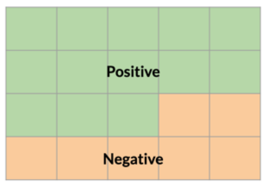
Sentiment analysis with Naïve Bayes

Probability and Bayes' Rule

Corpus of tweets

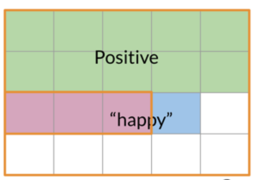


A → Positive tweet

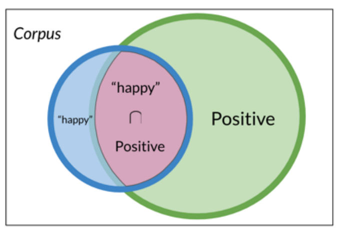
P(A) = N_{pos} / N = 13 / 20 = 0.65

P(Negative) = 1 - P(Positive) = 0.35

To calculate a probability of a certain event happening, you take the count of that specific event and you divide by the sum of all events. Furthermore, the sum of all probabilities has to equal 1.



$$P(A \cap B) = P(A, B) = \frac{3}{20} = 0.15$$



To compute the probability of 2 events happening, like "happy" and "positive" in the picture above, you would be looking at the intersection, or overlap of events.

From the equations presented below, express the probability of a tweet being positive given that it contains the word happy in terms of the probability of a tweet containing the word happy given that it is positive

$$P(\text{Positive}|\text{“happy”}) = \frac{P(\text{Positive} \cap \text{“happy”})}{P(\text{“happy”})} \qquad P(\text{“happy”}|\text{Positive}) = \frac{P(\text{“happy”} \cap \text{Positive})}{P(\text{Positive})}$$

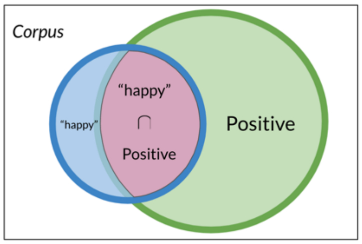
$$P(\text{Positive}|\text{“happy”}) = P(\text{“happy”} \cap \text{Positive}) \times \frac{P(\text{“happy”})}{P(\text{Positive})}$$

$$P(\text{Positive}|\text{“happy”}) = P(\text{“happy”}|\text{Positive}) \times \frac{P(\text{Positive})}{P(\text{“happy”})}$$

Correct
That's right. You just derived Bayes' rule.

Bayes' Rule

Conditional probabilities help us reduce the sample search space. For example given a specific event already happened, i.e. we know the word is happy:



$$P(\text{Positive} | \text{“happy”}) = \frac{P(\text{Positive} \cap \text{“happy”})}{P(\text{“happy”})}$$

Then you would only search in the blue circle above. The numerator will be the red part and the denominator will be the blue part. This leads us to conclude the following:

$$P(\text{Positive} | \text{“happy”}) = \frac{\boxed{P(\text{Positive} \cap \text{“happy”})}}{P(\text{“happy”})}$$

$$P(\text{“happy”} | \text{Positive}) = \frac{\boxed{P(\text{“happy”} \cap \text{Positive})}}{P(\text{Positive})}$$

Substituting the numerator in the right-hand side of the first equation, you get the following:

$$\boxed{P(\text{Positive} | \text{“happy”})} = \boxed{P(\text{“happy”} | \text{Positive})} \times \boxed{\frac{P(\text{Positive})}{P(\text{“happy”})}}$$

Note that we multiplied by P(positive) to make sure we don't change anything. That concludes Bayes Rule which is defined as

P(X|Y) = P(Y|X) P(X) / P(Y)

Naive Bayes Introduction

To build a classifier, we will first start by creating conditional probabilities given the following table:

Positive tweets

I am happy because I am learning NLP

I am happy, not sad.

