sec01

September 12, 2023

1 Naive Bayes for text classification

“I liked the movie” positive “It’s a good movie. Nice story” positive “Nice songs. But sadly boring ending.” negative “Hero’s acting is bad but heroine looks good. Overall nice movie” positive “Sad, boring movie” negative

1.1 Basic Naive Bayes

[ ]:
```
train_sentence_list = [
    "I liked the movie",
    "It's a good movie. Nice story",
    "Nice songs. But sadly boring ending.",
    "Hero's acting is bad but heroine looks good. Overall nice movie",
    "Sad, boring movie"
]
train_label_list = [1, 1, 0, 1, 0]

test_sentence_list = [
    "I loved the acting in the movie",
    "The movie was bad",
    "Sad"
]
test_label_list = [1, 0, 0]
```

[ ]:
```
word_list = " ".join(train_sentence_list).split()
```

[ ]:
```
word_list
```

[ ]:
```
['I',
 'liked',
 'the',
 'movie',
 "It's",
 'a',
 'good',
 'movie.',
 'Nice',
 'story',
```

```
 'Nice',
 'songs.',
 'But',
 'sadly',
 'boring',
 'ending.',
 "Hero's",
 'acting',
 'is',
 'bad',
 'but',
 'heroine',
 'looks',
 'good.',
 'Overall',
 'nice',
 'movie',
 'Sad,',
 'boring',
 'movie']
```

2 Pre-processing

[ ]:
```
import string
import nltk
from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
```

[ ]:
```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/xiangpan/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[ ]:
```
True
```

[ ]:
```
def preprocess(sentence):
    # Remove punctuations
    sentence = sentence.translate(str.maketrans('', '', string.punctuation))
    # Convert words to lower case and split them
    sentence = sentence.lower().split()
    # Remove stop words
    stop_words = set(stopwords.words("english"))
    sentence = [word for word in sentence if word not in stop_words]
    # stem words
    porter = PorterStemmer()
    sentence = [porter.stem(word) for word in sentence]
```

```
    return sentence
```

[ ]:
```
postive_words = []
negative_words = []
for sentence, label in zip(train_sentence_list, train_label_list):
    sentence = preprocess(sentence)
    if label == 0:
        negative_words += sentence
    else:
        postive_words += sentence
postive_words = list(postive_words)
negative_words = list(negative_words)
print(len(postive_words), len(negative_words))
```

```
15 8
```

Note here we removed the stop words and stemd, so the calculation is different from the slides.

[ ]:
```
postive_words
```

[ ]:
```
['good',
 'act',
 'like',
 'movi',
 'look',
 'hero',
 'heroin',
 'bad',
 'overal',
 'stori',
 'nice']
```

[ ]:
```
negative_words
```

[ ]:
```
['end', 'bore', 'movi', 'sad', 'song', 'nice', 'sadli']
```

2.1 Calculate the prior probability

[ ]:
```
positive_prior = len([label for label in train_label_list if label == 1]) /␣
 ↪len(train_label_list)
negative_prior = len([label for label in train_label_list if label == 0]) /␣
 ↪len(train_label_list)
```

[ ]:
```
positive_prior
```

[ ]:
```
0.6
```

[ ]:
```
negative_prior
```

[ ]: 0.4

2.2 Calculate the P(W|C)

[ ]:
```
def get_conditional_word_likelihood(word, label):
    if label == 0:
        return negative_words.count(word) / len(negative_words)
    else:
        return postive_words.count(word) / len(postive_words)
```

[ ]:
```
def get_conditional_word_likelihood_with_smoothing(word, label):
    if label == 0:
        return (negative_words.count(word) + 1) / (len(negative_words) +␣
 ↪len(word_list))
    else:
        return (postive_words.count(word) + 1) / (len(postive_words) +␣
 ↪len(word_list))
```

[ ]:
```
def get_sentence_likelihood(sentence, label):
    sentence = preprocess(sentence)
    likelihood = 1
    for word in sentence:
        likelihood *= get_conditional_word_likelihood_with_smoothing(word,␣
 ↪label)
    return likelihood
```

3 Do inference

[ ]:
```
# do inference
for test_sentence in test_sentence_list:
    pos_prob = positive_prior * get_sentence_likelihood(test_sentence, 1)
    neg_prob = negative_prior * get_sentence_likelihood(test_sentence, 0)
    print(pos_prob, neg_prob)
    if pos_prob > neg_prob:
        print("Positive")
    elif pos_prob < neg_prob:
        print("Negative")
    else:
        print("Neutral!")
```

