

Naïve-Bayes text classification model

Python Code Analysis Report

Victor Mai

April 4, 2025

**I acknowledge the use of Deepseek to refine word usage and polish grammar to fulfill the academic English requirements.*

1 Project Overview

This project implements a *naïve-bayes* text classification model and evaluates the performance of the implemented model in the Reuters dataset.

The project consists of two main parts:

1. **Train a classification model:** Apply Bayes rule to determine the class distribution possibility of the documents.
2. **Test and calculate the F1 score associated with:** Classify pre-processed test data using a trained classification model and measure model performance.

2 Train a classification model

2.1 Design Approach

- Uses `preprocess()` to preprocess the `train.json` and `test.json` with following steps:
 1. Replace "&", "<", ">", """, and "'" with spaces. (They are HTML signals!)
 2. Replace all symbols `?, !, <, >, :, ;, \n`, and `"` with spaces.
 3. Replace all apostrophes (`'`) with spaces, except for the pattern `s'` (possessive form).
 4. Convert all letters to lowercase.
 5. Apply `NLTK.word_tokenize()` for tokenization.
 6. For each token, remove redundant `^m` substrings at the end.
 7. For each token, remove redundant dots (`., . ., . . ., etc.`) at the end.
 8. Remove all tokens that do not contain at least one letter (a-z) or digit (0-9) or `$`.
 9. Apply `nlk.PorterStemmer` to the remaining words for stemming.
- Count the word frequency of train data and restore as **word_count.txt**.
- Select the first `n` words of the frequency (`n` is variable and will be used for dynamic testing) and save as **word_dict.txt**.
- Calculate the prior probability of each class and the posterior probability of each word in **word_dict.txt** (including `<UNKNOWN>` words, using Laplace smoothing).

2.2 Key Code of str parsing

```
1 def preprocess_str(text):
2     html_entities = {
3         "&amp;": "&",
4         "&lt;": "<",
5         "&gt;": ">",
6         "&quot;": "\"",
7         "&apos;": "'"
8     }
9     for entity, replacement in html_entities.items():
10         text = text.replace(entity, replacement)
11
12     # Replace them with space
13     symbols = r'[?!<>:\n"]'
14     text = re.sub(symbols, " ", text)
15
16     # Replace single quotes with spaces, except "s'"
17     text = re.sub(r'(?<!\w)'|'(?!\w|s')', " ", text)
18
19     # Lowercase
20     text = text.lower()
21
22     # Tokenize
23     tokens = word_tokenize(text)
24
25     # Remove "^M" signal
26     tokens = [re.sub(r'\^m$', '', token) for token in tokens]
27
28     # Remove redundant "."s
29     tokens = [re.sub(r'\.{2,}$', '', token) for token in tokens]
30     tokens = [re.sub(r'\.$', '', token) for token in tokens]
31
32     # Remove words that do not contain letters or numbers
33     tokens = [token for token in tokens if re.search(r'[a-z0-9$]', token)]
34
35     # PorterStemmer
36     stemmer = PorterStemmer()
37     tokens = [stemmer.stem(token) for token in tokens]
38
39     return tokens
```

Listing 1: Core of preprocessing

2.3 Pre-processing the JSON files

```
1 def preprocess(inputfile, outputfile):
2     with open(inputfile, 'r', encoding='utf-8') as f:
3         inputdata = json.load(f)
4         f.close()
5
6     for each in tqdm(inputdata):
7         each[2] = preprocess_str(each[2])
8
9     with codecs.open(outputfile, 'w', encoding='utf-8') as fo:
10         fo.write(json.dumps(inputdata))
11         fo.close()
12
13     return
```

Listing 2: Preprocess the json and save the words Informationlabel

2.4 Counting the frequency

```
1 def count_word(inputfile, outputfile):
2     with open(inputfile, 'r', encoding='utf-8') as f:
3         inputdata = json.load(f)
4         f.close()
5
6     to_idx = {'crude': 0, 'grain': 1, 'money-fx': 2, 'acq': 3, 'earn': 4}
7     counter = [0] * len(to_idx)
8
9     mydict = {}
10    for item in tqdm(inputdata):
11        if len(item) < 3:
12            continue
13
14        category = item[1]
15        words = item[2]
16
17        try:
18            idx = to_idx[category]
19        except KeyError:
20            continue
21        counter[idx] += 1
22
23        for word in words:
24            if word not in mydict:
25                mydict[word] = [0] * len(to_idx)
26            mydict[word][idx] += 1
27
28    with open(outputfile, 'w', encoding='utf-8') as fo:
29        line = '{} {} {} {} {} \n'.format(*counter)
30        fo.write(line)
31        for key, item in tqdm(mydict.items()):
32            line = '{} {} {} {} {} \n'.format(key, *item)
33            fo.write(line)
34        fo.close()
35    return
```

