NLP Homework 2

Álvaro Faúndez

November 2021

Part I

Assume that you have trained a Naïve Bayes classifier for the task of sentiment classification (please refer to Chapter 4 in the J&M book). The classifier uses only bag-of-word falwaysures. Assume the following parameters for each word being part of a positive or negative movie review, and the prior probabilities are 0.4 for the positive class and 0.6 for the negative 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

Question: What class will Naïve Bayes assign to the sentence "I always like foreign films"? **Show your work**.

Answer

Bag of words {`I`, `always`, `like`, `foreign`, `films`}

Features $\vec{x} = [1, 1, 1, 1, 1, 1]$

Priors:

$$\begin{aligned} P_{prior}(C_{\texttt{pos}}) &= 0.4 \\ P_{prior}(C_{\texttt{neg}}) &= 0.6 \end{aligned} \tag{1}$$

$$\begin{aligned} P(C_{\texttt{pos}} \mid \vec{x}) &\propto P(C_{\texttt{pos}}) \times P(\vec{x} \mid C_{\texttt{pos}}) \\ &\propto P(\texttt{I} \mid C_{\texttt{pos}}) \times [P(\texttt{always} \mid C_{\texttt{pos}}) \times P(\texttt{like} \mid C_{\texttt{pos}}) \times \\ &\quad P(\texttt{foreign} \mid C_{\texttt{pos}}) \times P(\texttt{films} \mid C_{\texttt{pos}})] \\ &\propto 0.4 \times [0.09 \times 0.07 \times 0.29 \times 0.04 \times 0.08] \\ &\propto 0.00000233856 \end{aligned} \tag{2}$$

$$\begin{aligned} P(C_{\texttt{neg}} \mid \vec{x}) &\propto P(C_{\texttt{neg}}) \times P(\vec{x} \mid C_{\texttt{neg}}) \\ &\propto P(\texttt{I} \mid C_{\texttt{neg}}) \times [P(\texttt{always} \mid C_{\texttt{neg}}) \times P(\texttt{like} \mid C_{\texttt{neg}}) \times \\ &\quad P(\texttt{foreign} \mid C_{\texttt{neg}}) \times P(\texttt{films} \mid C_{\texttt{neg}})] \\ &\propto 0.6 \times [0.16 \times 0.06 \times 0.06 \times 0.15 \times 0.11] \\ &\propto 0.0000057024 \end{aligned} \tag{3}$$

Since $P(C_{\texttt{neg}} \mid \vec{x}) > P(C_{\texttt{pos}} \mid \vec{x})$, the class assigned is $C_{\texttt{neg}}$.

Part II

[Implementing the Naïve Bayes classifier for movie review classification – 90 points] In this assignment, you will write 2 scripts: NB.py and pre-process.py. NB.py should take the following parameters: the training file, the test file, the file where the parameters of the resulting model will be saved, and the output file where you will write predictions made by the classifier on the test data (one example per line). The last line in the output file should list the overall accuracy of the classifier on the test data. The training and the test files should have the following format: one example per line; each line corresponds to an example; first column is the label, and the other columns are feature values.

pre-process.py should take the training (or test) directory containing movie reviews, should perform pre-processing1 on each file and output the files in the vector format to be used by NB.py.

a Implement in Python a Naïve Bayes classifier with bag-of-word (BOW) features and Add-one smoothing. Note: Do not use smoothing for the prior parameters. You should implement the algorithm from scratch and should not use off-the-shelf software. [35 points]

b Use the following small corpus of movie reviews to train your classifier. Save the parameters of your model in a file called movie-review-small.NB (you can man- ually convert this small corpus into the vector format, so that you can run NB.py on it). [10 points]

i. fun, couple, love, love **comedy**

ii. fast, furious, shoot **action**

iii. couple, fly, fast, fun, fun **comedy**

iv. furious, shoot, shoot, fun **action**

v. fly, fast, shoot, love **action**

c Test you classifier on the new document below: {fast,couple,shoot,fly}. Compute the most likely class. Report the probabilities for each class. [5 points]

d Now use the movie review dataset provided with this homework to train a Naive Bayes classifier for the real task. You will train your classifier on the training data and will test it on the test data. The dataset contains movie reviews; each review is saved as a separate file in the folder “neg” or “pos” (which are located in “train” and “test” folders, respectively). You should use these raw files and represent each review using a vector of bag-of-word features, where each feature corresponds to a word from the vocabulary file (also provided), and the value of the feature is the count of that word in the review file.

