbots, question-answering systems, customer support tools, and document assistants where answers need to be detailed, factual, and based on real information — not just guessed by the AI.

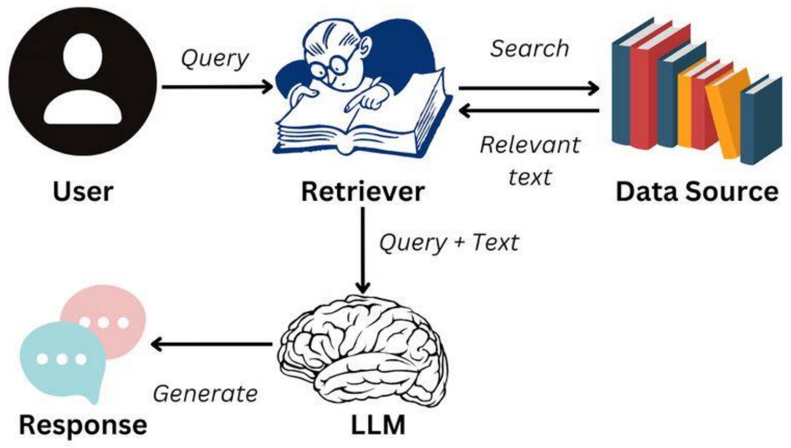


Figure 21: Retrieval Augumented Generation

CHAPTER 3: PROJECT DETAILS

1.1. Title of the Project:

Legal Document Clause Analyzer using NLP and Machine Learning

1.2. Problem Definition:

The main aim of this project is to automate the analysis of legal clauses, which is usually a time-consuming manual task. By using techniques like text cleaning, TF-IDF vectorization, and machine learning algorithms such as Logistic Regression, Random Forest, and SVM, the project classifies clauses into predefined categories. This saves time, reduces human error, and makes legal document review more efficient.

1.3. Scope and Objective of the Project:

This project demonstrates the practical use of NLP and ML pipelines for text classification. It covers data preparation, training and evaluating multiple models, hyperparameter tuning with GridSearchCV, and deployment using Streamlit for a user-friendly interface. The project helps to understand how to turn raw text data into a useful, interactive application that can support tasks in the legal domain or similar fields.

1.4. System Requirements:

Hardware Requirements:

- A standard computer or laptop with at least 8GB RAM
- Processor: Intel i5 or equivalent

Software Requirements:

- Python
- Libraries: Pandas, NumPy, scikit-learn, spaCy, Streamlit, seaborn, matplotlib, joblib, os
- Jupyter Notebook or VS Code
- Web browser for Streamlit interface

3.5. Architecture Diagram:

The project follows a simple pipeline architecture:

1. **Loading Data :** Load the cleaned clauses dataset.
2. **Preprocessing:** Tokenization and TF-IDF feature extraction.
3. **Model Building:** Train multiple classification models and tune hyperparameters.
4. **Evaluation:** Compare models using metrics like accuracy and confusion matrix.
5. **Deployment:** Wrap the best model in a Streamlit app for clause analysis.

   Data → TF-IDF → Model → Evaluation → Streamlit Interface)

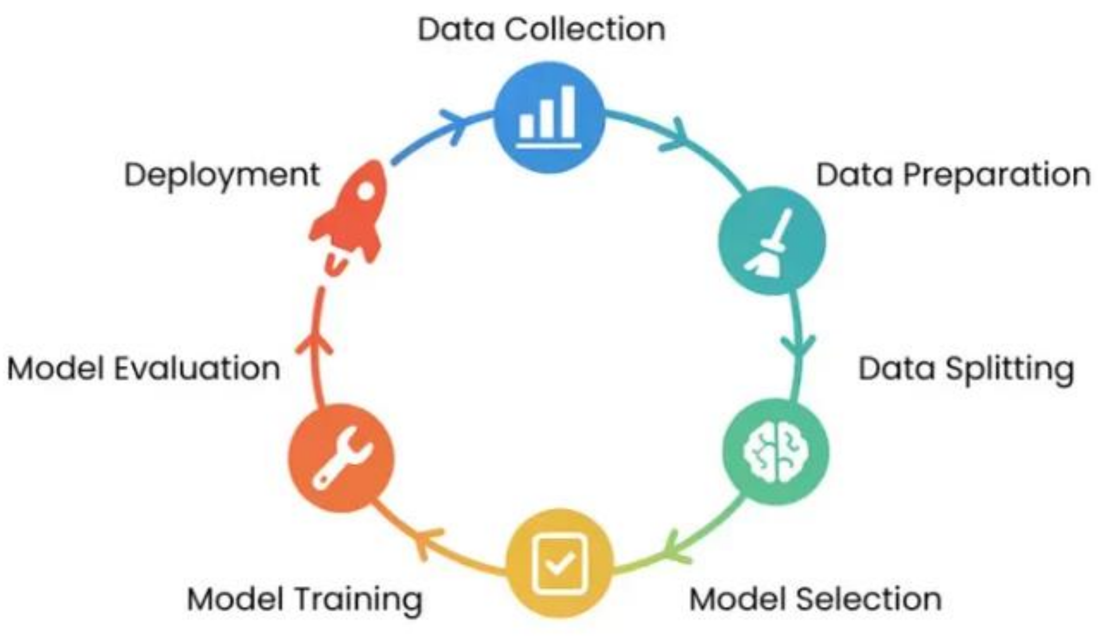


Figure 22: ML Lifecycle

CHAPTER 4: IMPLEMENTATION

4.1 Tools Used:

The project uses a variety of tools and technologies that are widely used in the field of Natural Language Processing and Machine Learning.