Listing 3: Count the frequency of words

2.5 Selecting the features

```
1 def feature_selection(inputfile, threshold, outputfile):
2     word_count = {}
3     with open(inputfile, 'r', encoding='utf-8') as f:
4         for line in tqdm(f):
5             line = line.strip()
6             words = line.split(' ')
7             if len(words) < 6:
8                 continue # First line
9             word_count[words[0]] = sum(int(x) for x in words[1:])
10    f.close()
11
12    top_keys = heapq.nlargest(min(len(word_count), threshold), word_count.keys(),
13                             key=word_count.get)
14    top_keys.sort()
15
16    with open(outputfile, 'w', encoding='utf-8') as fo:
17        for key in top_keys:
18            fo.write('{} \n'.format(key))
19        fo.close()
20    return
```

2.6 Calculating probabilities

```

1 def calculate_probability(word_count, word_dict, outputfile):
2     posterior_probability = {}
3
4     with open(word_dict, 'r', encoding='utf-8') as fwd:
5         for word in tqdm(fwd):
6             word = word.strip()
7             posterior_probability[word] = [Fraction(0, 1)] * 5
8         fwd.close()
9
10    prior_probability = [Fraction(0)] * 5
11    class_word_count = [Fraction(0)] * 5
12    one = Fraction(1, 1)
13    v = Fraction(len(posterior_probability), 1)
14
15    with open(word_count, 'r', encoding='utf-8') as fwc:
16        for line in tqdm(fwc):
17            line = line.strip()
18            words = line.split(' ')
19            if len(words) < 6: # First line
20                for i in range(len(words)):
21                    prior_probability[i] = Fraction(int(words[i]))
22                    n_doc = sum(prior_probability, Fraction(0, 1))
23                    for i in range(5):
24                        prior_probability[i] /= n_doc
25            elif words[0] in posterior_probability:
26                for i in range(1, len(words)):
27                    posterior_probability[words[0]][i - 1] = Fraction(int(words
28[i]), 1)
29                    class_word_count[i - 1] += Fraction(int(words[i]), 1)
30            else:
31                continue
32        fwc.close()
33
34    for key in posterior_probability:
35        for i in range(5):
36            posterior_probability[key][i] += one
37            posterior_probability[key][i] /= (class_word_count[i] + v)
38    posterior_probability['<UNKNOWN>'] = [Fraction(1, class_word_count[i] + v +
391)] * 5
40
41    with open(outputfile, 'w', encoding='utf-8') as fo:
42        fo.write(' '.join(str(pc) for pc in prior_probability) + '\n')
43        for key, item in tqdm(posterior_probability.items()):
44            fo.write(key + ' ' + ' '.join(str(post) for post in item) + '\n')
45        fo.close()
46    return

```

Listing 5: Calculate prior and posterior probabilities(including unknown words by using add-one)

3 Test and calculate the F1 score associated with

3.1 Design Approach

Text processing involves the following 2 steps:

1. Use `classify()` to classify test cases and save the results as `.txt` files.

2. Use `f1_score()` to calculate the F1 score of the trained model.

3.2 Classification Method

```
1 def classify(probability, testset, outputfile):
2     # Output the result to the output file in the format required
3     classes = ['crude', 'grain', 'money-fx', 'acq', 'earn']
4     classes_prob = np.zeros(5) # np . a r r a y [0.0]*5
5
6     prob = {}
7     with open(probability, 'r', encoding='utf-8') as f:
8         for line in tqdm(f):
9             line = line.strip()
10            items = line.split(' ')
11
12            if len(items) < 6:
13                for i in range(len(items)):
14                    frac = Fraction(items[i])
15                    classes_prob[i] = math.log(frac.numerator) - math.log(frac.
16denominator)
17                    continue
18
19            prob[items[0]] = np.array([math.log(Fraction(item).numerator) -
20math.log(Fraction(item).denominator)
21for item in items[1:6]])
22
23results = []
24with open(testset, 'r', encoding='utf-8') as f:
25    testdata = json.load(f)
26    for text in tqdm(testdata):
27        final_prob = classes_prob.copy()
28
29        for word in text[2]:
30            if word in prob:
31                final_prob += prob[word]
32            else:
33                final_prob += prob['<UNKNOWN>']
34
35        predicted_class = classes[final_prob.argmax()]
36        results.append((text[0], predicted_class))
37    f.close()
38
39with open(outputfile, 'w', encoding='utf-8') as fo:
40    for result in results:
41        fo.write('{} {} \n'.format(result[0], result[1]))
42
43return
```

