

Naive Bayes part 2

Spam Email Dataset

We have this email dataset, with the email body, and the first column says whether it is ham or spam, ham means its a good email and not a spam.

And u can see that, wherever there is a spam, in the email body u can see some similar words like free entry or freemsg or Winner, like just by reading it u can say it could be a Spam

Naive Bayes is a family of probabilistic classifiers based on Bayes' theorem with the "naive" assumption of independence between features. Despite its simplicity and the naive assumption, Naive Bayes classifiers often perform well in practice, especially in text classification and spam filtering.

Types of Naive Bayes Classifiers:

1. Multinomial Naive Bayes:

- **Use Case:** Commonly used for document classification tasks, where each document can be represented as a bag-of-words with word frequencies.
- **Example Applications:** Text classification, spam filtering.

2. Gaussian Naive Bayes:

- **Use Case:** Assumes that continuous features follow a Gaussian distribution. Suitable for datasets with continuous numerical features.
- **Example Applications:** Classification tasks where features are continuous and normally distributed.

3. Bernoulli Naive Bayes:

- **Use Case:** Designed for binary or binomial features, often used in text classification where features represent the presence or absence of words.
- **Example Applications:** Text classification, sentiment analysis.

Choosing the Right Naive Bayes Classifier:

- **Multinomial Naive Bayes:**
 - Use when dealing with discrete data, like word counts in text data.
- **Gaussian Naive Bayes:**
 - Use when features follow a Gaussian distribution, often suitable for continuous numerical data.
- **Bernoulli Naive Bayes:**
 - Use when dealing with binary features or presence/absence data.

The choice depends on the nature of your data. If you have text data with word frequencies, Multinomial Naive Bayes might be suitable. If your features are continuous and follow a Gaussian distribution, Gaussian Naive Bayes might be appropriate. If your features are binary, Bernoulli Naive Bayes could be a good choice.

It's often a good idea to try different variants and evaluate their performance on your specific dataset to determine which one works best. Naive Bayes classifiers are known for their simplicity and speed, making them a good choice for quick prototyping and baseline models.

```
In [1]: import pandas as pd
```

```
In [2]: data = pd.read_csv('spam.csv')
data.head()
```

```
Out[2]:
```

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Now, lets do some Data exploration to see wats going on with this data,

First lets just Group it by Category then describing it

```
In [5]: data.groupby('Category').describe()
```

```
Out[5]:
```

	count	unique	Message	top	freq
Category					
ham	4825	4516	Sorry, I'll call later		30
spam	747	641	Please call our customer service representativ...		4

So we can see from above there are 4825 ham and 747 spam so theres a good amount of spam in our dataset

Now, both the columns are text so we have to convert them into Numeric values. The first column is easy to convert into Numeric values since we only have 2 Categories so we can use 1 and 0, so the way to do that is using the **apply()** function.

We use it with Lambda, below it takes the category column and 1 if x is spam else 0 then store it in a new column called spam

```
In [8]: data['spam'] = data['Category'].apply(lambda x: 1 if x == 'spam' else 0)
data.head(10)
```

```
Out[8]:
```

	Category	Message	spam
0	ham	Go until jurong point, crazy.. Available only ...	0
1	ham	Ok lar... Joking wif u oni...	0
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1
3	ham	U dun say so early hor... U c already then say...	0
4	ham	Nah I don't think he goes to usf, he lives aro...	0
5	spam	FreeMsg Hey there darling it's been 3 week's n...	1
6	ham	Even my brother is not like to speak with me. ...	0
7	ham	As per your request 'Melle Melle (Oru Minnamin...	0
8	spam	WINNER!! As a valued network customer you have...	1
9	spam	Had your mobile 11 months or more? U R entitle...	1

As, we can see, where it is spam it put 1 and if its ham it puts 0, instead of this you can also use onehotencoder or pandas dummy variable.

Once this is done, we can do Train Test split method

Train Test split

```
In [9]: from sklearn.model_selection import train_test_split
```

```
In [10]: X_train, X_test, y_train, y_test= train_test_split(data.Message, data.spam, test_size=0.25)
```

SKLearn CountVectorizer

Once this is done, we still have a Message column which is a text so we have to also convert it into Numbers, and the way we can do is using **CountVectorizer** technique

In CountVectorizer technique, lets say you have this 4 documents or the email bodies with all these texts

This is the first document

This document is the second document

And this is the third one.

Is this the first document?

One of the ways to convert these into matrix or vectors is u find out the unique words in each of these documents, and u will find that there are 9 unique words combinely in all these documents

This is the first document

This document is the second document

And this is the third one.

Is this the first document?

and, document, first, is, one, second, the, third, this

Now u can take each of these documents and u can treat this 9 unique words as features or kind of like a Column and u can build this below kind of matrix

This is the first document

This document is the second document

And this is the third one.

Is this the first document?

and, document, first, is, one, second, the, third, this

and	document	first	is	one	second	the	third	this
0	1	1	1	0	0	1	0	1
0	2	0	1	0	1	1	0	1
1	0	0	1	1	0	1	1	1

Each row represent documents, so 1st row represent the first document and so on, so u can see in the first row "and" is 0 meaning the occurance of "and" is 0 in the first document, "document" is one since it appeared once in the first doc, then "first" is also 1 since it occured once in the first doc, and similary in the second row "document" occured twice, and so on...