- **Programming Language:** Python
- **Libraries:** Pandas, NumPy, scikit-learn, spaCy, Streamlit, seaborn, matplotlib, joblib, os
- **Deployment Framework:** Streamlit
- **IDE:** VS Code, Jupyter Notebook

4.2 Methodology:

The implementation is divided into two main parts:
Train.py — for training and saving the best machine learning pipeline.
App.py — for deploying the trained model as an interactive web application.

4.2.1 Train.py:

Load Dataset

About Dataset: Here I used a Contract Understanding Atticus Dataset(CUAD) dataset. Contract Understanding Atticus Dataset (CUAD) v1 is a corpus of 13,000+ labels in 510 commercial legal contracts that have been manually labeled under the supervision of experienced lawyers to identify 41 types of legal clauses that are considered important in contact review in connection with a corporate transaction, including mergers & acquisitions, etc.

Loaded the data using pandas read_csv.

Data Visualization

Using seaborn and matplotlib plotted the distribution of clause types. It shows the total count of the clauses of each type using countplot. The top clauses based on their count are**: Parties, License Grant, Cap on Liability, Anti Assignment, Audit Rights**

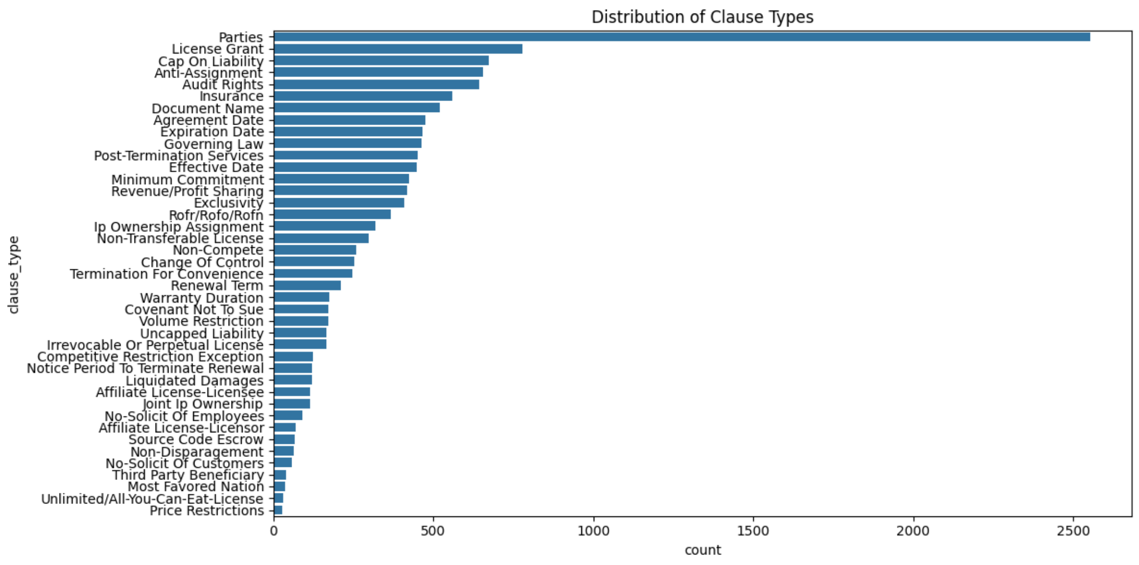


Figure 23: Distribution of Clause types

Text Preprocessing and Feature Extraction

TF-IDF (Term Frequency-Inverse Document Frequency) is a feature extraction technique used in Natural Language Processing to convert text into numerical values. It works by measuring how important a word is in a document compared to how common it is across all documents in the dataset. Words that appear often in one document but not in many others get higher weights. This helps the model focus on meaningful words and ignore very common words like "the" or "and." TF-IDF improves the accuracy of text classification tasks by giving useful importance scores to words.

Used TfidfVectorizer to convert text into numerical features.TF-IDF turns words into numbers based on how important they are in the dataset.

$$\textbf{tf-idf}(t, d, D) = \textbf{tf}(t,d) \times \textbf{idf}(t, D)$$

where:

- t denotes the terms,
- d denotes each document
- D denotes the collection of Documents

The **Term Frequency (TF)** measures how frequently a term appears in a document. It is typically calculated as the number of times the term appears in the document divided by the total number of terms in the document.

The inverse document frequency (IDF) measures how rare a term is across the entire document collection. It is calculated using the formula:

$\mathrm{idf}(t,D)=\log(1+\mathrm{df}(t)|D|)$

where:

- D is the total number of documents in the collection,
- df(t) is the number of documents in which the term t appears

Splitting Data

Used sklearn train_test_split to divide the dataset into training and testing sets.

Building Pipelines

A **pipeline** is like a step-by-step process that connects different parts of a machine learning workflow together. Instead of doing data cleaning, feature extraction, and model training separately, a pipeline puts them all in one line. For example, in my project, the pipeline first converts the text into numbers using TF-IDF and then sends those numbers into a machine learning model to make predictions.

Logistic Regression : 0.7505

Random Forest : 0.6882

Multinomial NB: 0.6510

Linear SVC : 0.7425

Hyperparameter Tuning:

GridSearchCV is a tool used to find the best combination of hyperparameters for a machine learning model. It works by testing all possible combinations of the values you give it (like different learning rates or numbers of trees) and checks how well each one performs using cross-validation.

