ex9-naive-bayes-classifier

August 5, 2024

[1]:
```
import numpy as np
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
```

[2]:
```
# We defined the categories which we want to classify
categories = ["rec.motorcycles", "sci.electronics", "comp.graphics", "sci.med"]
# sklearn provides us with subset data for training and testing
train_data = fetch_20newsgroups(
    subset="train", categories=categories, shuffle=True, random_state=42
)
print(train_data.target_names)
print("\n".join(train_data.data[0].split("\n")[:3]))
print(train_data.target_names[train_data.target[0]])
# Let's Look at categories of our first ten training data
for t in train_data.target[:10]:
    print(train_data.target_names[t])
```

```
['comp.graphics', 'rec.motorcycles', 'sci.electronics', 'sci.med']
From: kreyling@lds.loral.com (Ed Kreyling 6966)
Subject: Sun-os and 8bit ASCII graphics
Organization: Loral Data Systems
comp.graphics
comp.graphics
comp.graphics
rec.motorcycles
comp.graphics
sci.med
sci.electronics
sci.electronics
comp.graphics
rec.motorcycles
sci.electronics
```

[3]:
```
# Builds a dictionary of features and transforms documents to feature vectors␣
↪anc
# matrix of token counts (CountVectorizer)
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(train_data.data)
# transform a count matrix to a normalized tf-idf representation
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
```

[4]:
```
# training our classifier ; train_data.target will be having numbers assigned␣
↪for
clf = MultinomialNB().fit(X_train_tfidf, train_data.target)
# Input Data to predict their classes of the given categories
docs_new = ["I have a Harley Davidson and Yamaha.", "I have a GTX 1050 GPU"]
# building up feature vector of our input
X_new_counts = count_vect.transform(docs_new)
# We call transform instead of fit_transform because it's already been fit
X_new_tfidf = tfidf_transformer.transform(X_new_counts)
```

[5]:
```
# predicting the category of our input text: Will give out number for category
predicted = clf.predict(X_new_tfidf)
for doc, category in zip(docs_new, predicted):
    print("%r => %s" % (doc, train_data.target_names[category]))
```

```
'I have a Harley Davidson and Yamaha.' => rec.motorcycles
'I have a GTX 1050 GPU' => sci.med
```

[6]:
```
text_clf = Pipeline(
    [
        ("vect", CountVectorizer()),
        ("tfidf", TfidfTransformer()),
        ("clf", MultinomialNB()),
    ]
)

# Fitting our train data to the pipeline
text_clf.fit(train_data.data, train_data.target)

# Test data
test_data = fetch_20newsgroups(
    subset="test", categories=categories, shuffle=True, random_state=42
)
docs_test = test_data.data

# Predicting our test data
predicted = text_clf.predict(docs_test)
```

```
print("We got an accuracy of", np.mean(predicted == test_data.target) * 100,␣
  ↪"%")
```

```
We got an accuracy of 91.49746192893402 %
```