Mohammad Idrees Bhat

Tech Skills Trainer | AI/ML Consultant

NLP with Python

Using NLTK and SpaCy for text classification

AGENDA

1. Introduction to Text Classification
2. Overview of Nltk and SpaCy
3. Text classification in Nltk
4. Text classification in spaCy

"You were born with wings, why prefer to crawl through life?"
- Rumi

What does this mean to you? How might AI interpret 'wings' literally and miss the deeper meaning?

1. Introduction to Text Classification

Text classification is a Natural Language Processing (NLP) task where text is analyzed, understood, and categorized into predefined classes or categories.

It involves assigning a label or category to a given piece of text based on its content.

- Categorizing text into predefined categories based on its content.
- **Examples:**
  - Email filtering (spam vs. not spam).
  - Sentiment analysis (positive, negative, neutral).
  - News categorization (politics, sports, technology).

- **Importance:** Automates the understanding of large volumes of text.

Applications of Text Classification

- **Sentiment Analysis:** Classifying text as positive, negative, or neutral (e.g., analyzing customer reviews).

Spam Detection: Identifying spam or non-spam emails.

- **Topic Categorization:** Grouping articles or documents by topic (e.g., politics, sports, technology).
- **Language Detection:** Identifying the language of a given text.
- **Intent Detection:** Understanding the intent behind a query in chatbots or virtual assistants.

Methods Used in Text Classification

- **Rule-Based Systems:** Predefined rules are used to classify text based on patterns or keywords. For example, an email containing the phrase "Congratulations, you've won!" could be classified as spam.

- **Machine Learning Models:** Algorithms such as *Naive Bayes*, *Support Vector Machines (SVM)*, or *Logistic Regression* are trained on numerical features extracted from text data. These models learn patterns from labeled datasets to make predictions.

- **Deep Learning Models:** Advanced neural network architectures like Convolutional Neural Networks (CNNs), or Transformers (e.g., BERT, GPT) are used for understanding and classifying complex text data, especially when large datasets are available.

2. Overview of NLTK and SpaCy

NLTK

- **NLTK:**
  - Python library for working with text.
  - Designed primarily as an educational tool for teaching and learning NLP.
  - For basic NLP tasks like tokenization, stopword removal, stemming, lemmatization, POS tagging, and syntactic parsing (where a sentence is analyzed to determine its grammatical structure).
  - Generally slower because it processes text one step at a time and is designed more for research and teaching.
- NLTK can be slower when dealing with large datasets or real-time processing due to its older design.

- NLTK Website
- NLTK Book
- NLTK Sentiment Analysis

Building a Text Classification Pipeline with NLTK

1. **Import libraries.**
2. **Load and preprocess text data.**
3. **Feature extraction using Bag of Words or TF-IDF.**
4. **Train a simple Naive Bayes classifier.**
5. **Test and evaluate the classifier.**

spaCy

- **SpaCy:**
  - Built for production-level tasks and focuses on efficiency and performance. It is optimized for industrial use cases, handling large-scale data and real-time NLP applications.
  - Known for its speed and efficiency.
  - For advanced preprocessing like lemmatization, POS tagging, dependency parsing, named entity recognition (NER), and word embeddings.

- spaCy 101 A brief introduction
- spaCy Course Advanced NLP with spaCy

Text Classification with SpaCy

1. **Load SpaCy and process text.**
2. **Extract linguistic features (POS, entities, etc.).**
3. **Train a classifier using features.**
4. **Evaluate the classifier.**

3. Text Classification with NLTK

In [ ]:

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.classify import NaiveBayesClassifier