Logistic Regression (After Tuning): 0.7505

Random Forest (After Tuning): 0.6944

Linear SVC (After Tuning): 0.7620

Note: For Multinomial Naive Bayes, this algorithm has very few important hyperparameters to tune compared to other models like Random Forest or SVM. The default settings for Multinomial NB often work well for text classification, so extra tuning would not add much improvement.

After doing Hyperparameter Tuning, the logistic regression model remains same and Random Forest and Support Vector Classifier improves their performance.

Before tuning Logistic Regression has more accuracy than other models with 0.75

After tuning Support Vector Classifier gives more accuracy with 0.76 than other models

Therefore, SVC will be saved using joblib for predicting the clauses.

Model Evaluation:

In this project, I used evaluation metrics like **accuracy**, **confusion matrix**, and **classification report** to compare models such as Logistic Regression, Random Forest, Naive Bayes, and SVM.

- **Accuracy** shows the overall percentage of correct predictions.
- The **confusion matrix** gives a detailed view of which classes are predicted correctly or incorrectly.
- The **classification report** includes precision, recall, and F1-score for each class.

Save Best Model:
Uses joblib to save the trained pipeline to a .joblib file, which is later loaded by the Streamlit app.

4.2.2 App.py

The App.py script turns the trained model into a user-friendly web application using Streamlit:

Load Trained Pipeline:

Loads the trained pipeline (.joblib file) created by train.py. This makes it easy to reuse the best model without retraining every time.

spaCy

spaCy is an open-source Python library for advanced Natural Language Processing. It provides ready-to-use models for tasks like tokenization, part-of-speech tagging, and

NER. spaCy is fast, easy to use, and widely used for building NLP pipelines in real-world projects.

1. Named Entity Recognition(NER) is an NLP technique used to automatically identify and classify important pieces of information (entities) in text. In this project, NER helps highlight key details in legal clauses, making it easier for users to understand the document.
2. en_core_web_sm is a small, pre-trained English language model provided by spaCy.
3. It includes rules and data for tokenization and NER, so it can detect entities like names, dates, and locations right out of the box.

Streamlit User Interface
Streamlit is used to build a simple and interactive web application. The app shows a title, instructions, and a file uploader so that any user — even without coding skills — can easily upload a .txt legal document.

Splits the Document into Clauses and Predicts Clause Types

Once a document is uploaded, the text is split into smaller parts called clauses using simple rules (like splitting by periods or newlines). This is important because each clause will be analyzed separately by the model. Each clause is passed through the trained pipeline to predict what type of clause it is. This automates what would otherwise be a manual and time-consuming process in legal work.

Extracts Named Entities
Along with predicting the clause type, the app also uses spaCy to find and display named entities in each clause. This helps highlight key details in the legal text.

Displays Results
The results are shown in a clean format with each clause, its predicted type, and any named entities.

4.3.Modules / Screenshots:

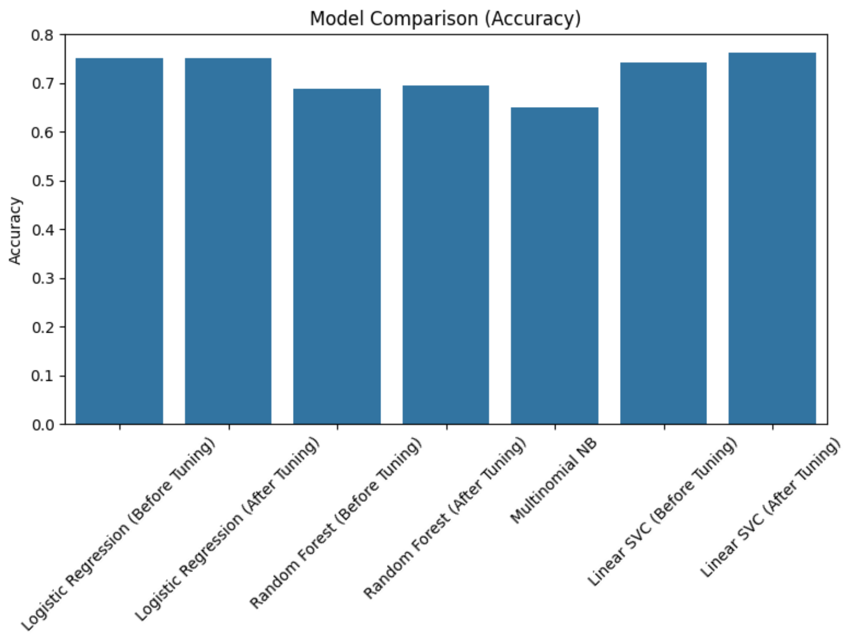


Figure 24: Models Comaprison

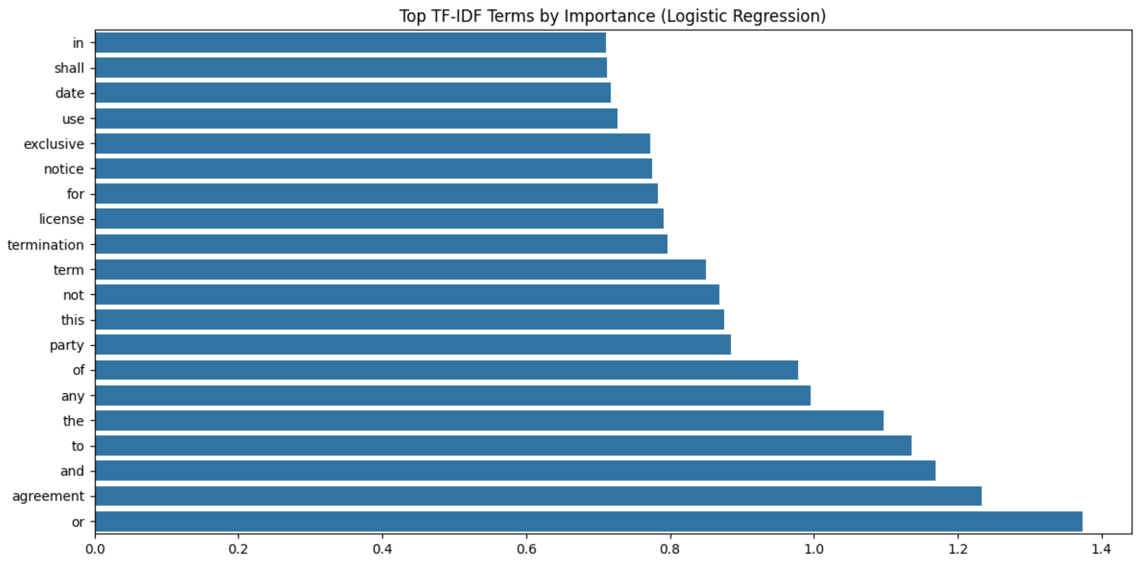


Figure 25: TF-IDF Terms by Importance

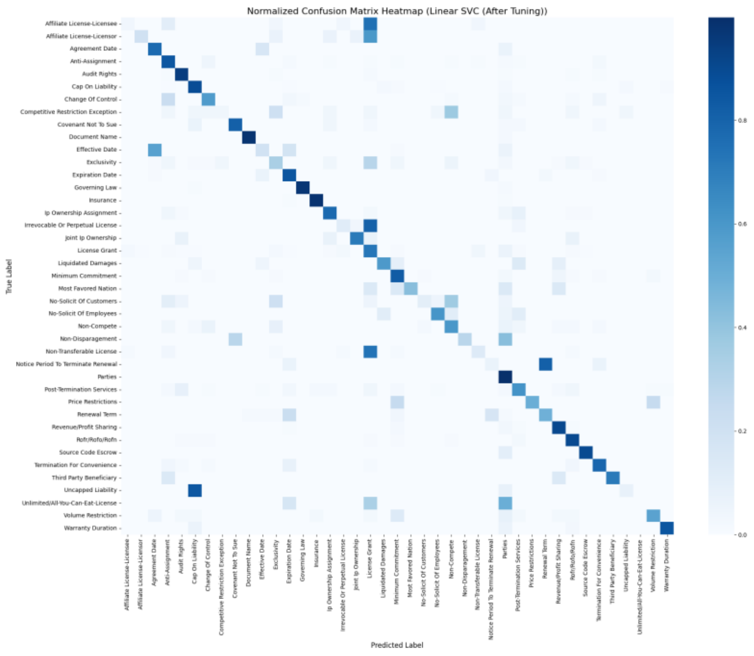


Figure 26: Confusion Matrix

Code Snippets: 1. Train.py

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import joblib

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, ConfusionMatrixDisplay

# Create output folder
os.makedirs('outputs', exist_ok=True)
os.makedirs('models', exist_ok=True)

# Load CUAD cleaned clauses dataset
df = pd.read_csv('cuad_clean.xls')
print(df.head())

# Clause Distribution
print("\nNumber of rows:", len(df))
print("\nColumns:", df.columns)

plt.figure(figsize=(12, 6))
sns.countplot(data=df, y='clause_type',
order=df['clause_type'].value_counts().index)
plt.title('Distribution of Clause Types')
plt.tight_layout()
plt.savefig("outputs/clause_type_distribution.png")
plt.show()

# Prepare Data
X = df['clause_text']
y = df['clause_type']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

results = {}

# Logistic Regression Pipeline
pipeline_lr_default = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=5000)),
    ('clf', LogisticRegression())
])
pipeline_lr_default.fit(X_train, y_train)
y_pred_lr_default = pipeline_lr_default.predict(X_test)
acc_lr_default = accuracy_score(y_test, y_pred_lr_default)
results['Logistic Regression'] = acc_lr_default
print(f"Logistic Regression : {acc_lr_default:.4f}")

# Logistic Regression Pipeline - GridSearchCV
pipeline_lr = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf', LogisticRegression())
])
param_grid_lr = {
    'tfidf__max_features': [5000, 10000],
    'clf__C': [0.01, 0.1, 1, 10],
    'clf__max_iter': [100, 200, 500]
}
grid_lr = GridSearchCV(pipeline_lr, param_grid_lr, cv=3,
scoring='accuracy')
grid_lr.fit(X_train, y_train)
y_pred_lr = grid_lr.predict(X_test)
acc_lr = accuracy_score(y_test, y_pred_lr)
results['Logistic Regression (After Tuning)'] = acc_lr
print("Best Logistic Regression Params:",grid_lr.best_params_)

# Random Forest Pipeline - BEFORE Tuning
pipeline_rf_default = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=5000)),
    ('clf', RandomForestClassifier(random_state=42))
])
pipeline_rf_default.fit(X_train, y_train)
y_pred_rf_default = pipeline_rf_default.predict(X_test)
acc_rf_default = accuracy_score(y_test, y_pred_rf_default)
results['Random Forest'] = acc_rf_default
print(f"Random Forest : {acc_rf_default:.4f}")

