Multinomial Naïve Bayes Learning

Yongxing NIE
College of Engineering
Northeastern University
Toronto, ON
nie.yo@northeastern.edu

Abstract

The objective of this project is constructing a naïve bayes classifier. The prior and conditional probability of the 20 news groups dataset is calculated separately, the naïve bayes classifier applied the prior and conditional probability to calculate the max posterior of the test data and assign a predict class to the test data. Classifying results with/without Inverse Document Frequency are compared.

1. Introduction

In this project, I will use the datasets 20 newsgroups. It is a popular dataset for experiment in text applications of machine learning techniques. I will construct three functions to implement calculation of logPrior, loglikelihood, and MNBclassifier. Laplace correction is applied in smoothing data. Classifying results with/without Inverse Document Frequency are compared.

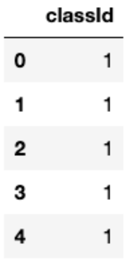
2. Experiments

(1) Datasets Inspection

In this project, I use the 20 newsgroups dataset (http://qwone.com/~jason/20Newsgroups/).
The data is organized into 20 different newsgroups, each corresponding to a different topic. Some of the newsgroups are very closely related to each other (group by the same color), while others are highly unrelated. Below is a list of the 20 newsgroups:

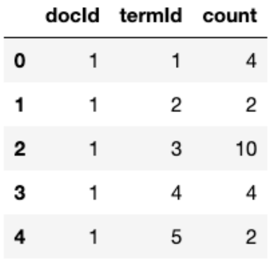
NO.	types	
1	comp.graphics	talk.politics.misc
2	comp.os.ms-windows.misc	talk.politics.guns
3	comp.sys.ibm.pc.hardware	talk.politics.mideast
4	comp.sys.mac.hardware	sci.crypt
5	comp.windows.x	sci.electronics
6	misc.forsale	sci.med
7	rec.autos	sci.space
8	rec.motorcycles	talk.religion.misc
9	rec.sport.baseball	alt.atheism
10	rec.sport.hockey	soc.religion.christian

There are six files in the dataset, the train label file contains only the groups information, I name it as class.



	classId
0	1
1	1
2	1
3	1
4	1

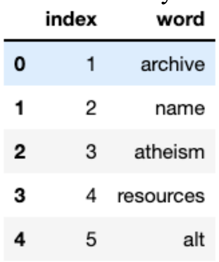
The train data contains the information for each document.



	docId	termId	count
0	1	1	4
1	1	2	2
2	1	3	10
3	1	4	4
4	1	5	2

The test label and test data have the same format.

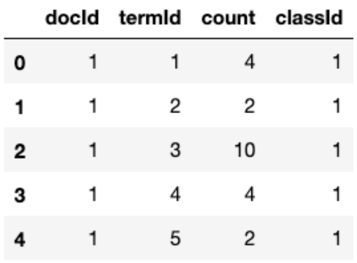
The vocabulary text file is the list of all the terms that are counted in the documents.



	index	word
0	1	archive
1	2	name
2	3	atheism
3	4	resources
4	5	alt

(2) Data Preparation

When calculating the probability of a term in a given class (news group type), I attached the train label data to the train data. The dataset with labels has the format as below:



	docId	termId	count	classId
0	1	1	4	1
1	1	2	2	1
2	1	3	10	1
3	1	4	4	1
4	1	5	2	1

(3) Methods

In this project, I have:

(1) Calculated the prior probabilities of the class.

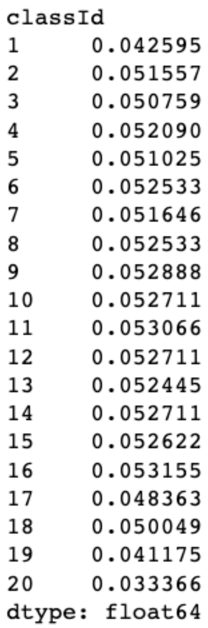
(2) Transformed the dataset by grouping "classId" and "termId", calculated the probability of each term in given a class, produced a probability table of any given term in any given class.

(3) Constructed a multinomial naïve bayes classifier, which will assign a "classId" to each document based on the maximum likelihood of the terms belonging to a class produced in rounds of class iteration.

(4) Applied inverse document frequency on the dataset to check whether stop words has affected the result or not.