Negative tweets

I am sad, I am not learning NLP

I am sad, not happy

word	Pos	Neg
I	3	3
am	3	3
happy	2	1
because	1	0
learning	1	1
NLP	1	1
sad	1	2
not	1	2
N_{class}	13	12

This allows us compute the following table of probabilities:

word	Pos	Neg
I	0.24	0.25
am	0.24	0.25
happy	0.15	0.08
because	0.08	0
learning	0.08	0.08
NLP	0.08	0.08
sad	0.08	0.17
not	0.08	0.17

Once you have the probabilities, you can compute the likelihood score as follows

Tweet: I am happy today; I am learning.

$$\prod_{i=1}^{m} \frac{P(w_i|pos)}{P(w_i|neg)} = \frac{0.14}{0.10} = 1.4 \quad > 1$$

$$\cancel{\frac{0.20}{0.20}} * \cancel{\frac{0.20}{0.20}} * \boxed{\frac{0.14}{0.10}} * \cancel{\frac{0.20}{0.20}} * \cancel{\frac{0.20}{0.20}} * \cancel{\frac{0.10}{0.10}}$$

word	Pos	Neg
I	0.20	0.20
am	0.20	0.20
happy	0.14	0.10
because	0.10	0.05
learning	0.10	0.10
NLP	0.10	0.10
sad	0.10	0.15
not	0.10	0.15

A score greater than 1 indicates that the class is positive, otherwise it is negative.

Laplacian Smoothing

We usually compute the probability of a word given a class as follows:

$$P\left(\mathrm{w_i} \mid \text{ class }\right)=\frac{\operatorname{freq}\left(\mathrm{w_i}, \text{ class }\right)}{\mathrm{N_{class}}} \qquad \text{class } \in \{\text{ Positive, Negative }\}$$

However, if a word does not appear in the training, then it automatically gets a probability of 0, to fix this we add smoothing as follows

$$P\left(\mathrm{w_i} \mid \text{class}\right)=\frac{\operatorname{freq}\left(\mathrm{w_i}, \text{ class }\right)+1}{\left(\mathrm{N_{class}}+\mathrm{V}\right)}$$

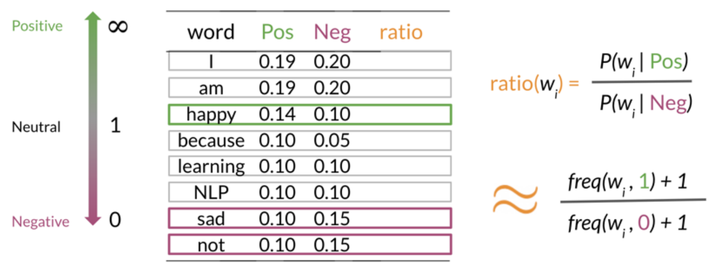
Note that we added a 11 in the numerator, and since there are VV words to normalize, we add VV in the denominator.

N_{class}: frequency of all words in class

V: number of unique words in vocabulary

Log Likelihood, Part 1

To compute the log likelihood, we need to get the ratios and use them to compute a score that will allow us to decide whether a tweet is positive or negative. The higher the ratio, the more positive the word is:



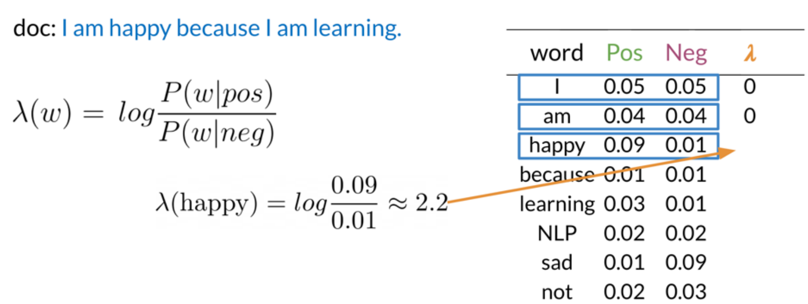
To do inference, you can compute the following:

$$\frac{P(pos)}{P(neg)} \prod_{i=1}^{m} \frac{P(w_i|pos)}{P(w_i|neg)} > 1$$

As m gets larger, we can get numerical flow issues, so we introduce the $\log$, which gives you the following equation:

$$\log\left(\frac{P(pos)}{P(neg)} \prod_{i=1}^{n} \frac{P(w_i|pos)}{P(w_i|neg)}\right) \Rightarrow \log \frac{P(pos)}{P(neg)} + \sum_{i=1}^{n} \log \frac{P(w_i|pos)}{P(w_i|neg)}$$

The first component is called the log prior and the second component is the log likelihood. We further introduce \lambda$\lambda$ as follows:



Having the λ dictionary will help a lot when doing inference.