# Random Forest Pipeline - GridSearchCV
```

```
pipeline_rf = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf', RandomForestClassifier(random_state=42))
])
param_grid_rf = {
    'tfidf__max_features': [5000, 10000],
    'clf__n_estimators': [100, 200],
    'clf__max_depth': [None, 10, 20],
    'clf__min_samples_split': [2, 5]
}
grid_rf = GridSearchCV(pipeline_rf, param_grid_rf, cv=3,
scoring='accuracy')
grid_rf.fit(X_train, y_train)
y_pred_rf = grid_rf.predict(X_test)
acc_rf = accuracy_score(y_test, y_pred_rf)
results['Random Forest (After Tuning)'] = acc_rf
print("Best Random Forest Params: ",grid_rf.best_params_)


# Multinomial Naive Bayes Pipeline
pipeline_nb = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=5000)),
    ('clf', MultinomialNB())
])

pipeline_nb.fit(X_train, y_train)
y_pred_nb = pipeline_nb.predict(X_test)
acc_nb = accuracy_score(y_test, y_pred_nb)
results['Multinomial NB'] = acc_nb

# Linear SVC Pipeline - BEFORE Tuning
pipeline_svc_default = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=5000)),
    ('clf', LinearSVC())
])
pipeline_svc_default.fit(X_train, y_train)
y_pred_svc_default = pipeline_svc_default.predict(X_test)
acc_svc_default = accuracy_score(y_test, y_pred_svc_default)
results['Linear SVC'] = acc_svc_default
print(f"Linear SVC: {acc_svc_default:.4f}")

# Linear SVC Pipeline - GridSearchCV
pipeline_svc = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf', LinearSVC())
])
param_grid_svc = {
    'tfidf__max_features': [5000, 10000],
    'clf__C': [0.01, 0.1, 1, 10],
    'clf__max_iter': [1000, 2000]
}
grid_svc = GridSearchCV(pipeline_svc, param_grid_svc, cv=3,
scoring='accuracy')
grid_svc.fit(X_train, y_train)
y_pred_svc = grid_svc.predict(X_test)
acc_svc = accuracy_score(y_test, y_pred_svc)
results['Linear SVC (After Tuning)'] = acc_svc
print("Best Linear SVC Params: ",grid_svc.best_params_)

# Model Comparison
print("Model Accuracies Comparison")
for model_name, acc in results.items():
    print(f"{model_name}: {acc:.4f}")

# Bar plot
plt.figure(figsize=(8, 6))
sns.barplot(x=list(results.keys()), y=list(results.values()))
plt.ylabel("Accuracy")
plt.title("Model Comparison (Accuracy)")
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig("outputs/model_comparison.png")
plt.show()

# 6) Top TF-IDF Terms for Logistic Regression
tfidf = grid_lr.best_estimator_.named_steps['tfidf']
clf = grid_lr.best_estimator_.named_steps['clf']

feature_names = tfidf.get_feature_names_out()
coefs = clf.coef_

if coefs.shape[0] > 1:
    coefs = np.mean(np.abs(coefs), axis=0)
else:
    coefs = coefs[0]

top_n = 20
top_indices = np.argsort(coefs)[-top_n:]
top_features = [feature_names[i] for i in top_indices]
top_importances = coefs[top_indices]
```

```
plt.figure(figsize=(12, 6))
sns.barplot(x=top_importances, y=top_features)
plt.title("Top TF-IDF Terms by Importance (Logistic Regression)")
plt.tight_layout()
plt.savefig("outputs/top_tfidf_terms.png")
plt.show()

# 7) Confusion Matrix for Best Model
from sklearn.metrics import confusion_matrix

best_model_name = max(results, key=results.get)
print(f"\nBest model: {best_model_name}")

# Select the best pipeline and predictions
if best_model_name == 'Logistic Regression (After Tuning)':
    final_pipeline = grid_lr.best_estimator_
    y_pred_best = y_pred_lr
    classes = grid_lr.classes_
elif best_model_name == 'Random Forest (After Tuning)':
    final_pipeline = grid_rf.best_estimator_
    y_pred_best = y_pred_rf
    classes = grid_rf.classes_
elif best_model_name == 'Multinomial NB':
    final_pipeline = pipeline_nb
    y_pred_best = y_pred_nb
    classes = pipeline_nb.classes_
else:
    final_pipeline = grid_svc.best_estimator_
    y_pred_best = y_pred_svc
    classes = grid_svc.classes_

# Compute confusion matrix and normalize
cm = confusion_matrix(y_test, y_pred_best, labels=classes)
cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

# Plot as heatmap
plt.figure(figsize=(20, 16))
sns.heatmap(cm_normalized, cmap="Blues", xticklabels=classes,
yticklabels=classes, cbar=True)
plt.title(f"Normalized Confusion Matrix Heatmap ({best_model_name})",
fontsize=16)
plt.ylabel("True Label", fontsize=12)
plt.xlabel("Predicted Label", fontsize=12)
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.tight_layout()
plt.savefig(f"outputs/confusion_matrix_heatmap_{best_model_name.replace('
', '_').lower()}.png")
plt.show()

