Text Classification for Economics

A Practical Introduction to Python & Machine Learning

HKU ECON6087 Tutorial

2026 年 2 月 5 日

Objective

- **Goal**: Classify news articles into categories (World, Sports, Business, Sci/Tech).
- **Tools**: Python and its data science libraries.
- **Method**: K-Nearest Neighbors (KNN).
- **Key Learning**: How specific text representations (Bag of Words vs. TF-IDF) affect model performance.

Python Libraries

The Python Data Stack

We use three main libraries—think of them as your digital toolbox:

- **datasets (Hugging Face)**: Like a library catalog. It lets us easily download and access standard datasets without manual searching.
- **pandas (Data Manipulation)**: The ”Excel of Python”. It handles data tables (DataFrames), allowing us to load, view, and clean data efficiently.
- **scikit-learn (Machine Learning)**: The standard toolkit for machine learning. It contains the algorithms (like KNN) and tools to transform text into numbers.

The Data

The Dataset: AG News

We use the AG News dataset used in the tutorial.

- Content: News headlines and short descriptions.
- Classes: 4 Topics
  1. World
  2. Sports
  3. Business
  4. Sci/Tech
- Size:
  - Training Set: 120,000 articles (used to teach the model).
  - Testing Set: 7,600 articles (used to evaluate performance).

Step 1: Loading Data

Step 1: Loading and Saving Data

We first load the data from the cloud and save it locally as CSV files.

```python
from datasets import load_dataset

# 1. Download data
dataset = load_dataset("ag_news")

# 2. Save as CSV (for Pandas to read later)
for split in dataset.keys():
    # dataset[split] is the data (train or test)
    # .to_csv() saves it to a file
    dataset[split].to_csv(f"{split}.csv")
```

Result: We now have ‘train.csv‘ and ‘test.csv‘.

Concept: K-Nearest Neighbors

Concept: K-Nearest Neighbors (KNN)

Intuition
”Tell me who your neighbors are, and I’ll tell you who you are.”

To classify a new document:

1. We place the new document in the ”space” of all known documents.
2. We find the K closest documents (neighbors).
3. We look at their labels (topics).
4. We assign the majority label to the new document.

Challenge for Text: How do we measure distance between text? We must turn words into numbers.

Step 2: Bag of Words

Text Representation 1: Bag of Words (BoW)

The simplest way to turn text into numbers.

- We list all unique words in the dataset (the vocabulary).
- For each document, we simply **count** how many times each word appears.

例
”The economy grows” $\rightarrow$ {’the’: 1, ’economy’: 1, ’grows’: 1, ’football’: 0}

Pros: Simple.
Cons: Ignores grammar. Common words like ”the” appear frequently but have little meaning.

Implementing Bag of Words

In Python, `CountVectorizer` creates the Bag of Words matrix.

```python
from sklearn.feature_extraction.text import CountVectorizer

# Create a "counter" that looks at the top 5000 words
# stop_words='english' removes common words like "the", "is", "
    at"
vectorizer = CountVectorizer(stop_words='english', max_features
    =5000)

# Learn the vocabulary and count words in training data
X_train_bow = vectorizer.fit_transform(train_df['text'])
# Count words in test data (using same vocabulary)
X_test_bow = vectorizer.transform(test_df['text'])
```

Results: KNN with Bag of Words

```python
from sklearn.neighbors import KNeighborsClassifier

# Create the model (look at 5 nearest neighbors)
knn = KNeighborsClassifier(n_neighbors=5)

# Train: Model memorizes the training data
knn.fit(X_train_bow, train_df['label'])

# Predict and Evaluate
accuracy = knn.score(X_test_bow, test_df['label'])
print(f"Accuracy: {accuracy}")
```

Result
Accuracy $\approx$ 72.5%

Step 3: TF-IDF

Text Representation 2: TF-IDF

Term Frequency - Inverse Document Frequency.

A smarter way to count.

- **TF (Term Frequency)**: How often word *w* appears in document *d*.
- **IDF (Inverse Document Frequency)**: How rare is word *w* across *all* documents?

Idea: If a word appears in *every* document (e.g., ”said”, ”today”), it’s not useful for classification. We lower its weight. If a word is rare (e.g., ”touchdown”, ”deficit”), it gets a high weight.

Implementing TF-IDF

Changing the code is easy: swap `CountVectorizer` for `TfidfVectorizer`.

```python
from sklearn.feature_extraction.text import TfidfVectorizer

# Use TF-IDF instead of simple counts
tfidf_vectorizer = TfidfVectorizer(stop_words='english',
    max_features=5000)

# Transform data
X_train_tfidf = tfidf_vectorizer.fit_transform(train_df['text'])
X_test_tfidf = tfidf_vectorizer.transform(test_df['text'])
```

Results: KNN with TF-IDF

We train the exact same KNN model, just using the new weighted data.

```python
# Train KNN on TF-IDF data
knn_tfidf = KNeighborsClassifier(n_neighbors=5)
knn_tfidf.fit(X_train_tfidf, train_df['label'])

# Evaluate
accuracy_tfidf = knn_tfidf.score(X_test_tfidf, test_df['label'])
print(f"Accuracy: {accuracy_tfidf}")
```

Result
Accuracy $\approx$ 89.0%

Takeaway: Weighting words by importance (TF-IDF) massively improves our simple neighbor-based classifier.

Extension: Chinese Text

Working with Chinese Data

Challenge: English words are separated by spaces. Chinese words are not.

- English: ”I love economics” $\rightarrow$ [”I”, ”love”, ”economics”]
- Chinese: ” **我爱经济学**” $\rightarrow$?

Solution: Word Segmentation (Tokenization).

- We use a library called **`jieba`** (”Stutter” in Chinese) to cut sentences into words.
- ” **我爱经济学**” $\xrightarrow{jieba}$ [” **我**”, ” **爱**”, ” **经济学**”]

Chinese Segmentation with Jieba

```python
import jieba

# Example Function
def segment_text(text):
    # jieba.cut returns a generator, we join it with spaces
    return " ".join(jieba.cut(text))

# Apply to a sentence
print(segment_text("我爱经济学"))
# Output: "我 爱 经济学"
```

Once the text is space-separated, we can use `CountVectorizer` and `TfidfVectorizer` exactly as before!

Conclusion

Summary

1. Pandas allows economists to handle large datasets easily.
2. Scikit-Learn provides plug-and-play machine learning tools.
3. Text must be converted to numbers before analysis.
4. TF-IDF is often superior to simple counting because it filters out ”noise” and highlights unique keywords.
5. Even a simple algorithm like KNN can achieve high accuracy ($\sim$89%) with the right data representation.