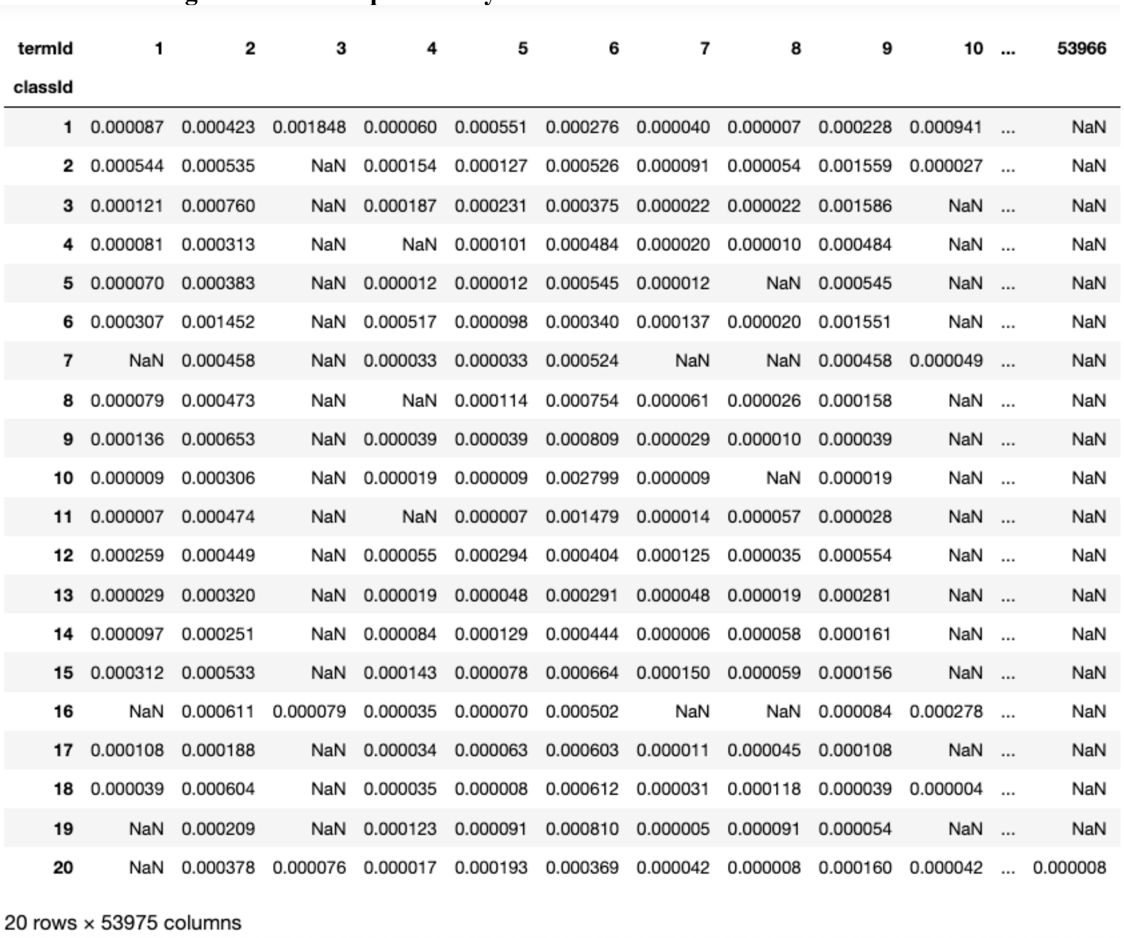
3. Results

(1) The prior probability of each class is calculated by the appearance of each class over the appearance of 20 classes in the train label data. The result is shown below:



```
classId
1     0.042595
2     0.051557
3     0.050759
4     0.052090
5     0.051025
6     0.052533
7     0.051646
8     0.052533
9     0.052888
10    0.052711
11    0.053066
12    0.052711
13    0.052445
14    0.052711
15    0.052622
16    0.053155
17    0.048363
18    0.050049
19    0.041175
20    0.033366
dtype: float64
```