```
5.2674897119341567e-05 1.4579384749963548e-05
Positive
0.0023703703703703703 0.0005540166204986149
Positive
0.013333333333333334 0.021052631578947368
Negative
```

4 Tokenization

[ ]:
```
import nltk
nltk.download('punkt')

from nltk.tokenize import word_tokenize
word_tokenize("Hello World!")
```

```
[nltk_data] Downloading package punkt to /Users/xiangpan/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
```

[ ]:
```
['Hello', 'World', '!']
```

4.1 Stemming with NLTK

[ ]:
```
# import these modules
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

ps = PorterStemmer()

# choose some words to be stemmed
words = ["program", "programs", "programmer", "programming", "programmers"]

for w in words:
    print(w, " : ", ps.stem(w))
```

```
program  :  program
programs  :  program
programmer  :  programm
programming  :  program
programmers  :  programm
```

[ ]:
```
# import these modules
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

ps = PorterStemmer()

# choose some words to be stemmed
words = ["sad", "sadly"]

for w in words:
    print(w, " : ", ps.stem(w))
```

```
sad  :  sad
sadly  :  sadli
```

[ ]:
```
# remove punctuation
import string
string.punctuation
```

[ ]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

5 Pipeline

5.1 Words Counting as Features

[ ]:
```
from collections import defaultdict
from nltk import word_tokenize
```

[ ]:
```
train_feature_list
```

[ ]:
```
[(defaultdict(int, {'like': 1, 'movi': 1}), 1),
 (defaultdict(int, {'good': 1, 'movi': 1, 'nice': 1, 'stori': 1}), 1),
 (defaultdict(int, {'nice': 1, 'song': 1, 'sadli': 1, 'bore': 1, 'end': 1}),
  0),
 (defaultdict(int,
             {'hero': 1,
              'act': 1,
              'bad': 1,
              'heroin': 1,
              'look': 1,
              'good': 1,
              'overal': 1,
              'nice': 1,
              'movi': 1}),
  1),
 (defaultdict(int, {'sad': 1, 'bore': 1, 'movi': 1}), 0)]
```

[ ]:
```
# NaiveBayesClassifier
train_feature_list = []
for sentence, label in zip(train_sentence_list, train_label_list):
    sentence = preprocess(sentence)
    feature_dict = defaultdict(int)
    for word in sentence:
        feature_dict[word] += 1
    train_feature_list.append((feature_dict, label))

nb_classifier = nltk.NaiveBayesClassifier.train(train_feature_list)
for test_sentence in test_sentence_list:
    test_sentence = preprocess(test_sentence)
    feature_dict = defaultdict(int)
    for word in test_sentence:
        feature_dict[word] += 1
```

```
    print(nb_classifier.classify(feature_dict))
```

```
1
1
0
```

[ ]:
```
# TF-IDF as features
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
tfidf_vectorizer.fit(train_sentence_list)
train_feature_list = tfidf_vectorizer.transform(train_sentence_list)
# LogisticRegression
from sklearn.linear_model import LogisticRegression
lr_classifier = LogisticRegression()
lr_classifier.fit(train_feature_list, train_label_list)
test_feature_list = tfidf_vectorizer.transform(test_sentence_list)
lr_classifier.predict(test_feature_list)
```

[ ]:
```
array([1, 1, 1])
```

TF-IDF are better features for document classification