Using NLP for Sentiment Analysis

This guide will walk you through the process of leveraging Natural Language Processing (NLP) techniques to perform sentiment analysis on textual data. Whether you're working with social media posts, customer reviews, or any other form of text, sentiment analysis can help you gain valuable insights into the emotions and opinions expressed within the content.

In this guide, you'll learn how to:

1. **Load Your Data:** Start by loading your dataset, whether it's stored locally or on Google Drive.

2. **Set Up the Environment:** Install the TweetNLP library, a powerful collection of pretrained models designed for various NLP tasks.

3. **Predict Sentiments:** Use the TweetNLP library to predict sentiment labels for individual pieces of text.

4. **Systematically Analyze Text:** Expand your analysis by predicting sentiments across an entire dataset and calculating the distribution of emotions.

By following this guide, you'll gain the tools and knowledge needed to conduct comprehensive sentiment analysis and extract meaningful insights from your text data.

Set up

Load the data from a CSV file, either locally or through Google Drive:

```
# Replace this file path with your own
csv ="/content/drive/MyDrive/dataset_reddit-scraper-lite_2024-04-15_20-52-19-472.csv"

import pandas as pd

df = pd.read_csv(csv)
```

Install and import the TweetNLP library, a collection of pretrained models and resources:

```
pip install tweetnlp
import tweetnlp
```

Find a model you want to use in the TweetNLP Library ⇗, then load it in:

```
model = tweetnlp.load_model('topic_classification',
model_name='cardiffnlp/twitter-roberta-base-emotion-multilabel-latest')
```

Individual Prediction:

You can now use the model to predict sentiment labels for individual strings. To make this process more scalable, consider writing functions to perform these predictions systematically.

Example Prediction:

```
model.predict("I bet everything will work out in the end :)")
```

Output:

```
{'label': ['joy', 'optimism']}
```

Systemic prediction:

To predict sentiments for each item in your list:

```
from tqdm import tqdm
predictions = []

for item in tqdm(text_data, desc="Processing"):
    prediction = model.predict(item)
    predictions.append(prediction)
```

For a high-level overview of sentiment within the text, calculate the percentage of each label:

```
from collections import Counter

labels = [item['label'] for item in predictions]

# Flattens the list of lists into a single list
all_labels = [label for sublist in labels for label in sublist]

# Calculates the percentage of each label
label_counts = Counter(all_labels)
total_predictions = len(all_labels)
label_percentages = {label: (count / total_predictions) * 100
for label, count in label_counts.items()}

# Prints the percentages
for label, percentage in label_percentages.items():
    print(f"{label}: {percentage:.2f}%")
```

Output:

```
joy: 13.68%
fear: 7.48%
sadness: 16.70%
disgust: 18.60%
pessimism: 2.72%
optimism: 15.73%
anticipation: 6.10%
anger: 16.39%
surprise: 0.46%
love: 2.15%
```

For individual analysis, connect the predictions to their corresponding text. This allows you to filter the text DataFrame by these predictions:

```
from tqdm import tqdm
augmented_text_data = []

for idx, (item, prediction) in tqdm(enumerate(zip(text_data, predictions)),
total=len(text_data), desc="Processing"):
    augmented_text_data.append((item, prediction))
```

You can now sort and access text by their predicted labels:

```
import random
# Change surprise to the emotion you wish to filter by
label_visual = "surprise"

# Filters DataFrame by the specified label
filtered_examples = [(item, prediction) for item,
prediction in augmented_text_data if label_visual in prediction['label']]

# Change 3 to the number of samples you wish to see
random_filtered_examples = random.sample(filtered_examples, k=min(3, len(filtered_example

for example in random_joy_examples:
    print("Item:", example[0])
    print()
    print()
```