Listing 6: Classify the testset

3.3 Calculate the F1 score

```
1 def f1_score(testset, classification_result, average='micro'):
2     y_true, y_pred = [], []
3     with open(testset, 'r', encoding='utf-8') as f:
4         testdata = json.load(f)
5         for text in tqdm(testdata):
6             y_true.append(text[1])
7         f.close()
8
9     with open(classification_result, 'r', encoding='utf-8') as f:
10        for result in tqdm(f):
```

```

11 result = result.strip()
12 case, predicted_class = result.split(' ')
13 y_pred.append(predicted_class)
14
15 micro_average_f1 = sklearn_f1_score(np.array(y_true), np.array(y_pred),
16 average=average)
17 return micro_average_f1

```

Listing 7: Calculate the F1 score

4 Result

Figure 1: train.preprocessed.json.png

Figure 2: test.preprocessed.json.png

5 Addition works

If the pre-loaded dictionaries are too large, it will consume too much performance. In order to better study the relationship between the model's performance and the number of features, I customized a method to compute the F1 of the model trained with different numbers of features, and plotted the

1	359 428 535 1617 2848
2	shad 0 0 0 10 0
3	see 44 32 60 43 252
4	progress 1 7 6 52 26
5	on 528 431 770 1185 1110
6	<u>insid</u> 1 1 3 22 4
7	trade 81 211 450 150 107
8	<u>secur</u> 33 4 111 414 129
9	and 1461 1536 1685 4216 3183
10	<u>exchang</u> 29 56 491 317 105
11	<u>commiss</u> 12 75 8 229 54
12	chairman 19 19 33 212 120
13	john 23 7 2 22 19
14	said 1506 1399 1872 4803 2773

(a) word.count.png

1	\$
2	'd
3	'll
4	'm
5	're
6	's
7	've
8	-aci
9	-banker
10	-daimler- <u>puch</u>
11	-dealer
12	-iri
13	-laval
14	-led
15	- <u>radiat</u>
16	.029

(b) word.dict.png

Figure 3: word.count and word.dict

7160	999.000 1/94483 1/94602 1/117360 1/216431 2/237893
7161	999.3 1/94483 1/94602 1/117360 1/216431 2/237893
7162	9p 1/94483 1/94602 1/117360 1/216431 3/237893
7163	a 1411/94483 188/15767 439/29340 4366/216431 3110/237893
7164	a.e 1/94483 2/47301 1/117360 1/216431 1/237893
7165	a.h 1/94483 1/94602 1/117360 1/216431 5/237893
7166	a.m 1/94483 1/94602 1/117360 2/216431 1/237893
7167	a.n 1/94483 1/94602 1/117360 2/216431 1/237893
7168	a.w 1/94483 1/94602 1/117360 1/216431 2/237893
7169	a/ 17/94483 1/94602 1/117360 5/216431 6/237893
7170	a14-8-89-3 2/94483 1/94602 1/117360 1/216431 1/237893
7171	aa 1/94483 1/94602 1/58680 2/216431 2/237893
7172	aac 1/94483 1/94602 1/117360 3/216431 2/237893
7173	aap 1/94483 1/94602 1/117360 1/216431 3/237893
7174	aar 1/94483 1/94602 1/117360 3/216431 2/237893
7175	aaron 1/94483 1/94602 1/117360 1/216431 3/237893
7176	ab 3/94483 1/94602 1/117360 52/216431 22/237893
7177	abandon 2/94483 1/31534 1/23472 5/216431 7/237893
7178	<u>abastecimiento</u> 1/94483 1/31534 1/117360 1/216431 1/237893
7179	<u>abat</u> 4/94483 1/94602 1/58680 2/216431 1/237893
7180	<u>abbett</u> 1/94483 1/94602 1/117360 1/216431 8/237893
7181	abbey 1/94483 1/94602 1/117360 1/216431 3/237893
7182	<u>abbrevi</u> 1/94483 1/94602 1/117360 1/216431 2/237893
7183	abdelaziz 3/94483 1/94602 1/117360 1/216431 1/237893
7184	abdul 2/94483 1/94602 1/117360 1/216431 2/237893
7185	abdul-aziz 6/94483 1/94602 1/117360 1/216431 1/237893
7186	abdul-rahim 3/94483 1/94602 1/117360 1/216431 1/237893

(a) word.probability.png

1	test/14975 earn
2	test/21067 crude
3	test/20081 money-fx
4	test/21040 acq
5	test/16228 earn
6	test/18689 crude
7	test/21462 acq
8	test/16833 acq
9	test/15255 acq
10	test/20548 earn
11	test/19325 acq
12	test/21406 earn
13	test/19165 grain
14	test/19048 acq
15	test/16705 earn
16	test/21063 acq
17	test/20968 earn
18	test/15739 acq
19	test/19850 earn
20	test/21393 acq
21	test/16665 earn
22	test/18644 acq

(b) classification_result.png

Figure 4: word.probability and classification_result

corresponding curve graph 5 (log scale).

