Lenning_JM4

September 23, 2021

0.1 1

Assume the following likelihoods for each word being part of a positive or negative movie review, and equal prior probabilities for each class.

pos neg

I 0.09 0.16 always 0.07 0.06 like 0.29 0.06 foreign 0.04 0.15 films 0.08 0.11

What class will Naive bayes assign to the sentence "I always like foreign films."?

P("I always like foreign films"|+) = $0.09 \times 0.07 \times 0.29 \times 0.04 \times 0.08 = 0.0000058464$ P("I always like foreign films"|-) = $0.16 \times 0.06 \times 0.06 \times 0.15 \times 0.11 = 0.0000095040$

Assigned negative

0.2 2

Given the following short movie reviews, each labeled with a genre, either comedy or action:

1. fun, couple, love, love- comedy
2. fast, furious, shoot- action
3. couple, fly, fast, fun, fun- comedy
4. furious, shoot, shoot, fun- action
5. fly, fast, shoot, love- action

and a new document D:

fast, couple, shoot, fly

compute the most likely class for D. Assume a naive Bayes classifier and use add-1 smoothing for the likelihoods.

[ ]:
```
from collections import Counter
import numpy as np

docs = [['fun', 'couple', 'love', 'love'],['fast', 'furious',␣
↪'shoot'],['couple' , 'fly', 'fast', 'fun', 'fun'],['furious', 'shoot',␣
↪'shoot', 'fun'], ['fly', 'fast', 'shoot', 'love']]
genres = ['comedy', 'action', 'comedy', 'action', 'action']
D = list(zip(docs, genres))
C = Counter(elem[1] for elem in D)
```

```
def train_naive_bayes(D, C, binary=False):
        """trains a naive bayes model on list of documents

        Args:
            D (list of lists): list of documents to train on
            C (Counter/dictionary): dictionary of key = class, and value =␣
 ↪occurences of class
            binary (bool, optional): whether to run binarized Naive Bayes.␣
 ↪Defaults to False.

        Returns:
            dict, dict, list: returns the logprior for the classes, the␣
 ↪loglikelihood of each word given a class, and the vocab
        """
        logprior = dict()
        bigdoc = dict()
        loglikelihood = dict()

        for c in C: # Calculate P(c) terms
                n_doc = len(D)
                n_c = C[c]
                logprior[c] = np.log(n_c / n_doc)
                vocab = [item for sublist in docs for item in sublist]

                bigdoc[c] = []
                for d in D:
                        if d[1] == c:
                                if binary:
                                        bigdoc[c].extend(np.unique(np.
 ↪array(d[0])))
                                else:
                                        bigdoc[c].extend(d[0])

                counts = Counter(bigdoc[c])

                for word in vocab: # Calculate P(w|c) terms
                        count_w_c = counts[c]
                        if word not in loglikelihood.keys():
                                loglikelihood[word] = dict() # creating new␣
 ↪dict entirely for each class
                        loglikelihood[word][c] = np.log((count_w_c + 1) /␣
 ↪(len(bigdoc[c]) + len(vocab)))
```

```python
        return logprior, loglikelihood, vocab


def test_naive_bayes(testdoc, logprior, loglikelihood, C, V):
        """predicts class of testdoc given Naive Bayes trained data

        Args:
            testdoc (list): list of words
            logprior (dict): the logprior for classes
            loglikelihood (dict): the loglikelihood of each word given a class
            C (Counter/dict): list of classes and counts for each class
            V (list): vocabulary

        Returns:
            [str]: the predicted class
        """
        tot = dict()
        for c in C:
                tot[c] = logprior[c]
                for word in testdoc:
                        if word in V:
                                tot[c] = tot[c] + loglikelihood[word][c]
        return max(tot, key=tot.get)


logprior, loglikelihood, vocab = train_naive_bayes(D, C)
class_pred = test_naive_bayes(['fast', 'couple', 'shoot', 'fly'], logprior,␣
 ↪loglikelihood, C, vocab)
print(class_pred)
```

```
action
```

0.3 3

Train two models, multinomial naive Bayes and binarized naive Bayes, both with add-1 smoothing, on the following document counts for key sentiment words, with positive or negative class assigned as noted.

doc "good" "poor" "great" (class) d1. 3 0 3 pos d2. 0 1 2 pos d3. 1 3 0 neg d4. 1 5 2 neg d5. 0 2 0 neg

Use both naive Bayes models to assign a class (pos or neg) to this sentence:

A good, good plot and great characters, but poor acting.

Recall from page 6 that with naive Bayes text classification, we simply ignore (throw out) any word that never occurred in the training document. (We don't throw out words that appear in some classes but not others; that's what add-one smoothing is for.) Do the two models agree or disagree?

[ ]:

```python
# Create training documents
d1 = ['good']*3
d1.extend(['great']*3)
d2 = ['poor', 'great', 'great']
d3 = ['good']
d3.extend(['poor']*3)
d4 = ['good']
d4.extend(['poor']*5)
d4.extend(['great']*2)
d5 = ['poor', 'poor']

# Create input data for training algorithm
docs = [d1, d2, d3, d4, d5]
classes = ['pos', 'pos', 'neg', 'neg', 'neg']
D = list(zip(docs, classes))
C = Counter(elem[1] for elem in D)

# Run Multinomial Naive Bayes on testdoc
multi_logprior, multi_loglikelihood, multi_vocab = train_naive_bayes(D, C)
multi_class_pred = test_naive_bayes('A good good plot and great characters but␣
↪poor acting'.split(sep=' '), multi_logprior, multi_loglikelihood, C,␣
↪multi_vocab)
print("multinomial prediction:", multi_class_pred)

# Run Binarized Naive Bayes on testdoc
bin_logprior, bin_loglikelihood, bin_vocab = train_naive_bayes(D, C,␣
↪binary=True)
bin_class_pred = test_naive_bayes('A good good plot and great characters but␣
↪poor acting'.split(sep=' '), bin_logprior, bin_loglikelihood, C, bin_vocab)
print("binary prediction:", bin_class_pred)
```

```
multinomial prediction: pos
binary prediction: neg
```