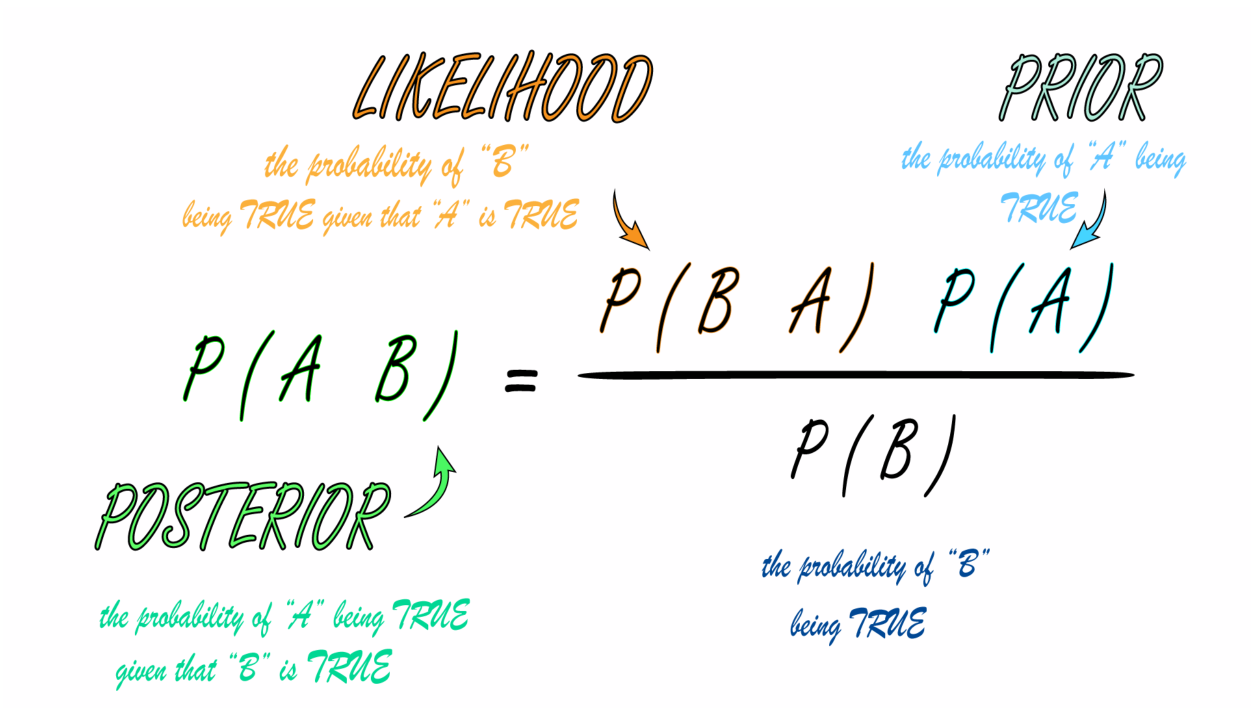
Naive Baye’s Classifier Working along with Calculations

This document describes naive baye’s with its working, multinomial naive baye’s formula explained with an example. Naive Baye’s pros and cons, its usage, error are briefly summarized. Manipulate default behavior of multinomial naive baye’s by changing default smoothing, explained with pseudo code with an example along with the calculations, and given the same dataset, multinomial naive baye’s is compared with other classifier.

What is Naive Baye’s algorithm?

One of sub-classes of Bayes’ Network is Naive Baye’s, which is a classification technique. It makes independent assumptions, and yet provides an interesting point on the trade off curve.

Example:
A boy has short hair, small lips, a strong body, small eyes, wider chin, and hollow cheek. All these features look dependent on each other but they independently contribute towards the probability that the boy is a male. This is why it is considered naive.


LIKELIHOOD
the probability of "B" being TRUE given that "A" is TRUE
PRIOR
the probability of "A" being TRUE
P(A B) = P(B A) P(A) / P(B)
POSTERIOR
the probability of "A" being TRUE given that "B" is TRUE
the probability of "B" being TRUE

How Naive Baye's Works?

A detailed explanation of Multinomial Naive Baye's from given training and testing data tables, along with calculation of likelihood, prior, and maximum posteriori.
Likelihood, prior are calculated to get maximum posteriori.
Likelihood of a given word with given class is defined as the count of unique words with smoothing added from the count of the given class and vocabulary.
Note: By default, the value of alpha is 1, but can be changed. It is recommended not to escalate the smoothing from 1, otherwise, it will affect the accuracy of the model.

- Probability of word in class = count of given word in class + alpha(1) /
- $P(w|c) = count(w,c) + 1 / count(c) + V$

Prior is the probability of a given class is the number of the unique class token over all the class tokens.