So this is a simple technique of representing words as counts and we can use these individual columns as feature for our problem,

Below is an example from SKLearn Documentation and it explains the same thing but with the SKLearn API

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> corpus = [
...     'This is the first document.',
...     'This document is the second document.',
...     'And this is the third one.',
...     'Is this the first document?',
... ]
>>> vectorizer = CountVectorizer()
>>> X = vectorizer.fit_transform(corpus)
>>> print(vectorizer.get_feature_names())
['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
>>> print(X.toarray())
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
```

And these are

the API that we are going to use

For our dataset, we use CountVectorizer and it created a matrix which we have seen above

```
In [11]: from sklearn.feature_extraction.text import CountVectorizer
```

```
In [14]: v = CountVectorizer()
X_train_count = v.fit_transform(X_train.values)
X_train_count.toarray()
```

```
Out[14]: array([[0, 2, 0, ..., 0, 0, 0],
               [0, 0, 0, ..., 0, 0, 0],
               [0, 0, 0, ..., 0, 0, 0],
               ...,
               [0, 0, 0, ..., 0, 0, 0],
               [0, 0, 0, ..., 0, 0, 0],
               [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

So it created proolly many features thats why its showing some dots inbetween and these features are equal to Numbers of unique words in our Corpus, Corpus is basically all the unique words that u see in this huge dataset so u realize that it will be many columns

Types of Naive Bayes Classifier

Now Naive Bayes have 3 kind of Classifiers: 1. Bernouli NB 2. Multinomial NB 3. Gaussian NB

In the Naive Bayes Part 1, we used GaussianNB, You can refer below to know the difference between the three

Bernoulli Naive Bayes : It assumes that all our features are binary such that they take only two values. Means **0s** can represent "word does not occur in the document" and **1s** as "word occurs in the document".

Multinomial Naive Bayes : Its is used when we have **discrete data** (e.g. movie ratings ranging 1 and 5 as each rating will have certain **frequency** to represent). In text learning we have the count of each word to predict the class or label.

Gaussian Naive Bayes : Because of the assumption of the **normal distribution**, Gaussian Naive Bayes is used in cases when all our features are **continuous**. For example in **Iris dataset** features are sepal width, petal width, sepal length, petal length. So its features can have different values in data set as width and length can vary. We can't represent features in terms of their occurrences. This means data is continuous. Hence we use Gaussian Naive Bayes here.

We are going to use **Multinomial Naive Bayes** here

```
In [15]: from sklearn.naive_bayes import MultinomialNB
```

```
In [16]: model = MultinomialNB()
```

```
In [18]: model.fit(X_train_count, y_train)
```

```
Out[18]: ▼ MultinomialNB
MultinomialNB()
```

Remember the X_train_count is basically the Message converted into a number metric, Now once the model is trained, it is ready to

make predictions, so lets give it two emails and see

The first email seems to be like a good email, however the second one seems like to be a spam

```
In [19]: emails = [  
        'Hey mohan, can we get together to watch football game tomorrow?',  
        'Upto 20% discount on parking, exclusive offer just for you. Dont miss this reward!'  
        ]  
        emails_count = v.transform(emails)  
        model.predict(emails_count)
```

```
Out[19]: array([0, 1], dtype=int64)
```

And when you run it, you can see that it detected the second email as 1 which is Spam!

Now lets measure the accuracy and the way u do that is first u need to convert X_test into count as well using CountVectorizer

```
In [20]: X_test_count = v.transform(X_test)  
        model.score(X_test_count, y_test)
```

```
Out[20]: 0.9827709978463748
```

You can see our model performed really well! So u can see that for Spam/not Spam type of problems the Naive Bayes model works really great

SKLearn Pipeline

Now we found that converting it into a metric created a lil bit of inconvenient in terms of when we was supplying X_test_count we had to perform this transoform method, and also when we were trying the test emails to predict, we also had to transform it, so SKLearn has a nice feature called **Pipeline** where u can define a Pipeline of ur Transoformation, above what we were doing is, on our row data we are applying some sort of Transformation before fitting it into our model, right now we only used CountVectorizer, some people use more than 1 transoformation like TFIDF and so on, in that case we can use SKLearn Pipeline and its super Convenient.

So first thing u do is import the Pipeline

```
In [21]: from sklearn.pipeline import Pipeline
```

And then u create ur pipeline using the pipeline steps, so our first step is CountVectorizer, so just to remind u what we are doing here is, we are trying to simplify the same codebase so since we have to perform this transformation step everytime, so here we will create a piepleie with 2 steps, so first step is convert my text to vector of CountVectorizer and then apply the MultinomialNB.

```
In [22]: clf = Pipeline([  
        ('vectorizer', CountVectorizer()),  
        ('nb', MultinomialNB())  
        ])
```

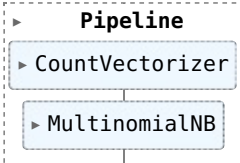
So now when we have a pipeline, we can now train it, but this time we can train X_train directly now,

Remember what was X_train? it has texts inside and in the previous above we used X_train_count as we converted it into count and then trained the model.

Here we can directly fit the text into our model cuz internally inside our pipeline it will convert it to a vector first and then apply Naive Bayes on that

```
In [23]: clf.fit(X_train, y_train)
```

```
Out[23]:
```



```
  ► Pipeline  
  ► CountVectorizer  
  ► MultinomialNB
```

```
In [24]: clf.score(X_test,y_test)
```

```
Out[24]: 0.9827709978463748
```

```
In [25]: clf.predict(emails)
```

```
Out[25]: array([0, 1], dtype=int64)
```

```
In [ ]:
```