Python Tutorial: ML Applications in NLP

January 18, 2025

Introduction

This tutorial demonstrates the application of Machine Learning (ML) in Natural Language Processing (NLP). The key components covered are:

- Feature Extraction (Bag of Words, TF-IDF)
- Model Development: Training and Testing
- Classification and Prediction
- Error Analysis

1 Bag of Words (BoW)

The **Bag of Words (BoW)** is one of the simplest methods for feature extraction in text processing. In BoW, a text is represented as the frequency (or presence) of words in a document, while disregarding the grammar, word order, or meaning.

1.1 Concept

1. Build a vocabulary of all unique words in the dataset. 2. Represent each document as a vector, where:

- Each element of the vector corresponds to a word in the vocabulary.
- The value is the frequency of that word in the document.

1.2 Example

Corpus:

1. Document 1: *"I love programming in Python."*
2. Document 2: *"Python programming is fun."*

Vocabulary: [I, love, programming, in, Python, is, fun]

Frequency Representation:

Word	Doc 1	Doc 2
I	1	0
love	1	0
programming	1	1
in	1	0
Python	1	1
is	0	1
fun	0	1

Each document is represented as:

- Document 1: $[1, 1, 1, 1, 1, 0, 0]$
- Document 2: $[0, 0, 1, 0, 1, 1, 1]$

1.3 Limitations of BoW

- It ignores word context and semantics.
- Large vocabularies can lead to high-dimensional, sparse representations.
- Frequently used words (e.g., "the", "is") can dominate the representation.

2 TF-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) improves on BoW by taking into account how important a word is relative to the entire corpus.

2.1 Concept

TF-IDF combines two measures:

- **Term Frequency (TF)**: Measures how frequently a word appears in a document.

$$\text{TF}(w, d) = \frac{\text{Number of occurrences of } w \text{ in } d}{\text{Total words in } d}$$

- **Inverse Document Frequency (IDF)**: Measures the rarity of a word across all documents.

$$\text{IDF}(w, D) = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing } w}\right)$$

$$\text{TF-IDF}(w, d, D) = \text{TF}(w, d) \cdot \text{IDF}(w, D)$$

2.2 Example

For the same corpus:

1. Compute TF for each word:

$$\text{TF}(\text{Python}, \text{Doc 1}) = \frac{1}{5}, \quad \text{TF}(\text{fun}, \text{Doc 2}) = \frac{1}{4}$$

2. Compute IDF for each word:

$$\text{IDF}(\text{programming}) = \log\left(\frac{2}{2}\right) = 0, \quad \text{IDF}(\text{love}) = \log\left(\frac{2}{1}\right) = \log 2$$

3. Compute TF-IDF values:

Word	TF-IDF (Doc 1)	TF-IDF (Doc 2)
I	0.22	0
love	0.22	0
programming	0.00	0.00
in	0.22	0
Python	0.22	0.22
is	0	0.22
fun	0	0.22

Each document is represented as:

- Document 1: $[0.22, 0.22, 0, 0.22, 0.22, 0, 0]$
- Document 2: $[0, 0, 0, 0, 0.22, 0.22, 0.22]$

2.3 Advantages of TF-IDF

- Reduces the impact of common words like "is" and "the".
- Highlights words that are significant for a document.
- Produces a denser representation compared to BoW.

3 Python Implementation

In this section, we will implement both the Bag of Words (BoW) and TF-IDF methods, followed by training a machine learning model to classify text data.

3.1 Step 1: Import Required Libraries

First, we need to import the required libraries for text processing and machine learning.

```
import numpy as np
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer,
    TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, accuracy_score
```

3.2 Step 2: Load the Dataset

We will use the 20 Newsgroups dataset from Scikit-learn, which contains documents from 20 different categories. The dataset is easily accessible from ‘sklearn.datasets‘.

```
# Load the 20 newsgroups dataset
categories = ['alt.atheism', 'comp.graphics', 'sci.med', 'soc.religion.
    christian']
data = fetch_20newsgroups(subset='all', categories=categories, remove
    =('headers', 'footers', 'quotes'))
```

3.3 Step 3: Text Vectorization (BoW and TF-IDF)

We will create both BoW and TF-IDF representations for our dataset.

3.3.1 Bag of Words (BoW)

```
# Bag of Words
bow_vectorizer = CountVectorizer(stop_words='english', max_features
    =500)
bow_features = bow_vectorizer.fit_transform(data.data)
```

3.3.2 TF-IDF Representation

```
# TF-IDF Vectorization
tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_features
    =500)
tfidf_features = tfidf_vectorizer.fit_transform(data.data)
```

3.4 Step 4: Model Development and Training

We will use the Multinomial Naive Bayes classifier, which is well-suited for text classification tasks.

3.4.1 Train-Test Split

```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(bow_features, data.
    target, test_size=0.3, random_state=42)
```

3.4.2 Train a Naive Bayes Classifier

```
# Train Naive Bayes model using BoW features
model = MultinomialNB()
model.fit(X_train, y_train)
```

3.5 Step 5: Evaluation (Prediction and Error Analysis)

Once the model is trained, we make predictions on the test set and evaluate its performance using common classification metrics like accuracy and F1-score.

3.5.1 Prediction and Accuracy

```
# Make predictions
y_pred = model.predict(X_test)

# Accuracy Score
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

3.5.2 Classification Report

```
# Classification report
print(classification_report(y_test, y_pred, target_names=data.
    target_names))
```