Pre-processing: prior to building feature vectors, you should separate punctuation from words and lowercase the words in the reviews. You will train NB classifier on the training partition using the BOW features (use add-one smoothing, as we did in class). You will evaluate your classifier on the test partition. In addition to BOW features, you should experiment with additional features. In that case, please provide a description of the features in your report. Save the parameters of your BOW model in a file called movie-review-BOW.NB. Report the accuracy of your program on the test data with BOW features. Investigate your results. For the reviews for which your program made incorrect predictions, were there any trends that you observed? That is, can you explain why these incorrect predictions were made? [40 points]

Answer

The Naïve Bayes classifier

The implementation of the Naive Bayes classifier is done class Model. It stores the frequencies of the training words and calculates the probabilities of the words given to predict.

To predict, the following steps are performed:

1. Create a vocabulary and the classifications using the Encoder class. This class picks up a set of words, stores them, and assigns an index to each word, starting in 0.
2. Create the model, providing a vocabulary and the classifications. This will create:
    - A matrix of frequencies of token events by classification initilized with None values, indexed by the vocabulary and the classifications encoders.
    - A vector of priors probabilities, initialized with zeroes, indexed by the classifications encoder.
    - A matrix of likelihoods probabilites initialized with zeroes, indexed by the vocabulary and the classifications encoders.

    Other structures are also created, such as classification events and total events, but they were mostly created to avoid extra computations.
3. Train the model, providing a labeled training corpus. This process updates the stored frequencies.
4. Predict the class of a new document or corpus. The document has the following formats:
    - A dictionary of frequencies, recongized as a single document.
    - A string text, recognized as a single document and transformed to a dictionary of frequencies.
    - A list, recognized as a list of documents. Each document could be a dictionary of frequencies or a string text.

    Each time a prediction is requested for a document or corpus, the prior and likelihoods probabilities of the document's tokens are looked up in the prior and likelihoods matrices. If they are not present, they are calculated and stored

As requested, a preprocess script has been implemented. Given a vocabulary, a document goes through two preprocessing steps.

First, the text as string goes through the following steps:

- URL removal
- Email removal
- HTML un-escaping characters
- HTML rendering
- Unicode characters removal
- Lowercase

Second, the text is tokenized is split into tokens, and goes through the following steps:

- If the word is in the vocabulary, it is added to the frequencies
- I the word is not in the vocabulary, the word goes through the following steps:
  - Remove punctuation, except dashes
  - Remove dashes that are not linking two words
  - Split the word into tokens
  - For each token, add to the frequencies if it is in the vocabulary

Training with the small corpus

There are two possible classifications for each document: **action** and **comedy**.

```
labeler = Encoder (['action', 'comedy'])
# Encoder (tokens=2, sample=['action', 'comedy'])
```

The vocabulary for this example consists in 7 words: **couple**, **furious**, **fun**, **fly**, **fast**, **shoot**, and **love**.

This vocabulary is stored in the file `homework-2/movie-review-small/aclImdb/imdb.vocab`.

```
vocabulary = Encoder.open('movie-review-small/aclImdb/imdb.vocab')
# Encoder(tokens=7, sample=['couple', 'love', 'fast', 'shoot', 'furious', 'fly
                                        ', 'fun'])
```

The training corpus is composed of 5 documents, stored at:

- `homework-2/movie-review-small/aclImdb/train/action/`
- `homework-2/movie-review-small/aclImdb/train/comedy/`

```
train_corpus = Corpus.open('movie-review-small/aclImdb/train/**/*.txt',
                                        vocabulary=vocabulary, verbose=True)
# Corpus(documents=5, tokens=7, words=20)
```