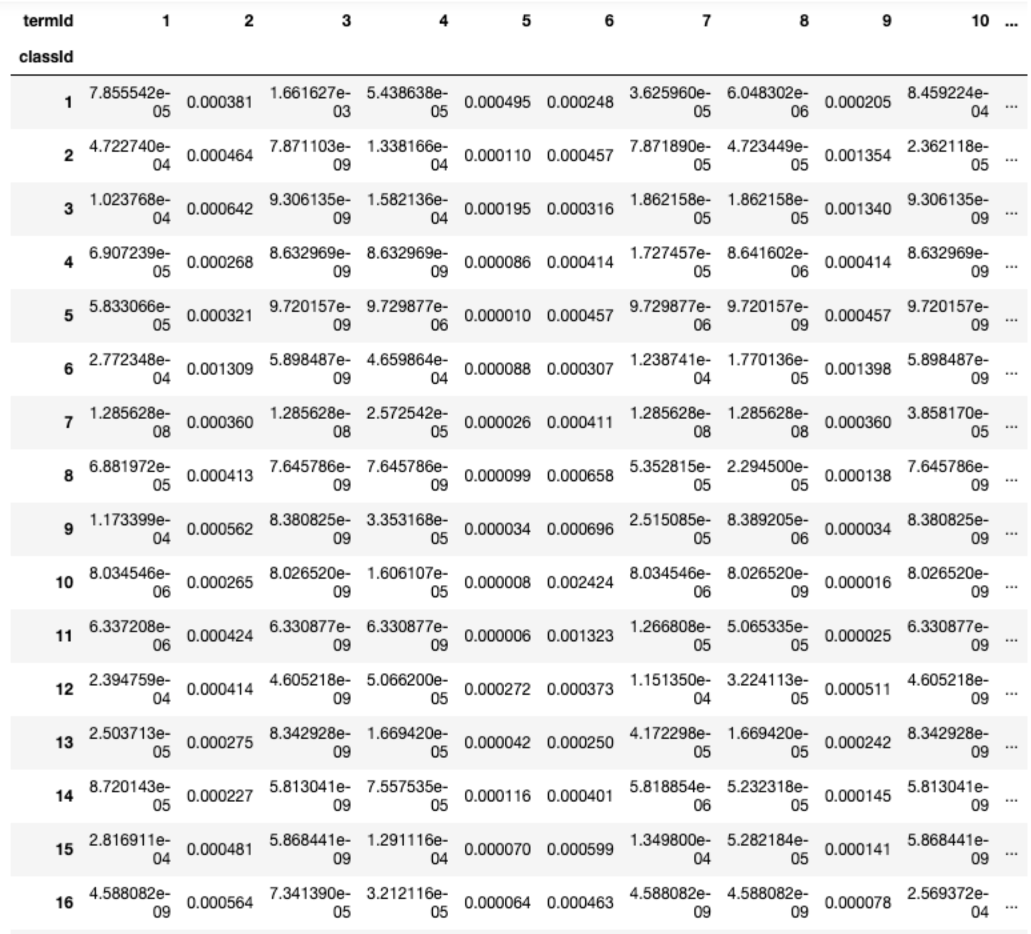
(2) The likelihood of each term given a class is calculated by the counts of each term over the counts of all terms in a given class. The probability table is shown below:



termId	1	2	3	4	5	6	7	8	9	10	...	53966
classId												
1	0.000087	0.000423	0.001848	0.000060	0.000551	0.000276	0.000040	0.000007	0.000228	0.000941	...	NaN
2	0.000544	0.000535	NaN	0.000154	0.000127	0.000526	0.000091	0.000054	0.001559	0.000027	...	NaN
3	0.000121	0.000760	NaN	0.000187	0.000231	0.000375	0.000022	0.000022	0.001586	NaN	...	NaN
4	0.000081	0.000313	NaN	NaN	0.000101	0.000484	0.000020	0.000010	0.000484	NaN	...	NaN
5	0.000070	0.000383	NaN	0.000012	0.000012	0.000545	0.000012	NaN	0.000545	NaN	...	NaN
6	0.000307	0.001452	NaN	0.000517	0.000098	0.000340	0.000137	0.000020	0.001551	NaN	...	NaN
7	NaN	0.000458	NaN	0.000033	0.000033	0.000524	NaN	NaN	0.000458	0.000049	...	NaN
8	0.000079	0.000473	NaN	NaN	0.000114	0.000754	0.000061	0.000026	0.000158	NaN	...	NaN
9	0.000136	0.000653	NaN	0.000039	0.000039	0.000809	0.000029	0.000010	0.000039	NaN	...	NaN
10	0.000009	0.000306	NaN	0.000019	0.000009	0.002799	0.000009	NaN	0.000019	NaN	...	NaN
11	0.000007	0.000474	NaN	NaN	0.000007	0.001479	0.000014	0.000057	0.000028	NaN	...	NaN
12	0.000259	0.000449	NaN	0.000055	0.000294	0.000404	0.000125	0.000035	0.000554	NaN	...	NaN
13	0.000029	0.000320	NaN	0.000019	0.000048	0.000291	0.000048	0.000019	0.000281	NaN	...	NaN
14	0.000097	0.000251	NaN	0.000084	0.000129	0.000444	0.000006	0.000058	0.000161	NaN	...	NaN
15	0.000312	0.000533	NaN	0.000143	0.000078	0.000664	0.000150	0.000059	0.000156	NaN	...	NaN
16	NaN	0.000611	0.000079	0.000035	0.000070	0.000502	NaN	NaN	0.000084	0.000278	...	NaN
17	0.000108	0.000188	NaN	0.000034	0.000063	0.000603	0.000011	0.000045	0.000108	NaN	...	NaN
18	0.000039	0.000604	NaN	0.000035	0.000008	0.000612	0.000031	0.000118	0.000039	0.000004	...	NaN
19	NaN	0.000209	NaN	0.000123	0.000091	0.000810	0.000005	0.000091	0.000054	NaN	...	NaN
20	NaN	0.000378	0.000076	0.000017	0.000193	0.000369	0.000042	0.000008	0.000160	0.000042	...	0.000008

20 rows × 53975 columns

I found that there are lots of NaN in the table. It occurs because not all the terms appear in every document. To solve this, we need to smooth the data by applying Laplace correction, which smooth all the data and fill the NaN with a term a/(count+|V|+1). The corrected likelihood table is as below:



termId	1	2	3	4	5	6	7	8	9	10	...
classId											
1	7.855542e-05	0.000381	1.661627e-03	5.438638e-05	0.000495	0.000248	3.625960e-05	6.048302e-06	0.000205	8.459224e-04	...
2	4.722740e-04	0.000464	7.871103e-09	1.338166e-04	0.000110	0.000457	7.871890e-05	4.723449e-05	0.001354	2.362118e-05	...
3	1.023768e-04	0.000642	9.306135e-09	1.582136e-04	0.000195	0.000316	1.862158e-05	1.862158e-05	0.001340	9.306135e-09	...
4	6.907239e-05	0.000268	8.632969e-09	8.632969e-09	0.000086	0.000414	1.727457e-05	8.641602e-06	0.000414	8.632969e-09	...
5	5.833066e-05	0.000321	9.720157e-09	9.729877e-06	0.000010	0.000457	9.729877e-06	9.720157e-09	0.000457	9.720157e-09	...
6	2.772348e-04	0.001309	5.898487e-09	4.659864e-04	0.000088	0.000307	1.238741e-04	1.770136e-05	0.001398	5.898487e-09	...
7	1.285628e-08	0.000360	1.285628e-08	2.572542e-05	0.000026	0.000411	1.285628e-08	1.285628e-08	0.000360	3.858170e-05	...
8	6.881972e-05	0.000413	7.645786e-09	7.645786e-09	0.000099	0.000658	5.352815e-05	2.294500e-05	0.000138	7.645786e-09	...
9	1.173399e-04	0.000562	8.380825e-09	3.353168e-05	0.000034	0.000696	2.515085e-05	8.389205e-06	0.000034	8.380825e-09	...
10	8.034546e-06	0.000265	8.026520e-09	1.606107e-05	0.000008	0.002424	8.034546e-06	8.026520e-09	0.000016	8.026520e-09	...
11	6.337208e-06	0.000424	6.330877e-09	6.330877e-09	0.000006	0.001323	1.266808e-05	5.065335e-05	0.000025	6.330877e-09	...
12	2.394759e-04	0.000414	4.605218e-09	5.066200e-05	0.000272	0.000373	1.151350e-04	3.224113e-05	0.000511	4.605218e-09	...
13	2.503713e-05	0.000275	8.342928e-09	1.669420e-05	0.000042	0.000250	4.172298e-05	1.669420e-05	0.000242	8.342928e-09	...
14	8.720143e-05	0.000227	5.813041e-09	7.557535e-05	0.000116	0.000401	5.818854e-06	5.232318e-05	0.000145	5.813041e-09	...
15	2.816911e-04	0.000481	5.868441e-09	1.291116e-04	0.000070	0.000599	1.349800e-04	5.282184e-05	0.000141	5.868441e-09	...
16	4.588082e-09	0.000564	7.341390e-05	3.212116e-05	0.000064	0.000463	4.588082e-09	4.588082e-09	0.000078	2.569372e-04	...