- Probability of given class = count of unique class tokens / total number of classes
- $P(c) = Nc / N$

Maximum Posteriori is computed to pick the best class, it is defined as the conditional probability of a class in a given document (row). Here the term document is represented by a row and features as words.

- Probability of given class in a document = probability of a document (vectors of features) in class * probability of class / probability of document
- Class_max = $P(c|d)$
  $= P(d|c) * P(c) / P(d)$ => eq.1
  $= P(d|c) * P(c)$ => eq.2
  $= P(f1,f2,...,fn|c) * P(c)$ where Document (d) be joint features $(f1,f2,...fn)$

In Bayes rule, the class which maximizes in eq.1 also maximizes in eq.2, so the denominator has not been taken into account. The probability of document $P(d)$ has dropped because the probability of the document is identical to all of the classes

Example 1:

In this example, three variables are initialized with integers that are divided with a constant denominator. Finally, the variable with the maximum result after division is considered

$a = 1.5, b = 19, c = 9.8$

result_1 = $max (a / 5, b / 5, c / 5)$
$= max (0.3, 3.8, 1.96)$
$= 3.8$

Maximum variable is *b*

Example 2:

In this example, variables from the above example are taken. Variable with maximum value is considered.

$a = 1.5, b = 19, c = 9.8$

First Result without dropping denominator

result_1 = $max (a, b, c)$
$= max (1.5, 19, 9.8)$
$= 19$

Maximum variable is b

In the above examples, the maximum variable is b. This satisfies that removing denominator from eq.1: $P(c|d) = P(d|c) * P(c) / P(d)$ in Maximum Posteriori will have same result as eq.2: $P(c|d) = P(d|c) * P(c)$. Hence, a divisor with the same value for all the classes will result the same as dropping the divisor.

What are the advantages and disadvantages of using Naive Baye’s?

Advantages:

- Computationally efficient
- Used for large datasets
- Can also be used with a small training dataset
- Handle both continuous data, as well as discrete data
- Its performance is better for datasets with equally important features
- Easy to Construct

Disadvantages:

- Naive Baye’s undertake all features not dependent on each other
- For numerical data, the Naive Baye’s classifier does not perform well as compared to categorical data
- Zero-frequency issue (words having no frequency within a class)

Applications of Naive Baye’s Classifier

Naive Baye’s classifier is used in many areas, some of them are:

- Face Identification
- Weather prognosis
- Medical detection
- Spam discernment
- Age/gender/ language distinguishing
- comments analysis
- Document categorization

What is Zero-Frequency problem and how to deal with it?

Given below is the dataset having four documents and 2 classes.

Doc	Words	Class
1	Chinese Beijing Chinese	c
2	Chinese Chinese Shanghai	c
3	Chinese Macao	c
4	Tokyo Japan Chinese	j

By default, one is added to the count of each unique word to avoid the zero-frequency problem. The example below has represented word count in each class that is incremented by one. If we see the training table, the token “Tokyo” has occurred 0 times in class c, which is converted to for avoiding zero-frequency error.

Words in class		Actual Count	Actual Count + Smoothing
Chinese	c	5	5 + 1
Tokyo	c	0	0 + 1
Japan	c	0	0 + 1
Chinese	j	1	1 + 1
Tokyo	j	1	1 + 1
Japan	j	1	1 + 1

Types of Naive Baye's Classifier:

There are three types of Naive Baye's Classifiers, the table below describes these classes.

Naive Baye's Classifiers	Explanation
Multinomial Naive Baye's	Features represent certain events, commonly used in document classification
Bernoulli Naive Baye's	Independent Booleans features
Gaussian Naive Baye's	Features having continuous values

Tuning Hyperparameters of Multinomial Naive Baye's Model

P(class|testing_document) is directly proportional to P(class) * P(each word in class). In the testing data table, "Chinese" has occurred 5 times in class "c". While calculating probability of word "Chinese" in class "c", the formula is P(word|c) = count(class) + alpha / count(class) + Vocabulary

$$P(Chinese|c) = (5+1) / (8+6) = 6/14 = 0.43$$

By default, the hyperparameter alpha is set to one, it can be tuned to get better results. If alpha is set to 0.6, then the probability of "Chinese" in given class "c" will be

$$P(Chinese|c) = (5+0.6) / 8+6) = 5.6/14 = 0.40$$

Given code block under describes how to change hyperparameter alpha in multinomial naive baye's:

```python
from sklearn.naive_bayes import MultinomialNB
multinomial_naive_bayes = MultinomialNB(alpha=0.6)
```

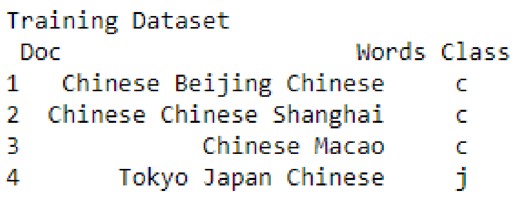
To know why alpha is used, check the section: What is zero-frequency problem and how to deal with it?