Log Likelihood Part 2

Once you computed the λ dictionary, it becomes straightforward to do inference:

doc: I am happy because I am learning.

$$\sum_{i=1}^{m} log \frac{P(w_i|pos)}{P(w_i|neg)} = \sum_{i=1}^{m} \lambda(w_i)$$

log likelihood = 0 + 0 + 2.2 + 0 + 0 + 0 + 1.1 = **3.3**

word	Pos	Neg	λ
I	0.05	0.05	0
am	0.04	0.04	0
happy	0.09	0.01	2.2
because	0.01	0.01	0
learning	0.03	0.01	1.1
NLP	0.02	0.02	0
sad	0.01	0.09	-2.2
not	0.02	0.03	-0.4

As you can see above, since $3.3 > 0$, we will classify the document to be positive. If we got a negative number, we would have classified it to the negative class.

Training naïve Bayes

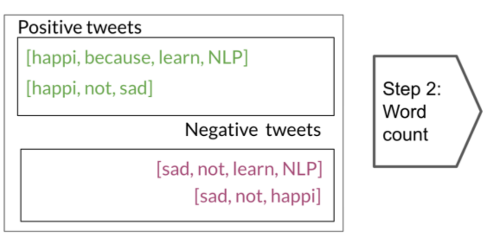
To train your naïve Bayes classifier, you have to perform the following steps:

1) Get or annotate a dataset with positive and negative tweets

2) Preprocess the tweets: process_tweet(tweet) ➞ [w1, w2, w3, ...]:

- Lowercase
- Remove punctuation, urls, names
- Remove stop words
- Stemming
- Tokenize sentences

3) Compute freq(w, class):



word	Pos	Neg
happi	2	1
because	1	0
learn	1	1
NLP	1	1
sad	1	2
not	1	2
N_{class}	7	7

freq(w, class)

4) Get $P(w|pos), P(w|neg)$

You can use the table above to compute the probabilities.

5) Get $\lambda(w)$

$\lambda(w)=\log\left(P(w|neg) / P(w|pos)\right)$

6) Compute logprior = log(P(pos) / P(neg))

$\text{logprior} = \log D_{pos} / D_{neg}$, where D_{pos} and D_{neg} correspond to the number of positive and negative documents respectively.

Testing naïve Bayes

- log-likelihood dictionary $\lambda(w) = log\frac{P(w|pos)}{P(w|neg)}$
- $logprior = log\frac{D_{pos}}{D_{neg}} = 0$
- Tweet: [I, pass, the, NLP, interview]

$score = -0.01 + 0.5 - 0.01 + 0 + logprior = 0.48$

$pred = score > 0$

word	λ
I	-0.01
the	-0.01
happi	0.63
because	0.01
pass	0.5
NLP	0
sad	-0.75
not	-0.75

The example above shows how you can make a prediction given your λ dictionary. In this example the $logprior$ is 0 because we have the same amount of positive and negative documents (i.e. $\log 1 = 0$).

Applications of Naive Bayes

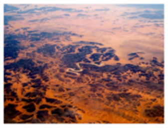
There are many applications of naive Bayes including:

- Author identification
- Spam filtering
- Information retrieval
- Word disambiguation

This method is usually used as a simple baseline. It is also really fast.

Naïve Bayes Assumptions

Naïve Bayes makes the independence assumption and is affected by the word frequencies in the corpus. For example, if you had the following



"It is sunny and hot in the Sahara desert."

"It's always cold and snowy in ___ ."

In the first image, you can see the word sunny and hot tend to depend on each other and are correlated to a certain extent with the word "desert". Naive Bayes assumes independence throughout. Furthermore, if you were to fill in the sentence on the right, this naive model will assign equal weight to the words "spring, summer, fall, winter".

Relative frequencies in corpus

On Twitter, there are usually more positive tweets than negative ones. However, some "clean" datasets you may find are artificially balanced to have to the same amount of positive and negative tweets. Just keep in mind, that in the real world, the data could be much noisier.

Error Analysis

There are several mistakes that could cause you to misclassify an example or a tweet. For example,

- Removing punctuation
- Removing words

Tweet: This is not good, because your attitude is not even close to being nice.

processed_tweet: [good, attitude, close, nice]

Tweet: My beloved grandmother :(

processed_tweet: [belov, grandmoth]

- Word order

Tweet: I am happy because I did not go.

Tweet: I am not happy because I did go.

- Adversarial attacks

These include sarcasm, irony, euphemisms.