(3) For the classifier, there are steps to take:

- **Transforming the test data into the form of:**

```
{{doc1: {term1, count},
        {term2, count},
        {term3, count},
        ...}
 {doc2: {term1, count},
        {term2, count},
        {term3, count},
        ...}
  ……
}
```

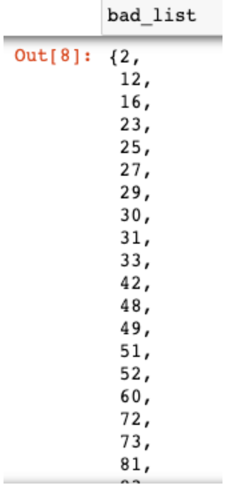
- **For each document in the test data, assuming they have the likelihood belonging to any class. Since I already have the probability table for each term in a given class, I will calculate the likelihood of 20 classes by adding up the log probability of each term in the document given a class.**
- **Above is just the conditional probability, then I will continue adding log prior probability of each class to the conditional one.**
- **I will get 20 probabilities; each corresponds to a class. At last, I will assign the class with the maximum probability to the document.**
- **For rounds of iteration, I will have the prediction of each document.**
- **During the iterations, I run into 4 cases where the terms only appear in the test data. At this step, I will assign 0 as its probability since the log result of Laplace smoothing is almost 0.**

The prediction accuracy is 17.95%. which is not satisfying.

(4) I then applied the TF-IDF to improve the classifying performance.

Since I have already imbedded the inverse document frequency in the MNB classifier, I then only need to choose the terms to be included. I found a stop words list online, which contains words that we frequently use but don't mean much in presentation, such as prepositions and pronouns.

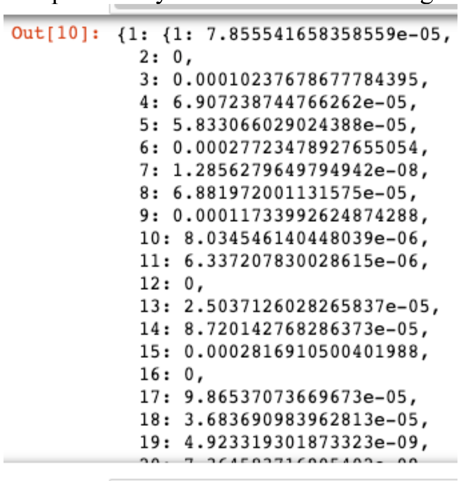
Of course, the documents in train data contain lots of terms as shown in stop words list. I then set the probability of those terms to 0 to remove those terms when calculating the likelihood . In total, I removed 250 terms. The removed term list is as below:



```
         bad_list

Out[8]: {2,
         12,
         16,
         23,
         25,
         27,
         29,
         30,
         31,
         33,
         42,
         48,
         49,
         51,
         52,
         60,
         72,
         73,
         81,
```

The probability table of each term in a given class after TF-IDF is as below:

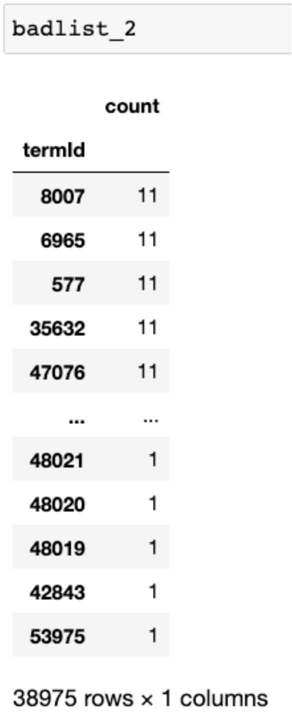


```
Out[10]: {1: {1: 7.855541658358559e-05,
           2: 0,
           3: 0.00010237678677784395,
           4: 6.907238744766262e-05,
           5: 5.833066029024388e-05,
           6: 0.00027723478927655054,
           7: 1.2856279649794942e-08,
           8: 6.881972001131575e-05,
           9: 0.00011733992624874288,
           10: 8.034546140448039e-06,
           11: 6.337207830028615e-06,
           12: 0,
           13: 2.5037126028265837e-05,
           14: 8.720142768286373e-05,
           15: 0.0002816910500401988,
           16: 0,
           17: 9.86537073669673e-05,
           18: 3.683690983962813e-05,
           19: 4.923319301873323e-09,
```