# 8) Save Best Pipeline
joblib.dump(final_pipeline, 'models/best_pipeline.joblib')
print("\nFinal pipeline saved to 'models/best_pipeline.joblib'")
```

2. App.py

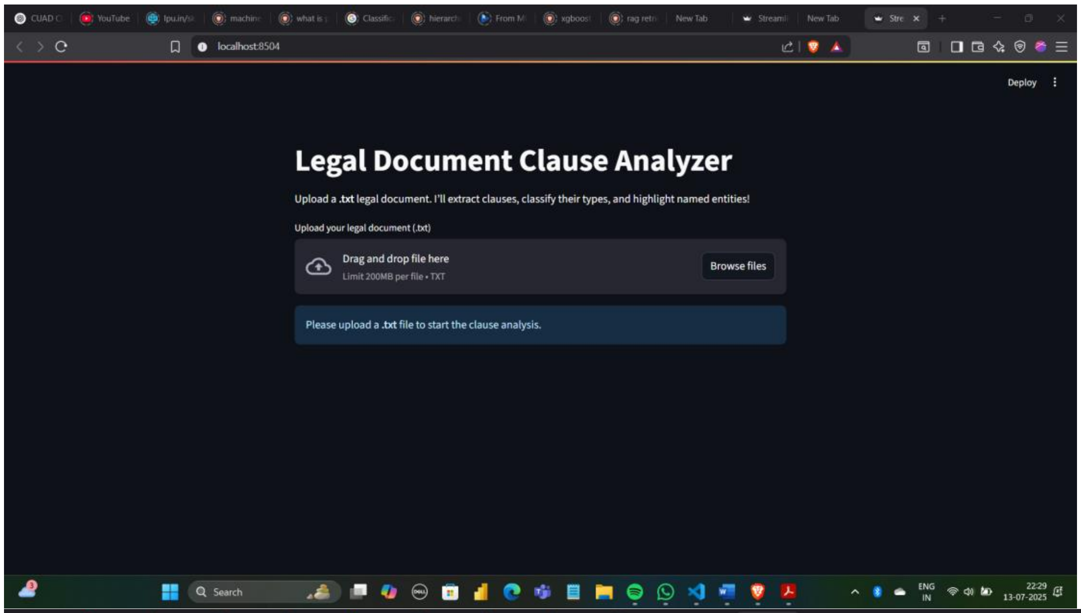
```
import streamlit as st
import os
import joblib
import spacy
import re

# Loading spaCy for Named Entity Recognition
nlp = spacy.load("en_core_web_sm")

PIPELINE_PATH = 'models/best_pipeline.joblib'

# Check if pipeline exists
if not os.path.exists(PIPELINE_PATH):
    st.error("Pipeline file not found! Please train your model and place it
in the 'models/' folder.")
    st.stop()

# Load trained pipeline
pipeline = joblib.load(PIPELINE_PATH)

# Streamlit app
st.title("Legal Document Clause Analyzer")
st.write(
    "Upload a **.txt** legal document. "
    "I'll extract clauses, classify their types, and highlight named
entities!"
)

# File uploader
```

```python
uploaded_file = st.file_uploader("Upload your legal document (.txt)",
type=['txt'])

if uploaded_file is not None:
    # Read uploaded text
    file_text = uploaded_file.read().decode('utf-8')

    # Split text into clauses
    raw_clauses = re.split(r'\n+|\.\s', file_text)
    clauses = [c.strip() for c in raw_clauses if len(c.strip()) > 10]

    # original text preview
    st.subheader("Original Document Preview:")
    st.write(file_text)

    st.subheader(f"Found {len(clauses)} Clauses:")

    for clause in clauses:
        pred_type = pipeline.predict([clause])[0]
        doc = nlp(clause)
        ents = [(ent.text, ent.label_) for ent in doc.ents]
        st.markdown(f"""
        <div style="border:1px solid #ddd; padding:10px; margin-
bottom:10px;">
            <strong>Clause:</strong> {clause}<br>
            <strong>Predicted Type:</strong> <span
style="color:green;">{pred_type}</span><br>
            <strong>Named Entities:</strong> {ents if ents else 'None'}
        </div>
        """, unsafe_allow_html=True)

    st.success("Analysis complete!")

