ML LAB LEGAL DOCUMENTION
NAME: SRINATH V
SRN : PES2UG24CS824

ALL CODE#

```
# ---------------------------------------------------------------------
# Cell 1: Setup, Imports, and Constants
# ---------------------------------------------------------------------
import os
import re
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC # Support Vector Machine model
from sklearn.pipeline import Pipeline
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    f1_score,
)

# Store results for the final summary table
model_results = {}

print("Setup Complete: Required libraries imported.")
print("Proceeding with manually generated data for safe execution.")
```

This cell imports all the necessary Python libraries (like Pandas for data, Scikit-learn for models, and Matplotlib/Seaborn for graphs). It also sets the correct file paths for your uploaded data, which the program will use to find the text files.

```
# ----------------------------------------------------------------------
# Cell 2: Dummy Data Loading/Processing Functions
# ----------------------------------------------------------------------
def get_lines(filename):
    """(Dummy function - not used in this run)"""
    return []

def preprocess_text_data(filepath):
    """(Dummy function - not used in this run)"""
    return pd.DataFrame({'text': [], 'label': []})

print("Dummy data processing functions defined (Data will be manually
generated in Cell 3).")
```

These are two functions (`get_lines` and `preprocess_text_data`) that define **how** the raw text files will be read and cleaned. They convert the raw text lines (like `OBJECTIVE\tThe aim was...`) into a structured table (a Pandas

DataFrame) with two columns: `text` (the sentence) and `label` (the abstract section

CEEL 3 ;
This cell executes the functions from Cell 2 to load your large training, development, and test files. It combines the training and development data to make one large training set and then splits the text (features) from the labels (targets) for the models. It also prints the size and a sample of the data to confirm it loaded correctly.

```
# ----------------------------------------------------------------------
# Cell 3: Load Data, Manually Generate DataFrames, and Initial Preparation
# ----------------------------------------------------------------------

# --- 1. Manually Generate DataFrames (FOR SAFE EXECUTION) ---
print("--- Data Loading and Preparation (Using MANUALLY GENERATED DATA)
---")

# Training Data
data_train = {
    'text': [
        "The primary objective was to test the new compound.",
        "We randomized 10 patients into two treatment arms.",
        "The control group showed no significant change.",
        "Previous research suggests current treatments are ineffective.",
        "A large sample size was required for statistical power."
    ],
    'label': [
        "OBJECTIVE",
        "METHODS",
        "RESULTS",
        "BACKGROUND",
        "METHODS"
    ]
}
```

```
df_train_full = pd.DataFrame(data_train)

# Test Data
data_test = {
    'text': [
        "In conclusion, the results support the use of the drug.",
        "Pain scores were measured using the VAS scale.",
        "The study design was a randomized controlled trial.",
        "We analyzed the data using t-tests."
    ],
    'label': [
        "CONCLUSIONS",
        "METHODS",
        "METHODS",
        "METHODS"
    ]
}
df_test = pd.DataFrame(data_test)

print(f"Full Train data size (manual data): {len(df_train_full)}
sentences")
print(f"Test data size (manual data): {len(df_test)} sentences")

# Extract features (X) and labels (y)
X_train = df_train_full['text'].values
y_train = df_train_full['label'].values
X_test = df_test['text'].values
y_test = df_test['label'].values

print("Data successfully loaded and split into features/labels.")

# --- 2. Display the Data ---
print("\n--- Head of the Manually Generated Training Data ---")
print(df_train_full.head())
print("\n--- Head of the Manually Generated Test Data ---")
print(df_test.head())
```

CELL 4 :

This cell creates a **bar chart** to show how often each label (e.g., METHODS, RESULTS, OBJECTIVE) appears in your training data. This is important to check for **class imbalance** (where some labels appear much more often than others

```
# ------------------------------------------------------------------
# Cell 4: Data Visualization (Label Distribution Graph)
# ------------------------------------------------------------------
plt.figure(figsize=(8, 5))
# Calculate the value counts and plot as a bar chart
sns.countplot(y=df_train_full['label'],
              order=df_train_full['label'].value_counts().index,
              palette='viridis')
plt.title('Distribution of Abstract Section Labels (Manual Training
Data)')
plt.xlabel('Number of Sentences')
plt.ylabel('Abstract Section Label')
plt.tight_layout()
plt.show()

print("Label distribution graph generated for the small dataset.")
```

: Before a computer can understand text, it must be converted into numbers. We use **TF-IDF (Term Frequency-Inverse Document Frequency)**, which turns each sentence into a list of numbers (a vector). The `ngram_range=(1, 3)` setting tells it to consider single words (unigrams), two-word phrases (bigrams), and three-word phrases (trigrams) as feature

```
# ----------------------------------------------------------------------
# Cell 5: Feature Extraction Setup (TF-IDF)
# ----------------------------------------------------------------------
# Define the TF-IDF Vectorizer step
tfidf_vectorizer = TfidfVectorizer(
    ngram_range=(1, 2),    # Reduced n-gram range for the tiny dataset
    stop_words='english'
)

print("TF-IDF Vectorizer configured with N-grams (1 to 2).")



# ----------------------------------------------------------------------
# Cell 6: Model 1: Multinomial Naive Bayes (MNB) Training
# ----------------------------------------------------------------------
print("--- Training Multinomial Naive Bayes Model ---")

# Define the MNB Pipeline
mnb_model = Pipeline([
    ('tfidf', tfidf_vectorizer),
    ('clf', MultinomialNB(alpha=1.0)) # Higher smoothing (alpha=1.0) for
tiny data
])

# Train the MNB model
mnb_model.fit(X_train, y_train)

# Make predictions
y_pred_mnb = mnb_model.predict(X_test)

print("Multinomial Naive Bayes Training Complete.")
```

This cell defines and trains the **Multinomial Naive Bayes** model. It uses a **Pipeline** to link the TF-IDF conversion (from Cell 5) directly to the

classifier. Naive Bayes is a simple, fast, and effective model for text classification based on probability.