Now, it's possible to save the corpus frequencies into a file, storing the frequencies of each document and the classification as a JSON document per line:

```
train_corpus.write('movie-review-small/aclImdb/train.NB', verbose=True)
```

This is the content of `movie-review-small/aclImdb/train.NB`:

```
{"frequencies": {"fun": 1, "couple": 1, "love": 2}, "label": "comedy"}
{"frequencies": {"couple": 1, "fly": 1, "fast": 1, "fun": 2}, "label": "comedy"}
{"frequencies": {"fast": 1, "furious": 1, "shoot": 1}, "label": "action"}
{"frequencies": {"fly": 1, "fast": 1, "shoot": 1, "love": 1}, "label": "action"}
{"frequencies": {"furious": 1, "shoot": 2, "fun": 1}, "label": "action"}
```

Now, the model is trained using the training corpus:

```
model = Model(vocabulary, labeler, log=True)
model.fit(train_corpus, train_corpus.labels())
model.summary()
```

The summary prints debugging information, in this case, the matrices and vectors stored within the model:

- Events by label:

```
c(C)   | value
-------+------
action |     3
comedy |     2
```

- Events by token and label:

```
c(t,C) | couple | love | fast | shoot | furious | fly | fun
-------+--------+------+------+-------+---------+-----+----
action |      0 |    1 |    2 |     4 |       2 |   1 |   1
comedy |      2 |    2 |    1 |     0 |       0 |   1 |   3
```

- Prior probabilities:

```
p(C)   | value
-------+------
action |  None
comedy |  None
```

- Likelihoods probabilities:

```
P(t|C) | couple | love | fast | shoot | furious |  fly |  fun
-------+--------+------+------+-------+---------+------+-----
action |   None | None | None |  None |    None | None | None
comedy |   None | None | None |  None |    None | None | None
```

Predicting with the small corpus

Now, predictions can be made for a new document:

```
result = model.predict(Document('fast,couple,shoot,fly'), debug=True)
```

When debugging, it outputs the posterior probabilites for each class:

```
p(C|d) |                  value
-------+-----------------------
action | 0.00017146776406035664
comedy |  7.324218750000001e-05
```

And the output is 0, that decoded means **action**:

```
print(labeler.decode(prediction))
# action
```

Finally, the model internal probabilities must have changed:

- Prior probabilities:

```
p(C)   | value
-------+------
action |   0.6
comedy |   0.4
```

- Likelihoods probabilities:

```
P(t|C) |   couple | love |    fast |    shoot | furious |     fly |  fun
-------+----------+------+---------+----------+---------+---------+-----
action | 0.055... | None | 0.16... | 0.277... |    None | 0.11... | None
comedy |   0.1875 | None |   0.125 |   0.0625 |    None |   0.125 | None
```

There are still None values, but that's normal because those words didn't need a prediction yet.

Testing and predicting with the IMDB reviews corpus

The process is exactly the same as before, but the corpus is now the IMDB reviews corpus. Internally, one big changed was made: the internal parameters are now being calculated using $\log_2$ probabilities, because preliminar trials showed accuracies no more than .51% and improved instantly after the change.

There are two possible classifications for each document: **pos** and **neg**.

```
labeler = Encoder(['pos', 'neg'])
# Encoder(tokens=2, sample=['pos', 'neg'])
```

And the vocabulary consists of 89,527 tokens:

```
vocabulary = Encoder.open('movie-review-HW2/aclImdb/imdb.vocab')
# Encoder(tokens=89527, sample=['redux', 'jaubert', 'pantangeli', 'overwatched
                                ', 'braun', 'abstractions'])
```

The training and testing copurs contain 25,000 documents each:

```
    train_corpus = Corpus.open('movie-review-HW2/aclImdb/train/**/*.txt',
    vocabulary=vocabulary,
    verbose=True
)
print(train_corpus)
train_corpus.write('movie-review-HW2/aclImdb/train-BOW.NB', verbose=True)
# Corpus(documents=25000, tokens=87884, words=5871238))
```

```
    test_corpus = Corpus.open('movie-review-HW2/aclImdb/test/**/*.txt',
    vocabulary=vocabulary,
    verbose=True
)
print(test_corpus)
test_corpus.write('movie-review-HW2/aclImdb/test-BOW.NB', verbose=True)
# Corpus(documents=25000, tokens=77976, words=5750635))
```

