# Natural Language Processing using Python Programming

## Notebook 10.3: Deploying NLP Models with Flask

Python 3.8+ Flask Latest scikit-learn Latest Joblib Latest License MIT

---

**Part of the comprehensive learning series:** Natural Language Processing using Python Programming

**Learning Objectives:**

- Master model persistence using joblib for production-ready ML pipeline serialization
- Build RESTful APIs with Flask framework for exposing NLP models as web services
- Implement proper error handling and JSON response formatting for robust API design
- Understand the deployment workflow from data science prototypes to production systems
- Test and validate API endpoints using HTTP requests and standard debugging practices

---

- This notebook demonstrates the **production deployment** of NLP models using Flask, bridging the gap between data science prototypes and real-world web services that can be consumed by applications.

- We'll explore **model persistence** with joblib and **API development** with Flask, covering the essential skills needed to make your NLP models accessible through professional web services.

## 1. Model Persistence: Saving the Scikit-learn Pipeline

- We need to save the entire classification pipeline (Vectorizer + Classifier) as a single file.

- We'll reuse the SVM pipeline trained in Chapter 8.2 and save it using Python's standard serialization module, `pickle` (or `joblib`).

In [1]:

```
# Import necessary libraries
# pickle is included for completeness, but joblib is preferred for model pers
# Both are shown here for educational purposes
```

```python
import pickle
import joblib
from sklearn.datasets import fetch_20newsgroups
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC

MODEL_SAVE_PATH = '../../models/svm_text_classifier.pkl'

# --- 1. Train the Model (Quick Retrain from 8.2) ---
categories = ['comp.graphics', 'rec.sport.baseball'] # Use a smaller subset f
newsgroups = fetch_20newsgroups(subset='all', categories=categories, shuffle=

X = newsgroups.data
y = newsgroups.target

svm_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf', LinearSVC(random_state=42, dual=True)),
])
svm_pipeline.fit(X, y)

# --- 2. Save the Pipeline ---
# joblib is preferred over pickle for large NumPy arrays often used in ML
joblib.dump(svm_pipeline, MODEL_SAVE_PATH)
joblib.dump(newsgroups.target_names, '../../models/target_names.pkl')

print(f"Trained pipeline saved successfully to {MODEL_SAVE_PATH}")

# Verification: Load the model back
loaded_model = joblib.load(MODEL_SAVE_PATH)
print("Model loaded successfully for verification.")
```

```
Trained pipeline saved successfully to ../../models/svm_text_classifier.pkl
Model loaded successfully for verification.
```

---

## 2. Setting up the Flask API Structure

- A Flask application typically lives in its own directory ( `/app` in our case) and defines routes (URL endpoints) that listen for requests, process the input using the loaded model, and return a JSON response.

### Step 2.1: Creating the Application File ( `app/app.py` )

- We need to create a Python script in the `/app` folder that will contain the logic for loading the model and defining the API routes.

> **ACTION:** Create a new file `app/app.py` and populate it with the following code. This file cannot be run inside the notebook.

In [2]:

```python
FLASK_APP_CODE = """
import joblib
```

```
from flask import Flask, request, jsonify
import os

# Define paths relative to the app.py location
MODEL_DIR = os.path.join(os.path.dirname(__file__), '..', 'models')
MODEL_PATH = os.path.join(MODEL_DIR, 'svm_text_classifier.pkl')
NAMES_PATH = os.path.join(MODEL_DIR, 'target_names.pkl')

app = Flask(__name__)

# --- Load Model on Startup ---
try:
    model = joblib.load(MODEL_PATH)
    target_names = joblib.load(NAMES_PATH)
    print("Model and Target Names loaded successfully.")
except Exception as e:
    print(f"Error loading model: {e}")
    model = None

# --- Define API Endpoints ---

@app.route('/predict', methods=['POST'])
def predict():
    # Get the data from the POST request (expected JSON with 'text' key)
    data = request.get_json(force=True)
    input_text = data.get('text', '')

    if not model:
        return jsonify({'error': 'Model not loaded'}), 500

    if not input_text:
        return jsonify({'error': 'No text provided'}), 400

    try:
        # The pipeline handles tokenization, vectorization, and classificatic
        prediction_id = model.predict([input_text])[0]
        predicted_label = target_names[prediction_id]

        return jsonify({
            'status': 'success',
            'input': input_text,
            'prediction': predicted_label
        })
    except Exception as e:
        return jsonify({'error': f'Prediction failed: {e}'}), 500

@app.route('/')
def home():
    return "NLP Classification Service is running! Use POST /predict."

if __name__ == '__main__':
    # For production, use a WSGI server like Gunicorn. For local testing:
    app.run(debug=True, host='0.0.0.0', port=5000)
"""
print("The code for app/app.py is displayed above. Please create and save thi
```

```
The code for app/app.py is displayed above. Please create and save this file.
```

## 3. Testing the API (Outside the Notebook)

- To test the deployment, you must leave the Jupyter environment, navigate to the `/app` directory, and run the script from your terminal:

### Step 3.1: Run the Flask Server

- In your terminal:

```
cd app
python app.py
```

(The terminal should show the server running at `http://0.0.0.0:5000/`)

### Step 3.2: Send a Prediction Request

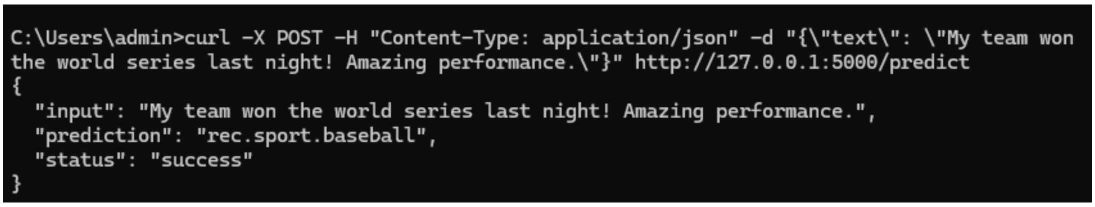
- Use a tool like `curl` (command line) or Postman/VS Code Thunder Client to send a POST request to `http://127.0.0.1:5000/predict` with a JSON body.

**Example Curl Command:**

```
curl -X POST -H "Content-Type: application/json" -d "{\"text\": \"My team won the world series last night! Amazing performance.\"}" http://127.0.0.1:5000/predict
```

- The server should return a JSON response:

```
{
  "input": "My team won the world series last night! Amazing performance.",
  "prediction": "rec.sport.baseball",
  "status": "success"
}
```



```
C:\Users\admin>curl -X POST -H "Content-Type: application/json" -d "{\"text\": \"My team won
the world series last night! Amazing performance.\"}" http://127.0.0.1:5000/predict
{
  "input": "My team won the world series last night! Amazing performance.",
  "prediction": "rec.sport.baseball",
  "status": "success"
}
```

## 4. Summary and Next Steps

- We successfully demonstrated model persistence using `joblib` and created the foundational files for a **Flask API**.

- This process is the bridge between data science and software engineering.

- You have now completed all the structured learning chapters (1-10) of the course.

- In **Chapter 11**, we begin the final **Capstone Project Work**, applying all these skills to larger, multi-step tasks like **Text Summarization** and **Chatbot Development**.

---

## About This Project

This notebook is part of the **Natural Language Processing using Python Programming for Beginners** repository - a comprehensive, beginner-friendly guide for mastering NLP using Python, NLTK, and SpaCy.

**Repository:** `NLP`

## Author

**Prakash Ukhalkar**
GitHub prakash-ukhalkar

---

Built with ❤ for the Python community