Multinomial Naive Baye's Explained with Pseudo Code along with Calculations

The example below is taken from **Dan Jurafsky** lectures on Multinomial Naive Baye's for Text Classification, using that example, this algorithm is implemented by **Nabeer Dar** and the dataset.csv, test.csv, reference code in python (algorithm_code.ipynb) are uploaded here: ***https://github.com/nabeerdar/naive_bayes***
Code from the main body from above Github repository is explained with pseudo-code along with calculations and screenshots of every output is described below

Step 01: Read training data

```python
dataset = read_dataset('dataset.csv')
```



```
Training Dataset
 Doc                      Words Class
1   Chinese Beijing Chinese     c
2  Chinese Chinese Shanghai     c
3             Chinese Macao     c
4       Tokyo Japan Chinese     j
```

Step 02: Read testing data

```
testdata = read_dataset('testdata.csv')
```

```
Testing Dataset
 Doc                                  Words  Class
1  Chinese Chinese Chinese Tokyo Japan    NaN
```

Step 03: Input class column name from user

```
class_attribute = input("Enter class column name: ")
```

```
Enter class column name:  Class
```

Step 04: Input words (tokens) column name from user

```
tokens_attribute = input("Enter words column name: ")
```

```
Enter words column name:  Words
```

Step 05: Initialize an empty dictionary. On step 11, posteriority for each class will be stored here

```
classes_posterior = {}
```

Step 06: Make a function variable to store unique words from test data

```
unique_test_tokens = get_unique_tokens(testdata, tokens_attribute)
```

```
Unique Test Tokens:  ['Chinese', 'Tokyo', 'Japan']
```

Step 07: Make a function variable that calculates document count of training data

```
document_count = calculate_document_count(dataset)
```

Step 08: Make a function variable to calculate unique classes from training data

```
unique_classes = get_unique_classes(dataset, class_attribute)
```

Step 09: Make a function variable to count number of unique tokens from training data for specific class

```
vucablary_count = count_vucabulary_tokens(dataset, tokens_attribute, class_attribute)
```

Step 10: Count number of occurrences of each token in test data

```
testdata_unique_token_count = count_unique_token_testdata(testdata, unique_token, tokens_attribute)
```

```
Japan Count: 1

Chinese Count: 3

Tokyo Count: 1
```

Step 11: Calculate posteriority for each class

1. Calculate class count for each class

```
unique_class_count = calculate_class_count(dataset, unique_class, class_attribute)
```

```
 Class c count: 3

Class j count: 1
```

2. Calculate class probability for each class (prior)

```
class_probibility = calculate_class_probibility(unique_class_count, document_count)
```

```
Probability of class c = 3/4 =  0.75

Probability of class j = 1/4 =  0.25
```

3. Calculate total tokens for each class

```
total_class_tokens = count_total_class_tokens(dataset, unique_token,
                     unique_class, tokens_attribute, class_attribute)
```

```
Total tokens in class c:  8

Total tokens in class j:  3
```

4. Count unique occurrence of given token for each class

```
class_unique_token_count = count_class_unique_token(dataset, unique_token,
                        unique_class,unique_test_tokens,tokens_attribute,
                        class_attribute)
```

```
Japan count in class j:  1

Chinese count in class j:  1

Tokyo count in class j:  1

Japan count in class c:  0

Chinese count in class c:  5

Tokyo count in class c:  0
```

5. Calculate hypothesis of each token for each class, formula is explained below

- Likelihood => $P(w|c) = count(w,c) + 1 / count(c) + |V|$
- *count(w,c)*: count the occurrences of specific word in given class
- *count(c)*: count of all the words in given class
- *V*: count of occurrences of set of all words within dataset

```
hypothesis = calculate_hypothesis(class_unique_token_count,
            total_class_tokens, vucablary_count)
```

```
Japan hypothesis in class c = 0 + 1 / (8 + 6) =   0.07142857142857142

Chinese hypothesis in class c = 5 + 1 / (8 + 6) =   0.42857142857142855

Tokyo hypothesis in class c = 0 + 1 / (8 + 6) =   0.07142857142857142
```

```
Japan hypothesis in class j = 1 + 1 / (3 + 6) =   0.2222222222222222

Chinese hypothesis in class j = 1 + 1 / (3 + 6) =   0.2222222222222222

Tokyo hypothesis in class j = 1 + 1 / (3 + 6) =   0.2222222222222222
```

