Naïve Bayes Text Categorizer

Jonathan Lam

March 19, 2021

The code is available on GitHub at jlam55555/naive-bayes-text-categorization.

1 Build instructions

The build instructions can be found on the README on GitHub.

2 About the algorithm

2.1 Tokenization

Tokenization is first performed using nltk's `word_tokenizer`. No filtering (e.g., removing stop words or punctuation) is performed on tokens.

2.2 General algorithm

Naïve Bayes works by assuming that each word is conditionally independent of each other word given its class, which allows us to express the probability of a document D belonging to class C as:

$$P(C \mid D) = P(C) \prod_i P(w_i \mid C)$$

where $w_i \in D$ is a token in D. We can replace the product by a sum if we take the logarithm of the probabilities, and take the argmax to find the most probable class, i.e.:

$$\hat{C}_D = \operatorname*{argmax}_C \log P(C) \sum_i \log P(w_i \mid C)$$

2.3 General implementation

2.3.1 Attempt 1

A fairly basic Naïve Bayes algorithm is used for training and categorization. First, a list of all of the vocabulary words and categories is collected. A numpy matrix of size $V \times C$ is created to store term-class counts (to calculate class-conditional probabilities), where V is the size of the vocabulary and C is the number of classes. Another numpy vector of size $1 \times C$ is created to store the document frequency for each class (to calculate class priors).

During training, term and document counts are aggregated in the two matrices described above. When training is complete, the $V \times C$ matrix is converted to log-class-conditional probabilities by dividing the term counts by the total term frequency of the class. I.e. for a class C and term w_i:

$$P(w_i \mid C) = \frac{P(w_i \cap C)}{P(C)} = \frac{P(w_i \cap C)}{\sum_j P(w_j \cap C)} = \frac{\text{count}(w_i \cap C)}{\sum_j \text{count}(w_j \cap C)}$$

The log-priors are calculated by taking the logarithm of the frequency of documents having class C:

$$P(C) = \frac{\text{count(documents in class } C)}{\text{count(documents)}}$$

And thus we have all the components we need for the Naïve Bayes calculation. Calculating sums, dividing by total counts, and taking the elementwise logarithm was easy using numpy's vectorized array operations.

When performing inference, words not in the training vocabulary are simply discarded.

Laplace smoothing was used to prevent zero probabilities. This was chosen because it is simple and is known to work fairly well. No more complicated smoothing schemes were attempted, and this is a room for future improvement. For the first attempt, the smoothing factor was 1 (i.e., a +1 smoothing).

2.3.2 Attempt 2

To improve the speed of the program, the algorithm switched to using dictionaries rather than a dense numpy array. This had a (qualitatively) small speedup on the algorithm. This in itself should not change the performance (accuracy) of the algorithm.

I experimented with the smoothing factor. It seemed that a smoothing factor between 0.05 to 0.07 worked best, empirically, although this could be refined further through some automated hyperparameter search (which was not attempted for lack of time).

I also experimented with lemmatization, stemming, and stopword removal. Removing stopwords (using `nltk.corpus.stopwords`) had no considerable effect on the accuracy, but lemmatization and stemming had a noticeable effect. Notably, stemming performed slightly better than lemmatization, and either the porter or the snowball stemmers seemed to work well.

2.3.3 Attempt 3

I tried a different formula for the conditional probabilities following the recommendation from lecture. This comes from a slightly different interpretation and reasoning behind the conditional probabilities:

$$P(w_i \mid C) = \frac{\text{count(documents in class } C \text{ containing } w_i)}{\text{count(documents in class } C)}$$

the new interpretation being that this was the chance that a document contains a word (at least once).

However, this method did not achieve better results. While the results were similar for corpora 1 and 2, the accuracy significantly dropped for corpus 3 for unknown reasons. (It strongly favors the USN category.) This method should offer higher accuracy, so it is unknown what is working correctly.

3 Performance

3.1 Evaluating performance

The file `train_test_split.py` partitions the training datasets into a training and validation (evaluation) dataset with a default 80/20 train/test split. The performance metrics are then calculated using the provided `analyze.pl` script.

The following results are shown using the provided train/test split for corpus 1, as well as typical random 80/20 train/test split results for corpus2 and corpus3 (these particular splits and results are stored on the GitHub repo as well).

3.2 Attempt 1

3.2.1 Corpus 1

```
Processing answer file...
Found 5 categories: Pol Cri Oth Dis Str
Processing prediction file...

381 CORRECT, 62 INCORRECT, RATIO = 0.860045146726862.

CONTINGENCY TABLE:
        Pol     Cri     Oth     Dis     Str     PREC
Pol     122     2       11      0       5       0.87
Cri     0       38      0       0       1       0.97
Oth     1       0       6       0       0       0.86
Dis     0       0       5       86      0       0.95
Str     21      10      3       3       129     0.78
RECALL  0.85    0.76    0.24    0.97    0.96

F_1(Pol) = 0.859154929577465
F_1(Cri) = 0.853932584269663
F_1(Oth) = 0.375
F_1(Dis) = 0.955555555555556
F_1(Str) = 0.857142857142857
```

3.2.2 Corpus 2

```
Processing answer file...
Found 2 categories: O I
Processing prediction file...

155 CORRECT, 24 INCORRECT, RATIO = 0.865921787709497.

CONTINGENCY TABLE:
        O       I       PREC
O       117     14      0.89
I       10      38      0.79
RECALL  0.92    0.73

F_1(O) = 0.906976744186046
F_1(I) = 0.76
```