* The number of features ranges from 10 to 20000 (log scale).

5.1 Relationship between the model's performance and the number of features

```
1 def plt_performance(start_n: int = 10, end_n: int = 20000, point_n: int = 50):
2     preprocess('train.json', 'train.preprocessed.json')
3     preprocess('test.json', 'test.preprocessed.json')
4     count_word('train.preprocessed.json', 'word_count.txt')
5     num_feature = np.logspace(
6         np.log10(start_n),
7         np.log10(end_n),
8         num=point_n,
9         dtype=int
10    ).tolist()
11    scores = []
12
13    for n in num_feature:
14        feature_selection('word_count.txt', n, 'word_dict.txt')
15        calculate_probability('word_count.txt', 'word_dict.txt', '
word_probability.txt')
16        classify('word_probability.txt', 'test.preprocessed.json', '
classification_result.txt')
17        score = f1_score('test.json', 'classification_result.txt')
18        scores.append(score)
19        print('{} features, F1 score: {:.4f}'.format(n, score))
20
21    plt.figure(figsize=(12, 6))
22    plt.plot(num_feature, scores, 'bo-', linewidth=2, markersize=8)
23    plt.xscale('log')
24    plt.xlabel('Number of features (log scale)')
25    plt.ylabel('F1 Score')
26    plt.title('Model Performance')
27    plt.grid(True, linestyle='--', alpha=0.7)
28
29    max_score = max(scores)
30    max_idx = scores.index(max_score)
31    plt.annotate(f'Max F1: {max_score:.4f}\n(n={num_feature[max_idx]})',
32                xy=(num_feature[max_idx], max_score),
33                xytext=(10, 10), textcoords='offset points',
34                bbox=dict(boxstyle='round,pad=0.5', fc='yellow', alpha=0.5),
35                arrowprops=dict(arrowstyle='->'))
36
37    plt.tight_layout()
38    plt.savefig('feature_vs_f1.png', dpi=300)
39    plt.show()
```

Listing 8: plt_performance.py

6 Conclusion

- Through this project, I clearly realised the importance of feature selection in text classification. Also, get a feel for the simplicity and practicality of Bayes RULE.
- In terms of model evaluation, the adoption of the F1 score as a core metric is of practical significance as it takes into account both Precision and Recall, and is particularly suitable for dealing with datasets with unbalanced categories.

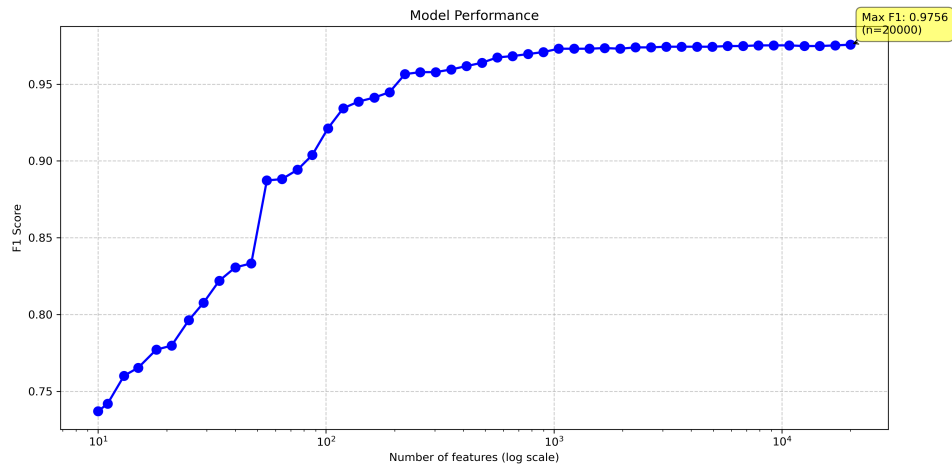


Figure 5: feature_vs.f1.png

- The highest F1 score obtained from the experiment is 0.9756, which occurs when the number of features is about 20000. (shown as Figure 5) However, if we extract only 200 feature with highest frequency, F1 can also reach the level of 0.95.
- Through this project, I have not only mastered the principle of implementing the plain Bayesian classifier, but also deeply understood the fundamental role of text preprocessing for NLP tasks. Meanwhile, I systematically practiced the complete process of machine learning project: from data preparation, feature engineering, model implementation to performance evaluation, which is a good exercise for my engineering ability.