Building End-to-End NLP Systems

Lecture 6: NLP Pipelines

Harito

September 15, 2025

Recap: Our NLP Components

So far, we've built several key components:

- **Tokenizer** (Lab 1): Breaks text into words.
- **Vectorizer** (Lab 2 3): Converts words into numerical vectors (Count, TF-IDF).
- **TextClassifier** (Lab 5): Learns to predict labels from these vectors.

Each component performs a specific task, but in a real-world application, they need to work together seamlessly.

The Need for Pipelines

Imagine processing a new document for classification:

1. Take the raw text.
2. Pass it to the tokenizer.
3. Take the tokens and pass them to the vectorizer.
4. Take the vector and pass it to the classifier.
5. Get the final prediction.

This sequential process is exactly what an **NLP Pipeline** formalizes.

What is an NLP Pipeline?

Definition

An NLP pipeline is a sequence of processing steps applied to raw text to achieve a specific NLP task. The output of one component serves as the input for the next.

Benefits of Using Pipelines

Pipelines are not just about convenience; they offer significant advantages:

- **Modularity**: Each step is a distinct, interchangeable component. You can easily swap out a ‘SimpleTokenizer‘ for a ‘RegexTokenizer‘.
- **Reusability**: Once a pipeline is defined, it can be applied consistently to any new data.
- **Consistency**: Ensures that the exact same preprocessing and modeling steps are applied during both training and prediction.
- **Simplicity**: Reduces boilerplate code and makes complex workflows easier to manage and understand.
- **Deployment**: Simplifies deploying your NLP solution, as the entire process is encapsulated.

Scikit-learn's Pipeline (Concept)

'scikit-learn' provides a powerful 'Pipeline' class that allows you to chain multiple estimators.
python from sklearn.pipeline import Pipeline from sklearn.feature$_e$xtraction.textimportTfidfVectorizerfromsklearn.linear$_m$odelimp
This is a conceptual example using sklearn's components
text$_c$lf = $Pipeline([('tfidf', TfidfVectorizer()), ('clf', LogisticRegression()),])$
You can then fit and predict with the whole pipeline
text$_c$lf.fit($X_t$rain, $y_t$rain)predicted = text$_c$lf.predict($X_t$est)
For our lab, we will build a custom pipeline using our own components.

Next Steps

Time for Lab 6!

Objective:

- Implement a ‘TextPipeline‘ class.
- Integrate your ‘Tokenizer‘, ‘Vectorizer‘, and ‘TextClassifier‘.
- Demonstrate end-to-end text processing.