3.2.3 Corpus 3

```
Processing answer file...
Found 6 categories: Sci Fin Wor USN Spo Ent
Processing prediction file...

150 CORRECT, 25 INCORRECT, RATIO = 0.857142857142857.

CONTINGENCY TABLE:
        Sci     Fin     Wor     USN     Spo     Ent     PREC
Sci     20      0       0       0       0       0       1.00
Fin     1       19      0       0       0       2       0.86
Wor     0       0       53      3       1       1       0.91
USN     5       2       3       43      3       4       0.72
Spo     0       0       0       0       14      0       1.00
Ent     0       0       0       0       0       1       1.00
RECALL  0.77    0.90    0.95    0.93    0.78    0.12

F_1(Sci) = 0.869565217391304
F_1(Fin) = 0.883720930232558
F_1(Wor) = 0.929824561403509
F_1(USN) = 0.811320754716981
F_1(Spo) = 0.875
F_1(Ent) = 0.222222222222222
```

3.3 Attempt 2

3.3.1 Corpus 1

```
Processing answer file...
Found 5 categories: Cri Dis Str Oth Pol
Processing prediction file...

393 CORRECT, 50 INCORRECT, RATIO = 0.887133182844244.

CONTINGENCY TABLE:
Cri     Dis     Str     Oth     Pol     PREC
Cri     40      0       2       1       1       0.91
Dis     0       89      1       3       0       0.96
Str     8       0       128     3       18      0.82
Oth     0       0       0       13      2       0.87
Pol     2       0       4       5       123     0.92
RECALL  0.80    1.00    0.95    0.52    0.85

F_1(Cri) = 0.851063829787234
F_1(Dis) = 0.978021978021978
F_1(Str) = 0.876712328767123
F_1(Oth) = 0.65
F_1(Pol) = 0.884892086330935
```

3.3.2 Corpus 2

```
Processing answer file...
Found 2 categories: I O
Processing prediction file...

159 CORRECT, 20 INCORRECT, RATIO = 0.888268156424581.

CONTINGENCY TABLE:
I       O       PREC
I       38      6       0.86
O       14      121     0.90
RECALL  0.73    0.95

F_1(I) = 0.791666666666667
F_1(O) = 0.923664122137405
```

3.3.3 Corpus 3

```
Processing answer file...
Found 6 categories: Fin Ent Sci USN Spo Wor
Processing prediction file...

157 CORRECT, 18 INCORRECT, RATIO = 0.897142857142857.

CONTINGENCY TABLE:
Fin     Ent     Sci     USN     Spo     Wor     PREC
Fin     20      2       3       0       0       0       0.80
Ent     0       4       0       0       0       0       1.00
Sci     0       0       21      0       0       1       0.95
USN     1       1       2       45      2       3       0.83
Spo     0       0       0       0       15      0       1.00
Wor     0       1       0       1       1       52      0.95
RECALL  0.95    0.50    0.81    0.98    0.83    0.93

F_1(Fin) = 0.869565217391304
F_1(Ent) = 0.666666666666667
F_1(Sci) = 0.875
F_1(USN) = 0.9
F_1(Spo) = 0.909090909090909
F_1(Wor) = 0.936936936936937
```

3.4 Attempt 3

3.4.1 Corpus 1

```
Processing answer file...
Found 5 categories: Oth Dis Pol Str Cri
Processing prediction file...

390 CORRECT, 53 INCORRECT, RATIO = 0.880361173814898.

CONTINGENCY TABLE:
Oth     Dis     Pol     Str     Cri     PREC
Oth     9       0       2       0       0       0.82
Dis     4       89      0       1       0       0.95
Pol     9       0       130     10      1       0.87
Str     3       0       12      123     10      0.83
Cri     0       0       0       1       39      0.97
RECALL  0.36    1.00    0.90    0.91    0.78

F_1(Oth) = 0.5
F_1(Dis) = 0.972677595628415
F_1(Pol) = 0.884353741496599
F_1(Str) = 0.869257950530035
F_1(Cri) = 0.866666666666667
```

3.4.2 Corpus 2

```
Processing answer file...
Found 2 categories: O I
Processing prediction file...

156 CORRECT, 23 INCORRECT, RATIO = 0.871508379888268.

CONTINGENCY TABLE:
O       I       PREC
O       118     14      0.89
I       9       38      0.81
RECALL  0.93    0.73

F_1(O) = 0.911196911196911
F_1(I) = 0.767676767676768
```

3.4.3 Corpus 3

```
Processing answer file...
Found 6 categories: USN Ent Spo Wor Sci Fin
Processing prediction file...

139 CORRECT, 36 INCORRECT, RATIO = 0.794285714285714.

CONTINGENCY TABLE:
USN     Ent     Spo     Wor     Sci     Fin     PREC
USN     44      3       5       5       11      7       0.59
Ent     0       2       0       0       0       0       1.00
```

```
Spo     0       0       13      0       0       0       1.00
Wor     2       1       0       51      0       0       0.94
Sci     0       0       0       0       15      0       1.00
Fin     0       2       0       0       0       14      0.88
RECALL  0.96    0.25    0.72    0.91    0.58    0.67

F_1(USN) = 0.727272727272727
F_1(Ent) = 0.4
F_1(Spo) = 0.838709677419355
F_1(Wor) = 0.927272727272727
F_1(Sci) = 0.731707317073171
F_1(Fin) = 0.756756756756757
```

3.5 Summary

Attempt	Corpus 1	Corpus 2	Corpus 3	Average
1	86.0	86.6	85.7	86.1
2	88.7	88.8	89.7	89.1
3	88.0	87.1	79.4	84.8

Table 1: Summary of performances of all attempts

3.6 Speed / Efficiency

The algorithm is not very fast, taking 10-20 seconds on the first and third corpus.

There is the potential for extreme speedup through parallelization, since the training stages can be entirely parallelized between train or evaluation cases (but the training and evaluation stages cannot be overlapped).