01_Setup_and_Data

November 22, 2025

1 Notebook 1: Project Setup, Data Acquisition & Initial Assessment

This notebook implements **Step 1** of the Ultimate News Classification Framework Roadmap.

Objectives: 1. **Reproducibility**: Set global random seeds. 2. **Data Acquisition**: Load AG News dataset from Hugging Face. 3. **Data Splitting**: Create Train (for CV), Validation (for tuning), and Test (holdout) splits. 4. **Initial Assessment**: Verify class balance, shapes, and basic statistics. 5. **Persistence**: Save processed splits for downstream tasks.

1.1 1. Reproducibility Infrastructure (Roadmap 1.3)

[1]:
```
import pandas as pd
import numpy as np
import torch
import random
import os
import sys
from datasets import load_dataset
from sklearn.model_selection import train_test_split

# Document Versions
print(f"Python Version: {sys.version}")
print(f"Pandas Version: {pd.__version__}")
print(f"Numpy Version: {np.__version__}")
print(f"Torch Version: {torch.__version__}")

# Global Seed Function
def seed_everything(seed=42):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)
        torch.backends.cudnn.deterministic = True

SEED = 42
```

```
seed_everything(SEED)
print(f"Global Seed set to {SEED}")
```

```
Python Version: 3.11.14 (main, Oct 31 2025, 23:04:14) [Clang 21.1.4 ]
Pandas Version: 2.3.3
Numpy Version: 2.3.5
Torch Version: 2.9.1+cu128
Global Seed set to 42
```

1.2 2. Data Acquisition (Roadmap 1.2)

Loading `sh0416/ag_news` which contains: - **Train**: 120,000 samples - **Test**: 7,600 samples - **Classes**: World (0), Sports (1), Business (2), Sci/Tech (3)

[2]:
```
# Load Dataset
dataset = load_dataset("sh0416/ag_news")

# Convert to Pandas
df_train_full = pd.DataFrame(dataset['train'])
df_test = pd.DataFrame(dataset['test'])

# Preprocessing: Combine Title and Description
# AG News columns are usually ['label', 'title', 'description']
print(f"Original Columns: {df_train_full.columns.tolist()}")

# Check for nulls in original columns
print("Nulls in Train (Original):")
print(df_train_full.isnull().sum())

# Fill NaNs with empty string to avoid NaN in 'text' column
df_train_full['title'] = df_train_full['title'].fillna('')
df_train_full['description'] = df_train_full['description'].fillna('')
df_test['title'] = df_test['title'].fillna('')
df_test['description'] = df_test['description'].fillna('')

df_train_full['text'] = df_train_full['title'] + " " +␣
 ↪df_train_full['description']
df_test['text'] = df_test['title'] + " " + df_test['description']

# Keep only Text and Label
df_train_full = df_train_full[['text', 'label']]
df_test = df_test[['text', 'label']]

# Normalize Labels to 0-3 Range if they are 1-4
unique_labels = sorted(df_train_full['label'].unique())
print(f"Unique Labels found: {unique_labels}")
if unique_labels == [1, 2, 3, 4]:
    print("Detected 1-based indexing. Converting to 0-based...")
```

```
    df_train_full['label'] = df_train_full['label'] - 1
    df_test['label'] = df_test['label'] - 1

# Map Labels for Readability
label_map = {0: 'World', 1: 'Sports', 2: 'Business', 3: 'Sci/Tech'}
df_train_full['label_name'] = df_train_full['label'].map(label_map)
df_test['label_name'] = df_test['label'].map(label_map)

# Verify mapping success
assert df_train_full['label_name'].isnull().sum() == 0, "Mapping failed! Check␣
 ↪label values."

print(f"Train Full Shape: {df_train_full.shape}")
print(f"Test Shape: {df_test.shape}")
```

```
README.md: 0.00B [00:00, ?B/s]

train.jsonl:   0%|          | 0.00/33.7M [00:00<?, ?B/s]

test.jsonl: 0.00B [00:00, ?B/s]

Generating train split:   0%|          | 0/120000 [00:00<?, ? examples/s]

Generating test split:   0%|          | 0/7600 [00:00<?, ? examples/s]

Original Columns: ['label', 'title', 'description']
Nulls in Train (Original):
label          0
title          0
description    0
dtype: int64
Unique Labels found: [np.int64(1), np.int64(2), np.int64(3), np.int64(4)]
Detected 1-based indexing. Converting to 0-based…
Train Full Shape: (120000, 3)
Test Shape: (7600, 3)
```

1.3 3. Splitting Strategy (Roadmap 1.2)

- **Training Set**: Used for Cross-Validation.
- **Validation Split**: 15% of Training Set held out for hyperparameter tuning and early stopping.
- **Test Set**: Official holdout (7,600 samples).

[3]:
```
VAL_SIZE = 0.15  # 15% as per roadmap

train_df, val_df = train_test_split(
    df_train_full,
    test_size=VAL_SIZE,
    stratify=df_train_full['label'],
    random_state=SEED
```

```
)

print(f"New Train Shape (for CV): {train_df.shape}")
print(f"Validation Shape: {val_df.shape}")
print(f"Test Shape: {df_test.shape}")

# Verify Stratification
print("\nClass Balance (Validation):")
print(val_df['label_name'].value_counts(normalize=True))
```

```
New Train Shape (for CV): (102000, 3)
Validation Shape: (18000, 3)
Test Shape: (7600, 3)

Class Balance (Validation):
label_name
Sci/Tech    0.25
Sports      0.25
Business    0.25
World       0.25
Name: proportion, dtype: float64
```

1.4 4. Initial Data Assessment (Roadmap 1.4)

Checking basic statistics and data quality.

[4]:
```
# 1. Null Checks
assert df_train_full.isnull().sum().sum() == 0, "Nulls found in Train!"
assert df_test.isnull().sum().sum() == 0, "Nulls found in Test!"

# 2. Class Balance Check
train_counts = train_df['label_name'].value_counts()
print("\nTrain Class Counts:")
print(train_counts)

# 3. Document Length Stats (Basic)
train_df['word_count'] = train_df['text'].apply(lambda x: len(str(x).split()))
print("\nWord Count Statistics (Train):")
print(train_df['word_count'].describe())
```

```
Train Class Counts:
label_name
Sci/Tech    25500
World       25500
Sports      25500
Business    25500
Name: count, dtype: int64
```

```
Word Count Statistics (Train):
count    102000.000000
mean         37.871020
std          10.117152
min           8.000000
25%          32.000000
50%          37.000000
75%          43.000000
max         171.000000
Name: word_count, dtype: float64
```

1.5 5. Persistence

Saving processed splits to `data/processed/` for subsequent steps.

[5]:
```python
os.makedirs('../data/processed', exist_ok=True)

cols_to_save = ['text', 'label', 'label_name']

train_df[cols_to_save].to_csv('../data/processed/train.csv', index=False)
val_df[cols_to_save].to_csv('../data/processed/val.csv', index=False)
df_test[cols_to_save].to_csv('../data/processed/test.csv', index=False)

print("Datasets saved to ../data/processed/")
```

```
Datasets saved to ../data/processed/
```