Confidence Ellipse to interpret Naïve Bayes

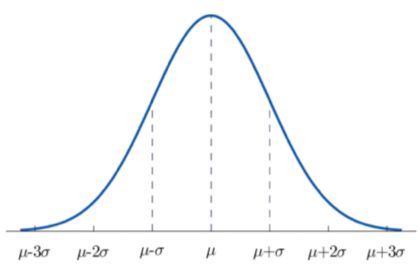
https://matplotlib.org/3.1.1/gallery/statistics/confidence_ellipse.html#sphx-glr-gallery-statistics-confidence-ellipse-py

A confidence ellipse is a way to visualize a 2D random variable. It is a better way than plotting the points over a cartesian plane because, with big datasets, the points can overlap badly and hide the real distribution of the data. Confidence ellipses summarize the information of the dataset with only four parameters:

- Center: It is the numerical mean of the attributes
- Height and width: Related with the variance of each attribute. The user must specify the desired amount of standard deviations used to plot the ellipse.
- Angle: Related with the covariance among attributes.

The parameter **n_std** stands for the number of standard deviations bounded by the ellipse. Remember that for normal random distributions:

- About 68% of the area under the curve falls within 1 standard deviation around the mean.
- About 95% of the area under the curve falls within 2 standard deviations around the mean.
- About 99.7% of the area under the curve falls within 3 standard deviations around the mean.



Naive bayes is an algorithm that could be used for sentiment analysis. It takes a short time to train and also has a short prediction time.

So how do you train a Naive Bayes classifier?

- The first part of training a naive bayes classifier is to identify the number of classes that you have.
- You will create a probability for each class. $P(D_{pos})$P(Dpos) is the probability that the document is positive. $P(D_{neg})$P(Dneg) is the probability that the document is negative. Use the formulas as follows and store the values in a dictionary:

$$P(D_{pos})=D_{pos}D(1)$$ (1)P(Dpos)=DposD

$$P(D_{neg})=D_{neg}D(2)$$ (2)P(Dneg)=DnegD

Where DD is the total number of documents, or tweets in this case, D_{pos}Dpos is the total number of positive tweets and D_{neg}Dneg is the total number of negative tweets.

Prior and Logprior

The prior probability represents the underlying probability in the target population that a tweet is positive versus negative. In other words, if we had no specific information and blindly picked a tweet out of the population set, what is the probability that it will be positive versus that it will be negative? That is the "prior".
The prior is the ratio of the probabilities $P(D_{pos})/P(D_{neg})$ We can take the log of the prior to rescale it, and we'll call this the logprior

$$\text{logprior}=log(P(D_{pos})/P(D_{neg}))=log(D_{pos}/D_{neg})$$

.

Note that $log(A/B)$ is the same as $log(A)-log(B)$ So the logprior can also be calculated as the difference between two logs:

$$\text{logprior}=\log(P(D_{pos}))-\log(P(D_{neg}))=\log(D_{pos})-\log(D_{neg})$$

Positive and Negative Probability of a Word

To compute the positive probability and the negative probability for a specific word in the vocabulary, we'll use the following inputs:

- $freq_{pos}$ and $freq_{neg}$ are the frequencies of that specific word in the positive or negative class. In other words, the positive frequency of a word is the number of times the word is counted with the label of 1.
- N_{pos} and N_{neg} are the total number of positive and negative words for all documents (for all tweets), respectively.
- VV is the number of unique words in the entire set of documents, for all classes, whether positive or negative.

We'll use these to compute the positive and negative probability for a specific word using this formula:

$$P(W_{pos})=freq_{pos}+1/N_{pos}+V(4)$$ (4)P(Wpos)=freqpos+1Npos+V

$$P(W_{neg})=freq_{neg}+1/N_{neg}+V(5)$$ (5)P(Wneg)=freqneg+1Nneg+V

Notice that we add the "+1" in the numerator for additive smoothing. This wiki article explains more about additive smoothing.

Log likelihood

To compute the loglikelihood of that very same word, we can implement the following equations:

$$\text{loglikelihood}=\log(P(W_{pos})/P(W_{neg}))(6)$$ (6)loglikelihood=log(P(Wpos)/P(Wneg))

Create `freqs` dictionary

- Given your `count_tweets()` function, you can compute a dictionary called `freqs` that contains all the frequencies.
- In this `freqs` dictionary, the key is the tuple (word, label)
- The value is the number of times it has appeared.