# Download NLTK resources
# Download necessary datasets
nltk.download('movie_reviews')  # Movie review dataset
nltk.download('stopwords')      # Stopwords for preprocessing
nltk.download('punkt')          # Tokenizer for text processing
```

We create a small dataset with sentences labeled as "pos" (positive) or "neg" (negative).

In [ ]:
```
# Step 1: Sample dataset of sentences labeled as positive or negative
training_data = [
    ("I love this movie", "pos"),
    ("This film is amazing", "pos"),
    ("I hated this movie", "neg"),
    ("This film is terrible", "neg")
]
```

Preprocessing:

- Tokenization: Break the sentence into individual words.
- Stopword Removal: Remove common words like "the", "and", etc., to focus on meaningful words.
- Feature Extraction: We create a dictionary where the keys are words, and the values are True (indicating the presence of the word in the sentence).

In [ ]:
```
# Step 2: Preprocessing and Feature Extraction
def extract_features(sentence):
    words = word_tokenize(sentence.lower())  # Tokenize and convert to lowercase
    stop_words = set(stopwords.words('english'))  # Get stop words
    words = [word for word in words if word.isalpha() and word not in stop_words
    return {word: True for word in words}  # Create feature dictionary with word

# Convert training data into feature sets
training_features = [(extract_features(sentence), label) for sentence, label in
```

Naive Bayes Classifier: We train the classifier on the feature set.

In [ ]:
```
# Step 3: Train Naive Bayes Classifier
classifier = NaiveBayesClassifier.train(training_features)
```

Testing: We classify new sentences using the trained classifier.

In [ ]:
```
# Step 4: Test the classifier with new sentences
test_sentences = [
    "I really enjoyed this movie",
    "This movie was awful"
]
```

In [ ]:
```
for sentence in test_sentences:
    features = extract_features(sentence)
    predicted_label = classifier.classify(features)
    print(f"Sentence: '{sentence}' => Predicted Sentiment: {predicted_label}")
```

4. Text Classification with SpaCy

In [ ]:
```
#pip install spacy
```

In [ ]:
```
#pip install blis
```

In [ ]:
```
!pip install spacy
```

In [ ]:
```
import spacy
from spacy.pipeline.textcat import Config, ConfigSchema, TextCategorizer
from spacy.training.example import Example
```

In [ ]:
```
#pip show spacy pydantic
```

In [ ]:
```
#pip install "pydantic<2.0"
```

In [ ]:
```
!pip install spacy
```

In [ ]:
```
pip install --upgrade pip
```

In [ ]:
```
pip show spacy
```

In [ ]:

In [ ]:

In [ ]:
```
# Load SpaCy's pre-trained English model
nlp = spacy.load('en_core_web_sm')

# Step 1: Create the text classifier component
text_cat = nlp.create_pipe('textcat', config={"exclusive_classes": True, "archit
nlp.add_pipe(text_cat, last=True)
```

In [ ]:
```
# Add labels (positive and negative)
text_cat.add_label("pos")
text_cat.add_label("neg")
```

In [ ]:
```
# Step 2: Prepare the training data
# The data format is [(text, label), where label is 'pos' or 'neg']
training_data = [
    ("I love this movie", {"cats": {"pos": 1, "neg": 0}}),
    ("This film is amazing", {"cats": {"pos": 1, "neg": 0}}),
    ("I hated this movie", {"cats": {"pos": 0, "neg": 1}}),
    ("This film is terrible", {"cats": {"pos": 0, "neg": 1}})
]
```

In [ ]:
```
# Convert training data to SpaCy's format
train_examples = []
for text, annot in training_data:
    doc = nlp.make_doc(text)
    example = Example.from_dict(doc, annot)
    train_examples.append(example)
```

In [ ]:
```
# Step 3: Train the classifier
optimizer = nlp.begin_training()
for epoch in range(10):  # 10 epochs for training
    losses = {}
    # Shuffle and batch the examples
    # Example usage of batching (make sure you shuffle the training data)
    # this is a simple approach with no batch strategy.
    for batch in spacy.util.minibatch(train_examples, size=2):
```

```
        nlp.update(batch, losses=losses)
    print(f"Epoch {epoch} Losses: {losses}")
```

In [ ]:

```


# Step 4: Test the classifier on new sentences
test_sentences = [
    "I really enjoyed this movie",
    "This movie was awful"
]

for sentence in test_sentences:
    doc = nlp(sentence)
    print(f"Sentence: '{sentence}' => Predicted Sentiment: {'positive' if doc.ca
```

In [ ]:

```

```

In [ ]:

```

```

Comprehensive Text Classification with NLTK including feature engineering (Optional)

What We Will Cover

1. Understanding Text Classification
2. Dataset Preparation
3. Preprocessing Steps
4. Feature Extraction
5. Training a Naive Bayes Classifier
6. Testing and Evaluating the Model

1. Understanding Text Classification

Text classification involves assigning a category to a given piece of text. For example:

- Labeling emails as **"Spam"** or **"Not Spam."**
- Categorizing movie reviews as **"Positive"** or **"Negative."**

In this exercise, we will classify movie reviews as either **Positive** or **Negative** using NLTK.

In [ ]:

```
# Import necessary libraries
import nltk
from nltk.corpus import movie_reviews
from nltk.corpus import stopwords
from nltk.classify import NaiveBayesClassifier
from nltk.classify.util import accuracy
```

2. Dataset Preparation

NLTK comes with a built-in dataset for movie reviews called `movie_reviews`. Each review in this dataset is pre-labeled as positive or negative.

In [ ]:

```
# Download necessary datasets
nltk.download('movie_reviews')  # Movie review dataset
```

```
nltk.download('stopwords')       # Stopwords for preprocessing
nltk.download('punkt')           # Tokenizer for text processing
```

In [ ]:

```
# Load Dataset
# Extract movie reviews and their associated labels
documents = [(list(movie_reviews.words(fileid)), category)
             for category in movie_reviews.categories()
             for fileid in movie_reviews.fileids(category)]
```

3. Preprocessing Steps

Before training the classifier, we need to preprocess the text:

1. **Tokenization:** Breaking text into individual words.
2. **Removing Stop Words:** Eliminating common, unimportant words like "the" and "is."
3. **Lowercasing:** Converting all words to lowercase to maintain consistency.

In [ ]:

```
# Tokenization
# Example of tokenizing one document for clarity
sample_document = documents[0][0]  # Get the first document (words)
print(f"Original Document: {sample_document[:20]}")  # Print the first 20 words

# Lowercasing
# Convert all words to lowercase
lowercased_words = [word.lower() for word in sample_document]
print(f"Lowercased Words: {lowercased_words[:20]}")

# Removing Non-Alphabetic Tokens
# Remove words that are not purely alphabetic
alphabetic_words = [word for word in lowercased_words if word.isalpha()]
print(f"Alphabetic Words: {alphabetic_words[:20]}")

# Stop Word Removal
# Load stopwords and remove them from the dataset
stop_words = set(stopwords.words('english'))
filtered_words = [word for word in alphabetic_words if word not in stop_words]
print(f"Filtered Words (No Stopwords): {filtered_words[:20]}")
```

4. Feature Extraction

We will use the following method for feature extraction:

- Represent each review as a list of words and create a feature set where each word's presence is marked as **True** or **False.**
- The goal is to extract features from each document in the dataset after preprocessing. These features are required for training the Naive Bayes classifier.

In [ ]:

```
#Extract Features

# Create a list of all words in the dataset
all_words = nltk.FreqDist(word.lower() for word in movie_reviews.words() if word

# Use the 2000 most common words as features
word_features = list(all_words.keys())[:2000]
```

In [ ]:

```
# Define a feature extractor function
def document_features(document):
    document_words = set(document)
```