Then I do the MNB classifier again by applying the new likelihood table. And the prediction accuracy is 17.95%, exactly the same as before TF-IDF.

The reason I thought might be that there are 53975 terms in the data set, removing 250 terms (0.5%) won't impact the terms data much.

I then increased the bad list by count the appearance of each term. I found that most of the terms appeared only once, and there are only 15000 terms (27.8%) which appeared 10 times or above. Then I choose the 10 times as a benchmark and label terms with frequency below 10 as bad term. The bad list is as below:

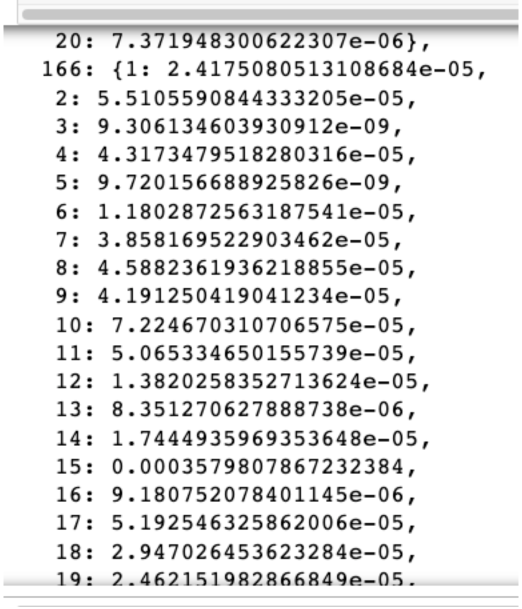


```
badlist_2
```

termId	count
8007	11
6965	11
577	11
35632	11
47076	11
...	...
48021	1
48020	1
48019	1
42843	1
53975	1

38975 rows × 1 columns

The probability table of each term in a given class after TF-IDF is as below:



```
  20: 7.371948300622307e-06},
 166: {1: 2.4175080513108684e-05,
  2: 5.5105590844333205e-05,
  3: 9.306134603930912e-09,
  4: 4.3173479518280316e-05,
  5: 9.720156688925826e-09,
  6: 1.1802872563187541e-05,
  7: 3.858169522903462e-05,
  8: 4.5882361936218855e-05,
  9: 4.191250419041234e-05,
  10: 7.224670310706575e-05,
  11: 5.065334650155739e-05,
  12: 1.3820258352713624e-05,
  13: 8.351270627888738e-06,
  14: 1.7444935969353648e-05,
  15: 0.0003579807867232384,
  16: 9.180752078401145e-06,
  17: 5.192546325862006e-05,
  18: 2.947026453623284e-05,
  19: 2.462151982866849e-05,
```

Then I do the MNB classifier again by applying the new likelihood table. Still the prediction accuracy is only 17.95%, exactly the same as before TF-IDF.

Even when I decrease the dataset to 27.8% of its original size, the prediction accuracy is not improved. The reason might be that TF-IDF helps determine how a term is related to a given document. However, if we want to improve the accuracy, we need to explore term distribution with respect to class. In this project, even I removed the low frequency terms, I do not change the distribution of the high frequency term with respect to the class. In other words, the probability distribution of important features are always the same with/without TF-IDF.

4. Conclusions

(1) My MNB classifier accuracy is pretty low, I have tried my best to write the classifier function, but now I haven't found the reason why it's low.
(2) In my project, TF-IDF does not improve the prediction accuracy.

Acknowledgement

The code is adjusted from https://towardsdatascience.com/multinomial-naive-bayes-classifier-for-text-analysis-python-8dd6825ece67 and https://towardsdatascience.com/implementing-naive-bayes-algorithm-from-scratch-python-c6880cfc9c41 .