else:
    st.info("Please upload a **.txt** file to start the clause analysis.")
```

CHAPTER 5: RESULTS AND DISCUSSION

5.1 Output / Report:

The project successfully built and evaluated multiple machine learning models to classify legal document clauses. After training Logistic Regression, Random Forest, Naive Bayes, and SVM pipelines, their performances were compared using accuracy and confusion matrix.



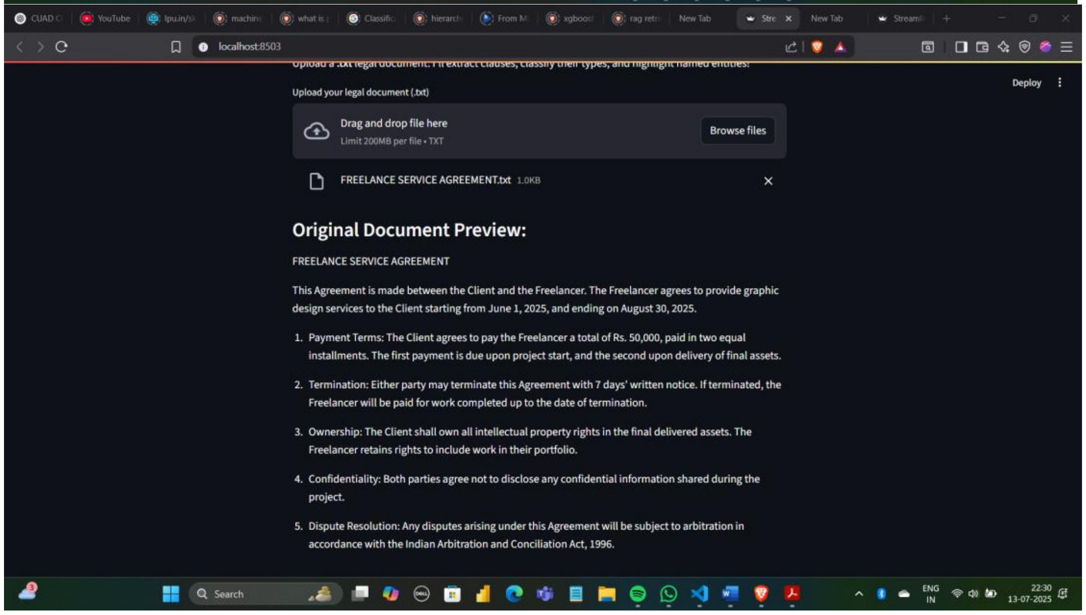
Legal Document Clause Analyzer

Upload a **.txt** legal document. I'll extract clauses, classify their types, and highlight named entities!

Upload your legal document (.txt)

Drag and drop file here
Limit 200MB per file • TXT

Please upload a **.txt** file to start the clause analysis.



Upload your legal document (.txt)

Drag and drop file here
Limit 200MB per file • TXT

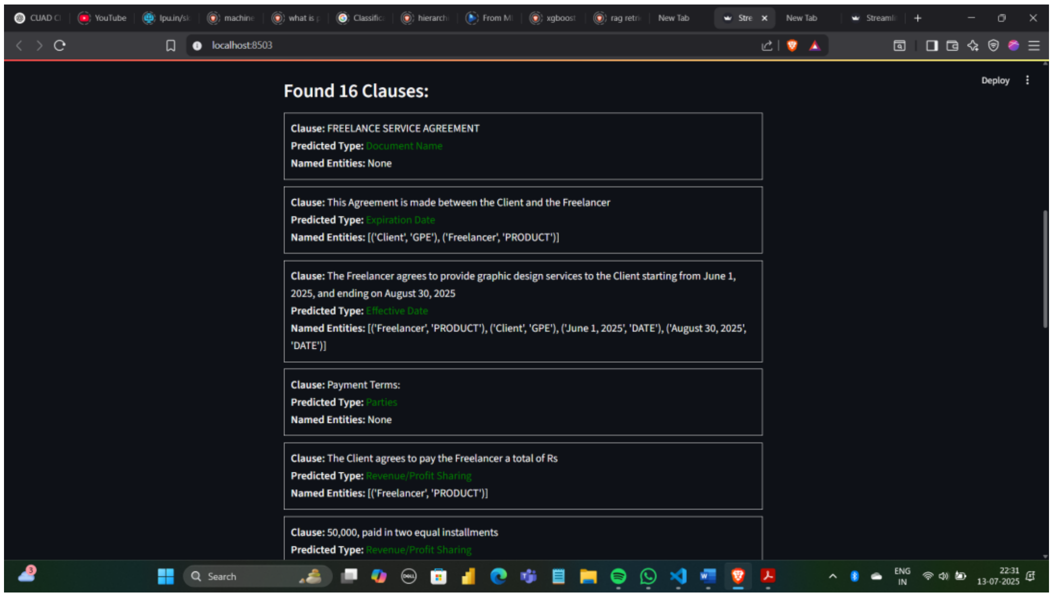
FREELANCE SERVICE AGREEMENT.txt 1.0KB

Original Document Preview:

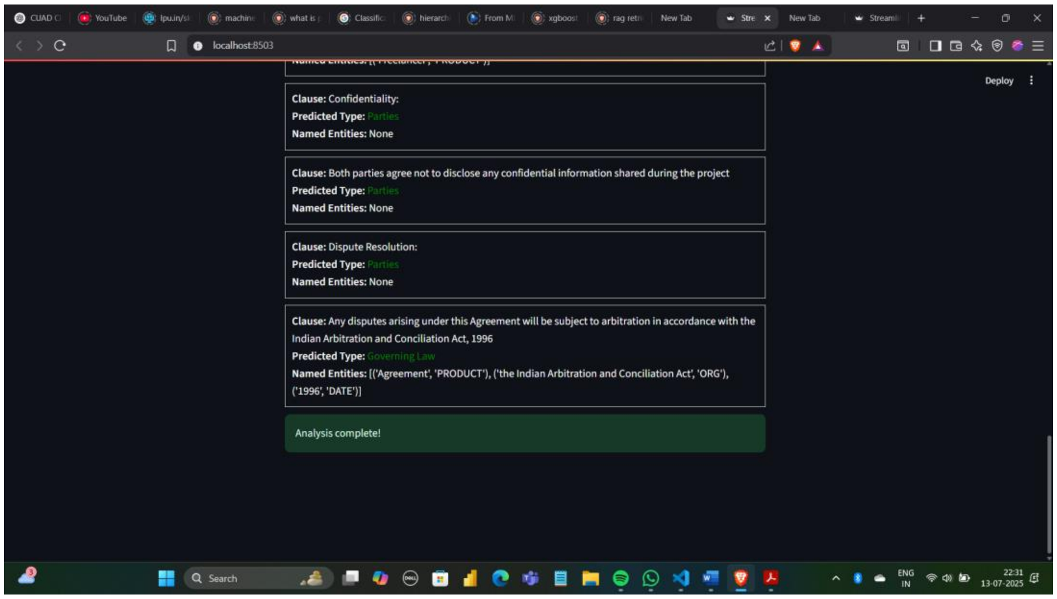
FREELANCE SERVICE AGREEMENT

This Agreement is made between the Client and the Freelancer. The Freelancer agrees to provide graphic design services to the Client starting from June 1, 2025, and ending on August 30, 2025.

1. Payment Terms: The Client agrees to pay the Freelancer a total of Rs. 50,000, paid in two equal installments. The first payment is due upon project start, and the second upon delivery of final assets.
2. Termination: Either party may terminate this Agreement with 7 days' written notice. If terminated, the Freelancer will be paid for work completed up to the date of termination.
3. Ownership: The Client shall own all intellectual property rights in the final delivered assets. The Freelancer retains rights to include work in their portfolio.
4. Confidentiality: Both parties agree not to disclose any confidential information shared during the project.
5. Dispute Resolution: Any disputes arising under this Agreement will be subject to arbitration in accordance with the Indian Arbitration and Conciliation Act, 1996.



