### **Step 6: Build a Machine Learning Pipeline**

Building a machine learning pipeline in scikit-learn involves creating a sequence of data transformation and model fitting steps. Each step in the pipeline is defined using a tuple of the form (name, transformer/estimator), where name is a string identifier, and transformer/estimator is a scikit-learn transformer or estimator.

Here’s a breakdown of each step in the pipeline:

#### **Step 6.1: CountVectorizer**

The CountVectorizer converts a collection of text documents into a matrix of token counts. This means it transforms the text data into numerical data that the machine learning algorithm can understand.

python

Copy code

from sklearn.feature\_extraction.text import CountVectorizer

# Example of how CountVectorizer works

count\_vect = CountVectorizer()

X\_counts = count\_vect.fit\_transform(data['message'])

print(X\_counts.shape) # Output: (number of samples, number of unique words)

#### **Step 6.2: TfidfTransformer**

The TfidfTransformer transforms the count matrix from CountVectorizer into a normalized Term Frequency-Inverse Document Frequency (TF-IDF) representation. TF-IDF reflects the importance of a word in a document relative to a collection of documents.

python

Copy code

from sklearn.feature\_extraction.text import TfidfTransformer

# Example of how TfidfTransformer works

tfidf\_transformer = TfidfTransformer()

X\_tfidf = tfidf\_transformer.fit\_transform(X\_counts)

print(X\_tfidf.shape) # Output: (number of samples, number of unique words)

#### **Step 6.3: MultinomialNB**

MultinomialNB is a Naive Bayes classifier for multinomially distributed data. It is particularly suited for text classification with word counts.

python

Copy code

from sklearn.naive\_bayes import MultinomialNB

# Example of how MultinomialNB works

clf = MultinomialNB()

clf.fit(X\_tfidf, data['label'])

#### **Step 6.4: Pipeline**

The Pipeline class in scikit-learn allows you to chain transformers and estimators together so you can use them as a single unit. This is useful for ensuring that all steps are applied in sequence and for simplifying the code.

python

Copy code

from sklearn.pipeline import Pipeline

pipeline = Pipeline([

('vect', CountVectorizer()),

('tfidf', TfidfTransformer()),

('clf', MultinomialNB())

])

#### **Step 6.5: Training the Model**

Fit the pipeline to the training data.

python

Copy code

pipeline.fit(X\_train, y\_train)

#### **Step 6.6: Making Predictions**

Use the trained pipeline to make predictions on the test set.

python

Copy code

y\_pred = pipeline.predict(X\_test)

### **Where to Run This Code**

You can run this code in any Python environment. Here are a few options:

1. **Jupyter Notebook**: Ideal for interactive development and visualization. You can create a new notebook and run each cell step-by-step.
2. **Python Script**: Save the code in a .py file and run it from the command line.
3. **Integrated Development Environment (IDE)**: Such as PyCharm, VS Code, or Spyder. These IDEs provide a convenient way to run and debug your code.

#### **Example in a Jupyter Notebook**

Create a new notebook and run the following cells step-by-step:

1. **Import Libraries**

python

Copy code

import numpy as np

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn.pipeline import Pipeline

from sklearn import metrics

import nltk

from nltk.corpus import stopwords

import string

import joblib

# Download stopwords

nltk.download('stopwords')

stop\_words = set(stopwords.words('english'))

1. **Preprocessing Function**

python

Copy code

def preprocess\_text(text):

text = ''.join([char for char in text if char not in string.punctuation])

text = text.lower()

text = ' '.join([word for word in text.split() if word not in stop\_words])

return text

1. **Load and Preprocess Data**

python

Copy code

data = pd.read\_csv('spam.csv', encoding='latin-1')

data = data[['v1', 'v2']]

data.columns = ['label', 'message']

data['label'] = data['label'].map({'ham': 0, 'spam': 1})

data['message'] = data['message'].apply(preprocess\_text)

1. **Train-Test Split**

python

Copy code

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data['message'], data['label'], test\_size=0.2, random\_state=42)

1. **Build and Train the Pipeline**

python

Copy code

pipeline = Pipeline([

('vect', CountVectorizer()),

('tfidf', TfidfTransformer()),

('clf', MultinomialNB())

])

pipeline.fit(X\_train, y\_train)

1. **Predict and Evaluate**

python

Copy code

y\_pred = pipeline.predict(X\_test)

print(f'Accuracy: {metrics.accuracy\_score(y\_test, y\_pred)}')

print(metrics.classification\_report(y\_test, y\_pred, target\_names=['ham', 'spam']))

### **Complete Example Code to Run**

Here’s the complete code to be run in a Jupyter Notebook or Python script:

python

Copy code

import numpy as np

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import MultinomialNB

from sklearn.pipeline import Pipeline

from sklearn import metrics

import nltk

from nltk.corpus import stopwords

import string

import joblib

# Download stopwords

nltk.download('stopwords')

stop\_words = set(stopwords.words('english'))

# Preprocessing function

def preprocess\_text(text):

# Remove punctuation

text = ''.join([char for char in text if char not in string.punctuation])

# Convert to lowercase

text = text.lower()

# Remove stopwords

text = ' '.join([word for word in text.split() if word not in stop\_words])

return text

# Load the dataset

try:

data = pd.read\_csv('spam.csv', encoding='latin-1', sep='\t', names=['label', 'message'])

except pd.errors.ParserError as e:

print(f"Error reading the CSV file: {e}")

exit()

print("Data loaded successfully")

print(data.head())

# Drop any rows with missing values

data.dropna(inplace=True)

print("After dropping missing values")

print(data.head())

# Encode labels (spam=1, ham=0)

data['label'] = data['label'].map({'ham': 0, 'spam': 1})

# Drop any rows with NaN labels after encoding

data = data.dropna(subset=['label'])

print("After encoding labels and dropping NaNs")

print(data.head())

# Preprocess messages

data['message'] = data['message'].apply(preprocess\_text)

print("After preprocessing messages")

print(data.head())

# Check if the DataFrame is empty

if data.empty:

print("DataFrame is empty after preprocessing. Exiting...")

exit()

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data['message'], data['label'], test\_size=0.2, random\_state=42)

# Build and train the pipeline

pipeline = Pipeline([

('vect', CountVectorizer()),

('tfidf', TfidfTransformer()),

('clf', MultinomialNB())

])

pipeline.fit(X\_train, y\_train)

# Predict on test set

y\_pred = pipeline.predict(X\_test)

# Evaluate the model

print(f'Accuracy: {metrics.accuracy\_score(y\_test, y\_pred)}')

print(metrics.classification\_report(y\_test, y\_pred, target\_names=['ham', 'spam']))

# Save the model

joblib.dump(pipeline, 'spam\_detector.pkl')

# Test with new data

new\_emails = ["Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005.",

"Hey Bob, can we schedule a meeting for tomorrow?"]

new\_emails = [preprocess\_text(email) for email in new\_emails]

predictions = pipeline.predict(new\_emails)

print(predictions)

This should help you understand each step in building a machine learning pipeline for spam email detection.