This part of the process takes about 2 3 minutes, then it's more efficient to load the stored `.NB` files, especially while debugging the predictions.

```
train_corpus = Corpus.open('movie-review-HW2/aclImdb/train-BOW.NB', frequencies
                                =True, verbose=True)
test_corpus = Corpus.open('movie-review-HW2/aclImdb/test-BOW.NB', frequencies=
                                True, verbose=True)
```

Predicting the test corpus labels takes no more that 8 10 seconds:

```
model = Model(vocabulary, labeler, log=True)
model.fit(train_corpus, train_corpus.labels(), verbose=True)
```

A Metrics module is provided to calculate the accuracy and the confusion matrix:

```
predictions = model.predict(test_corpus, verbose=True, debug=False)
score = Metrics.score(test_corpus.labels(), labeler.decode(predictions),
                                            labeler)
print(score)
# {'accuracy': 0.81464, 'confusion': [[9325, 3175], [1459, 11041]]}
```

Analizing the results

The confusion matrix obtained goes as follows:

```
true\predicted |  pos |   neg
---------------+------+------
           pos | 9343 |  3157
           neg | 1531 | 10969
```

There are 3157 false negatives and 1531 false positives.

False negative examples:

- This example is positive, but include lots of negations words, that may impact the prediction:

  `movie-review-HW2/aclImdb/test/pos/2823_10.txt`

  > I really like this show. It has drama, romance, and comedy all rolled into one. I am 28 and I am a married mother, so I can identify both with Lorelei's and Rory's experiences in the show. I have been watching mostly the repeats on the Family Channel lately, so I am not up-to-date on what is going on now. I think females would like this show more than males, but I know some men out there would enjoy it! I really like that is an hour long and not a half hour, as th hour seems to fly by when I am watching it! Give it a chance if you have never seen the show! I think Lorelei and Luke are my favorite characters on the show though, mainly because of the way they are with one another. How could you not see something was there (or take that long to see it I guess I should say)?

Happy viewing!

- This review is positive, but it describes the movie's plot, which is about negative concepts:

  `movie-review-HW2/aclImdb/test/pos/2823_10.txt`

  > This movie makes you think. It shows how a woman's weaknesses can result in nightmares for others. Her physically aggressive behavior is more often seen in men than women, so it made me feel even more uncomfortable to see the way the lead actress behaved. I think that women might think about this behavior, but I don't think they act on it. The dark scenes added to the sense of evil that needed to be

> hidden. I was relieved when the prisoners escaped. I was hopeful that the end would bring a satisfying solution, but it did not. Maybe that is more realistic. Life seems to run in the same direction instead of creating a new river bed running up hill.

False positive examples:

- This example is negative, but uses sarcasm to express it:

  `movie-review-HW2/aclImdb/test/neg/240_4.txt`

  > There must be an error. This movie belongs with "Plan 9", and a lot others as a quite entertaining, silly diversion. Youĺl never accept you like it, yet you will watch it whenever it comes out on TV. Itś as simple as that.

- This review is negative, but talks in positive terms about the movie's starring actors:

  1420894`movie-review-HW2/aclImdb/test/neg/1821_4.txt`

  > Alan Rickman & Emma Thompson give good performances with southern/New Orleans accents in this detective flick. It's worth seeing for their scenes- and Rickman's scene with Hal Holbrook. These three actors mannage to entertain us no matter what the movie, it seems. The plot for the movie shows potential, but one gets the impression in watching the film that it was not pulled off as well as it could have been. The fact that it is cluttered by a rather uninteresting subplot and mostly uninteresting kidnappers really muddles things. The movie is worth a view- if for nothing more than entertaining performances by Rickman, Thompson, and Holbrook.

Experimenting with bigrams

The implementation of bigrams differs from what has been done in the previous section only in the frequency counting. After processing a document and splitting it into tokens, the frequency of each bigram is calculated by iterating the tokens in pairs. This will generate a new vocabulary.

