NLI_stance_detection

November 28, 2023

1 Stance Detection: Zero-shot NLI Classifier

This tutorial demonstrations stance dection with a zero-shot NLI classifier. Our task is to classify support for Trump in a data set of tweets. To accomplish this, we will use a DeBERTa-Large model trained for natural language inference (NLI). The Transformers library will provide access to pre-trained language models as well as an easy to use pipeline for classification. Read the Transformers documentation

Explore the repository of pre-trained models

Requirements:

1. A very basic understanding of Python.
2. Access to a GPU is beneficial, but not necessarily required for smaller data sets. Free services like Google Colab can be used if you don't have a desktop GPU.

[1]:
```
import pandas as pd
from transformers import pipeline
from sklearn.metrics import matthews_corrcoef
```

For this example we will use a data set consists of tweets about President Trump that are manually labeled 1: approve, 0: not approve.

[2]:
```
test_df = pd.read_csv('https://raw.githubusercontent.com/XXXXX/
↪stance_detection_tutorials/main/data/test.csv')
```

The Transformers library offers a simple pipeline we can use to classify the data. All we need to do is specify the task and the model we will use. More information on the model can be found here

Setting the device argument to zero tells the classifier to use the GPU, and the batch_size determines how many documents are passed through the model at a time. Higher batch sizes require more memory, so try lowering the batch size if you run out.

[3]:
```
classifier = pipeline("zero-shot-classification", model='MoritzLaurer/
↪DeBERTa-v3-large-mnli-fever-anli-ling-wanli', device = 0, batch_size = 64)
```

Now we just prepare our data for classification by placing all of our documents in a list, and creating the hypotheses that will be used for classification. NLI classifiers work by pairing documents with "hypotheses" and determining if the hypothesis is true given the information in the text.

To create our hypotheses we start with a basic template, in this case we use "The author of this tweet ____ Trump." We then fill in the blank with possible labels, in this case "supports", "opposes", and

"does not express an opinion about." It's good to have a set of hypotheses that represent positive, negative, and neutral stances.

[4]:
```
samples = list(test_df['text'])
template = 'The author of this tweet {} Trump.'
labels = ['supports', 'opposes', 'does not express an opinion about']
```

Now we classify the data by passing our documents, labels, and template to the classifier. The model will pair each document with each of the three hypotheses: * The author of this tweet supports Trump. * The author of this tweet opposes Trump. * The author of this tweet does not express an opinion about Trump.

It well then determine the probablility that each hypothesis is true given the document. The assigned label will be the hypothesis that is most likely to be true.

[5]:
```
# classify the documents
res = classifier(samples, labels, hypothesis_template = template, multi_label =␣
↪False)
# return the most probable label and add it to our data frame
test_df['zs_labels'] = [label['labels'][0] for label in res]
test_df.head()
```

[5]:

	text	labels	zs_labels
0	@realDonaldTrump I like Mexicans who come to ...	1	supports
1	RT @AhmedtheBanker: Let's not forget @realDona...	0	opposes
2	@realDonaldTrump did not apply to immigrants o...	0	opposes
3	Been slacking on my @realDonaldTrump retweets...	1	supports
4	And how many #latinos enemies you gained in 1...	0	opposes

Labels are returned as plain text, so we now recode them to binary labels to evaluate classification performance.

[6]:
```
test_df['zs_labels'].replace(regex = {r'supports':1, r'opposes':0, r'does not␣
↪express an opinion about': 0}, inplace = True)
```

[7]:
```
matthews_corrcoef(test_df['labels'], test_df['zs_labels'])
```

[7]:
```
0.6270303800055436
```