```
    features = {word: (word in document_words) for word in word_features}
    return features
```

In [ ]:
```
# Preprocess all documents
preprocessed_documents = []
for (doc, category) in documents:
    # Step 1: Lowercase and remove stopwords + non-alphabetic tokens
    filtered_words = [word.lower() for word in doc if word.isalpha() and word.lc
    # Step 2: Apply feature extraction
    features = document_features(filtered_words)
    preprocessed_documents.append((features, category))
```

5. Training a Naive Bayes Classifier

Naive Bayes is a simple yet powerful algorithm for text classification. It works well with small datasets and uses probabilities to predict the most likely category.

In [ ]:
```
# Split the dataset into training and testing sets (80% training, 20% testing)
train_set = preprocessed_documents[:1600]
test_set = preprocessed_documents[1600:]
```

In [ ]:
```
# Train the Classifier
classifier = NaiveBayesClassifier.train(train_set)
```

6. Testing and Evaluating the Model

After training, we'll test the classifier on new data to measure its accuracy.

In [ ]:
```
# Evaluate the Model
print(f"Accuracy: {accuracy(classifier, test_set) * 100:.2f}%")
```

In [ ]:
```
# Test with New Data
test_review = "This movie was absolutely amazing, with great performances and a
test_tokens = nltk.word_tokenize(test_review)
test_words = [word.lower() for word in test_tokens if word.isalpha() and word.lc
test_features = document_features(test_words)
print(f"Prediction for test review: {classifier.classify(test_features)}")
```

In [ ]:
```
# Display the Most Informative Features
print("\nMost Informative Features:")
classifier.show_most_informative_features(10)
```

Feature (e.g., "chick"): These are the words found in the text that are most useful for predicting sentiment.

Association (e.g., "neg : pos = 8.6 : 1.0"): This shows how strongly the word is linked to one class. For example, "chick" is 8.6 times more likely to appear in negative reviews than positive ones.

Additional Tools used in NLP

1. Preprocessing Tools

- **NLTK (Natural Language Toolkit):** A comprehensive library for basic NLP tasks such as tokenization, stemming, lemmatization, and syntactic parsing.
- **SpaCy:** An efficient library for advanced NLP tasks like dependency parsing, named entity recognition (NER), and word embeddings.

2. Text Representation Tools

- **Gensim:** A library for creating word embeddings and topic modeling using techniques like Word2Vec and LDA.
- **Transformers (by Hugging Face):** Provides access to pre-trained models like BERT and GPT for text embeddings and advanced NLP tasks.

3. Data Manipulation and Visualization Tools

- **Pandas:** A powerful library for cleaning, organizing, and manipulating textual datasets.
- **Matplotlib/Seaborn:** Libraries for visualizing data trends, such as word frequencies and sentiment scores.
- **WordCloud:** A simple tool for creating visual word clouds to represent text data.

4. Machine Learning and Deep Learning Tools

- **Scikit-Learn:** A versatile library for training machine learning models like Naive Bayes and SVM on text data.
- **TensorFlow/Keras:** A deep learning framework for building and training neural networks for NLP tasks.
- **PyTorch:** A flexible deep learning library for implementing advanced NLP models.

5. Data Acquisition Tools

- **BeautifulSoup:** A library for web scraping and extracting text data from HTML pages.
- **Scrapy:** An advanced tool for building web scrapers to gather large amounts of textual data.
- **Tweepy:** A Python library for accessing Twitter data via the Twitter API.

6. Annotation Tools

- **Label Studio:** A user-friendly tool for manually annotating text datasets for training machine learning models.
- **Prodigy:** A more advanced annotation tool that incorporates active learning to improve dataset quality.

7. Cloud NLP Tools (Optional)

- **Google Cloud Natural Language API:** A cloud-based service for sentiment analysis, entity extraction, and more.
- **AWS Comprehend:** Amazon's NLP service for text classification, entity recognition, and sentiment analysis.

- **Azure Text Analytics:** A cloud tool for performing key phrase extraction, sentiment analysis, and more.

Live Exercise

Now it's your turn!

Task 1: description of task

```
- instructions
```

END

THANK YOU!

Live Exercise Solutions

In [ ]:

```
solutions
```

Programming Interveiw Questions

1. topic:
   - question

In [ ]:

Mohammad Idrees Bhat

Tech Skills Trainer | AI/ML Consultant