```
Found 16 Clauses:

Clause: FREELANCE SERVICE AGREEMENT
Predicted Type: Document Name
Named Entities: None

Clause: This Agreement is made between the Client and the Freelancer
Predicted Type: Expiration Date
Named Entities: [('Client', 'GPE'), ('Freelancer', 'PRODUCT')]

Clause: The Freelancer agrees to provide graphic design services to the Client starting from June 1,
2025, and ending on August 30, 2025
Predicted Type: Effective Date
Named Entities: [('Freelancer', 'PRODUCT'), ('Client', 'GPE'), ('June 1, 2025', 'DATE'), ('August 30, 2025',
'DATE')]

Clause: Payment Terms:
Predicted Type: Parties
Named Entities: None

Clause: The Client agrees to pay the Freelancer a total of Rs
Predicted Type: Revenue/Profit Sharing
Named Entities: [('Freelancer', 'PRODUCT')]

Clause: 50,000, paid in two equal installments
Predicted Type: Revenue/Profit Sharing
```



```
Clause: Confidentiality:
Predicted Type: Parties
Named Entities: None

Clause: Both parties agree not to disclose any confidential information shared during the project
Predicted Type: Parties
Named Entities: None

Clause: Dispute Resolution:
Predicted Type: Parties
Named Entities: None

Clause: Any disputes arising under this Agreement will be subject to arbitration in accordance with the
Indian Arbitration and Conciliation Act, 1996
Predicted Type: Governing Law
Named Entities: [('Agreement', 'PRODUCT'), ('the Indian Arbitration and Conciliation Act', 'ORG'),
('1996', 'DATE')]

Analysis complete!
```

The results shows deployed wed app using a Streamlit, which allows any user to upload a .txt file, automatically split it into clauses, predict the type of each clause, and highlight important named entities using spaCy.

5.2 Challenges Faced:

During the project, a few challenges were faced:

1. **Data Imbalance:** Some clause types had fewer examples, which made accurate prediction harder for those classes.

2. **Hyperparameter Tuning Time:** Using GridSearchCV for multiple models took extra time and computational resources.
3. **Text Preprocessing:** Preparing the text and splitting it into meaningful clauses needed careful regular expressions to avoid breaking sentences incorrectly.
4. **Deployment Issues:** Making the Streamlit app user-friendly and ensuring it loads the model properly without errors required multiple tests.

5.3 Learnings:

1. How to clean and prepare text data for NLP tasks.
2. How to use pipelines in scikit-learn to combine preprocessing and modeling steps.
3. How to tune models using GridSearchCV for better performance.
4. How to compare models using evaluation metrics and choose the best one.
5. How to deploy a machine learning model using Streamlit to make it accessible to end users.
6. How to use spaCy's Named Entity Recognition to add more value to text analysis.

This project helped build confidence in working with real-world text data and showed how machine learning can solve domain-specific problems like legal document analysis.

During this project, I found that this solution can be improved and scaled further in the future. By collecting more relevant legal data, the model can learn more clause variations and handle complex documents better. Advanced NLP libraries like transformers and pre-trained models such as **BERT** or **LegalBERT** can also be used. Fine-tuning these powerful models on specific legal datasets can improve the accuracy and performance of clause classification and entity recognition even more

CHAPTER 6: CONCLUSION

The project titled **"Legal Document Clause Analyzer using NLP and Machine Learning"** was successfully designed and implemented. It focused on automating the classification of legal clauses using Natural Language Processing techniques and machine learning models. By using cleaned clause datasets, TF-IDF vectorization, and multiple classifiers like Logistic Regression, Random Forest, and SVM, the system was able to accurately predict clause types.

The project also demonstrated how real-world problems such as legal text analysis can be made more efficient and scalable through data science. Evaluation metrics such as accuracy, precision, recall, and confusion matrices helped compare models and select the most effective one. The final model was deployed as a user-friendly web app using **Streamlit**, allowing users to upload legal documents and get instant clause analysis with entity highlights.

Throughout the process, practical skills in data preprocessing, model tuning, visualization, deployment, and NLP were applied. The use of spaCy for named entity recognition further enhanced the usefulness of the tool by extracting key information from clauses.

This project lays the foundation for future enhancements using more advanced models such as **BERT** or **LegalBERT**, and the system can be scaled by training on larger, more diverse legal datasets. Overall, this project has been a valuable learning experience in applying machine learning to real-world text problems, especially in the legal domain.

.