6. Calculate posteriory for each class, , formula is explained below

- o Class Posterior => *P(c|d) = P(w|c) * P(c)*
- o P(w|c): Product of all the hypothesis of words in given class (likelihood)
- o P(c): Probability of give class (prior)

```
class_posterior = calculate_class_posterior(hypothesis, class_probibility, unique_test_tokens,
                        testdata_unique_token_count)
```

```
class c posteriory:  0.0002733236151603498

 class j posteriory:  0.0027434842249657062
```

Step 12: Assign class posterior of each class to the dictionary created earlier

```
classes_posterior[unique_class] = class_posterior
```

```
{'j': 0.0027434842249657062, 'c': 0.0002733236151603498}
```

Step 13: Get maximum posteriority from all the classes present

```
maximum_posterior = max(classes_posterior)
```

```
maximum_posterior:  j
```

Results comparison of Multinomial Naive Baye's with Logistic Regression

Reference code in python is uploaded here:
https://github.com/nabeerdar/naive_bayes/blob/main/comparisson/gender_Classification.ipynb
Reading dataset using pandas builtin function(.read_csv) and checking shape of dataset

```
gender_data = pd.read_csv("persons_gender.csv")
gender_data.shape
```

```
Out[5]: (3233, 8)
```

Viewing features and classes from given data

```
gender_data.head()
```

	long_hair	forehead_width_cm	forehead_height_cm	nose_wide	nose_long	lips_thin	distance_nose_to_lip_long	gender
0	1	11.8	6.1	1	0	1	1	Male
1	0	14.0	5.4	0	0	1	0	Female
2	0	11.8	6.3	1	1	1	1	Male
3	0	14.4	6.1	0	1	1	1	Male
4	1	13.5	5.9	0	0	0	0	Female

Exploratory Data Analysis to check male and female count in given dataset

```
gender_data['gender'].value_counts()
```

```
Out[7]: Male      1783
        Female    1450
        Name: gender, dtype: int64
```

Splitting the gender dataset into training and testing purpose

```
gender_features = gender_data.drop('gender', axis='columns')
gender_classes = gender_data ['gender']

train_features, train_target, test_gender_data_features, test_target = train_test_split(gender_features, gender_classes,
test_size=0.33, stratify = gender_classes)
```

Classification Report with logistic regression on gender data

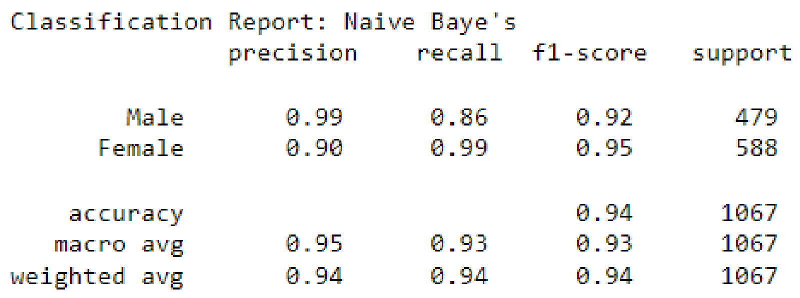
```
clf = LogisticRegression(random_state=0)
clf.fit(train_features, train_target)
target_predict = clf.predict(test_features)
class_names = ['Male', 'Female']
print("Classification Report: Logistic Regression\n", classification_report(test_target, target_predict,
target_names=class_names))
```

```
 Classification Report: Logistic Regression
               precision    recall  f1-score   support

        Male       0.95      0.95      0.95       479
      Female       0.96      0.96      0.96       588

    accuracy                           0.96      1067
   macro avg       0.96      0.96      0.96      1067
weighted avg       0.96      0.96      0.96      1067
```

Classification report with Multinomial Naive Baye’s on gender data

```
clf = MultinomialNB(alpha=0.2)
clf.fit(train_features, train_target)
target_predict = clf.predict(test_features)
class_names = ['Male', 'Female']
```

```
print("Classification Report: Naive Baye’s\n", classification_report(test_target, target_predict,
target_names=class_names))
```



```
Classification Report: Naive Baye's
               precision    recall  f1-score   support

        Male       0.99      0.86      0.92       479
      Female       0.90      0.99      0.95       588

    accuracy                           0.94      1067
   macro avg       0.95      0.93      0.93      1067
weighted avg       0.94      0.94      0.94      1067
```