Now, the corpora must be loaded with anextra "ngrams" parameter:

```
train_corpus = Corpus.open('movie-review-HW2/aclImdb/train/**/*.txt',
    ngrams=2,
    vocabulary=vocabulary,
    verbose=True,
```

```
)
# Corpus(documents=25000, tokens=1420894, words=5896238))
```

The number of tokens now ascends to 1420894, about 16 times the amount of tokens in the previous using unigram.

Using bigrams, also there is need to update the vocabulary for the model. This way, the training and fiotting goes:

```
vocabulary = Encoder(list(train_corpus.frequencies.keys()))
model = Model(vocabulary, labeler, log=True)
model.fit(train_corpus, train_corpus.labels(), verbose=True)

predictions = model.predict(test_corpus)
score = Metrics.score(test_corpus.labels(), labeler.decode(predictions),
                                            labeler)
print(score)
# {'accuracy': 0.87928, 'confusion': [[11532, 968], [2050, 10450]]}
```

The accuracy is now 0.87928, higher than the unigram accuracy.

Experimenting with different preprocessing

The following extra preprocessing steps are available:

- expand acronyms
- replace emoticons
- replace negative/positive words
- replace negations
- remove stopwords

Different combinations of pre=processing were tried, but the only improvement was achieved with using the unigram model and removing stopwords, increasing the accuracy slightly from 0.8236 to 0.8236. Results are in `/movie-review-HW2/score*`.

How to run the code

Pre-process

Two scripts are provided.

To pre-process and store the frequencies in a ṄB file:

```
python3 pre-process.py \
    --training-file='movie-review-HW2/aclImdb/train/**/*.txt' \
    --test-file='movie-review-HW2/aclImdb/test/**/*.txt' \
    --output-path=movie-review-HW2/aclImdb \
    --vocabulary-file=movie-review-HW2/aclImdb/imdb.vocab \
    --add-label=pos --add-label=neg \
    --ngrams=1
```

That will generate the files `train-1grams.NB` and `test-1grams.NB` in the selected output path `movie-review-HW2/aclImdb`.

Extra pre-processing steps can be added with the `-add-pre-process` flag:

```
    python3 pre-process.py \
    --training-file='movie-review-HW2/aclImdb/train/**/*.txt' \
    --test-file='movie-review-HW2/aclImdb/test/**/*.txt' \
    --output-path=movie-review-HW2/aclImdb \
    --vocabulary-file=movie-review-HW2/aclImdb/imdb.vocab \
    --add-label=pos --add-label=neg \
    --ngrams=1 \
    --add-pre-process=stopwords \
    --add-pre-process=acronyms
```

To predict using the previousliy created files:

```
python3 NB.py \
    --training-file='movie-review-HW2/aclImdb/train-1grams.NB' \
    --test-file='movie-review-HW2/aclImdb/test-1grams.NB' \
    --output-file='movie-review-HW2/aclImdb/score-1grams.txt' \
    --vocabulary-file='movie-review-HW2/aclImdb/imdb.vocab' \
    --use-train-vocabulary
```

It will generate an output file `movie-review-HW2/aclImdb/score-1grams.txt` with the accuracy, confusion matrix, and debug information.

In the case of using bigrams, these are the commands:

```
python3 pre-process.py \
    --training-file='movie-review-HW2/aclImdb/train/**/*.txt' \
    --test-file='movie-review-HW2/aclImdb/test/**/*.txt' \
    --output-path=movie-review-HW2/aclImdb \
    --vocabulary-file=movie-review-HW2/aclImdb/imdb.vocab \
    --add-label=pos --add-label=neg \
    --ngrams=2
```

```
python3 pre-process.py \
    --training-file='movie-review-HW2/aclImdb/train/**/*.txt' \
    --test-file='movie-review-HW2/aclImdb/test/**/*.txt' \
    --output-path=movie-review-HW2/aclImdb \
    --vocabulary-file=movie-review-HW2/aclImdb/imdb.vocab \
    --add-label=pos --add-label=neg \
    --ngrams=2
```

Tested with Python 3.7.12 and 3.10.0. No extra libraries needed.