CELL 7 :

```
# ----------------------------------------------------------------------
# Cell 7: Model 2: Support Vector Machine (SVM) Training
# ----------------------------------------------------------------------
print("--- Training Support Vector Machine (LinearSVC) Model ---")

# Define the SVM Pipeline
svm_model = Pipeline([
    ('tfidf', tfidf_vectorizer),
    ('clf', LinearSVC(C=10.0, penalty='l2', dual=True, random_state=42,
max_iter=2000))
])

# Train the SVM model
svm_model.fit(X_train, y_train)

# Make predictions
y_pred_svm = svm_model.predict(X_test)

print("Support Vector Machine (LinearSVC) Training Complete.")
```

his cell defines and trains the **Support Vector Machine (SVM)** model using `LinearSVC`. SVM is generally more powerful for text classification than MNB, as it finds the "best line" (hyperplane) to separate the classes in the high-dimensional feature space.

This cell prints the **Classification Report** for both models, showing metrics like Precision, Recall, and F1-score for every label. It also creates a **Confusion Matrix** (a heatmap) for the SVM model, which visually shows where the model is making mistakes (e.g., labeling a RESULTS sentence as a METHODS sentence)

```
# ----------------------------------------------------------------
# Cell 8: Detailed Evaluation and Confusion Matrix
# ----------------------------------------------------------------
print("--- Detailed Evaluation (Results are illustrative due to small
data) ---")

# 1. MNB Report
print("\n=== Multinomial Naive Bayes Classification Report ===")
# Use zero_division=0 to safely handle labels not present in the
predictions
print(classification_report(y_test, y_pred_mnb, zero_division=0))

# 2. SVM Report
print("\n=== Support Vector Machine (LinearSVC) Classification Report
===")
print(classification_report(y_test, y_pred_svm, zero_division=0))

# 3. Confusion Matrix Visualization (for SVM)
labels_train = np.unique(y_train)
cm_svm = confusion_matrix(y_test, y_pred_svm, labels=labels_train)

plt.figure(figsize=(8, 6))
sns.heatmap(cm_svm, annot=True, fmt='d', cmap='Reds',
            xticklabels=labels_train,
            yticklabels=labels_train)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for SVM Classifier (Manual Data)')
plt.show()
```

```
print("Confusion Matrix for SVM model generated.")
```

CELL 9 AND 10 ,

```
# ----------------------------------------------------------------------
# Cell 9: Comparative Accuracy Summary Table
# ----------------------------------------------------------------------
print("--- Comparative Accuracy Summary Table (Illustrative Results) ---")

# Calculate metrics
mnb_accuracy = accuracy_score(y_test, y_pred_mnb)
svm_accuracy = accuracy_score(y_test, y_pred_svm)
mnb_f1 = f1_score(y_test, y_pred_mnb, average='weighted', zero_division=0)
svm_f1 = f1_score(y_test, y_pred_svm, average='weighted', zero_division=0)

# Store results
model_results['MNB'] = {'Accuracy': mnb_accuracy, 'Weighted F1-Score':
mnb_f1}
model_results['SVM'] = {'Accuracy': svm_accuracy, 'Weighted F1-Score':
svm_f1}

# Create the summary DataFrame
summary_df = pd.DataFrame(model_results).T

# Format the results for better readability
summary_df_styled = summary_df.style.format({
    'Accuracy': "{:.4f}",
    'Weighted F1-Score': "{:.4f}"
}).set_caption("Test Set Performance Comparison (Manual Data)")

# Print as HTML table
print(summary_df_styled.to_html())
print("\nSummary table generated.")
```

```
# ----------------------------------------------------------------------
# Cell 10: Example Prediction and Conclusion
# ----------------------------------------------------------------------
print("--- Example Prediction using the best model (SVM) ---")

new_sentences = [
    "The goal of the experiment was to find a cure.",       # OBJECTIVE
    "We used a simple linear regression model.",           # METHODS
    "Final analysis showed a huge improvement.",           # RESULTS
]

# Predict using the SVM model
predictions = svm_model.predict(new_sentences)

print("\nSentence | Predicted Label (SVM)")
print("-" * 50)
for sentence, prediction in zip(new_sentences, predictions):
    print(f"[{prediction:^11}] {sentence}")

print("\n--- Analysis Conclusion ---")
print(f"The code successfully executed the full comparative study
pipeline, demonstrating an SVM Accuracy of
{model_results['SVM']['Accuracy']:.4f} and MNB Accuracy of
{model_results['MNB']['Accuracy']:.4f} on the small manual dataset.")
print("The structure is ready for use with your large dataset by simply
modifying Cell 3 back to file reading.")
```

IMAGES :