Conditional Probability

- Conditional probability as the name suggests, comes into play when the probability of occurrence of a particular event changes when one or more conditions are satisfied (these conditions again are events). Speaking in technical terms, if X and Y are two events then the conditional probability of X w.r.t Y is denoted by P(X|Y). So when we talk in terms of conditional probability, just for an example, we make a statement like "The probability of event X given that Y has already occurred".
- **What if X and Y are independent events?**
  - From the definition of independent events, the occurrence of event X is not dependent on event Y. Therefore P(X|Y)=P(X).
- **What if X and Y are mutually exclusive?**
  - As X and Y are disjoint events, the probability that X will occurs when Y has already occurred is 0. Therefore, P(X|Y)=0
- ** Bayes Theorem**
  - Product rule states that, P(X ∩ Y) = P(X|Y) ∗ P(Y)
  - Product rule states that, P(Y ∩ X) = P(Y|X) ∗ P(X)
    - as P(X ∩ Y) = P(Y ∩ X)
    - P(X|Y) ∗ P(Y) = P(Y|X) ∗ P(X)
    - P(Y|X) = $\frac{P(X|Y)*P(Y)}{P(X)}$

Naive Bayes algorithm

- In Naive Bayes algorithm uses Bayes Theorem. We have to predict Y (the target variable with k - classes), given X (set of $x_1, x_2, .., x_n$ for n - input features). We call this algorithm **Naive** as algorithm assumes that $x_1, x_2, .., x_n$ are independent.
- Naive Bayes algorithm assumes independence amoung features in predecting target class. Hence

$$P(Y_k|x_1, x_2, \ldots, x_n) = \frac{P(x_1, x_2, \ldots, x_n|Y_k)}{P(x_1, x_2, \ldots, x_n)} P(Y_k)$$

  - **As $x_1, x_2, \ldots, x_n$ are independent $P(x_1, x_2, \ldots, x_n|Y_k)$** (likelihood) **can be represented as $P(x_1|Y_k)P(x_2|Y_k)\ldots P(x_n|Y_k)$,** the formula for Naive Bayes algorithm can be represented as below.

$$P(Y_k|x_1, x_2, \ldots, x_n) = \frac{P(x_1|Y_k)P(x_2|Y_k)\ldots P(x_n|Y_k)}{P(x_1, x_2, \ldots, x_n)} P(Y_k)$$

    No need to compute denominator $P(x_1, x_2, \ldots, x_n)$ as we compare probabilities for different classes Y_k.
    - **Proir probability**
      - 1) $P(x_i|Y_k), i = 1, 2, \ldots, n$ - probability of occurrence of x_i given class y_k.
      - 2) $P(Y_k)$ - probability of occurrence of class - Y_k
      - We can derive these two proir probability using given dataset (Training Data Set)
    - **Posterior probability**: By using Naive Bayes algorithm we will determine posterior probability - $P(Y_k|x_1, x_2, \ldots, x_n)$.

Laplacian Correction (Laplacian smoothing)

- Let Y_k be a class in target, w be a word in feature x_i, then maximum likelihood estimator for occurrence of w for class Y_k is

$$P(w/Y_k) = \frac{count(w, Y_k)}{count(Y_k)}$$

- If w is not present in test set, above probability will become 0. To handle this we add 1 to numerator and deominator as shown below.

$$P(w/Y_k) = \frac{count(w, Y_k) + 1}{count(Y_k) + 1}$$

Naive Bayes - Play Tennis Example

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	NO
D2	Sunny	Hot	High	Strong	NO
D3	Overcast	Hot	High	Weak	YES
D4	Rain	Mild	High	Weak	YES
D5	Rain	Cool	Normal	Weak	YES
D6	Rain	Cool	Normal	Strong	NO
D7	Overcast	Cool	Normal	Strong	YES
D8	Sunny	Mild	High	Weak	NO
D9	Sunny	Cool	Normal	Weak	YES
D10	Rain	Mild	Normal	Strong	YES
D11	Sunny	Mild	Normal	Strong	YES
D12	Overcast	Mild	High	Strong	YES
D13	Overcast	Hot	Normal	Weak	YES
D14	Rain	Mild	High	Strong	NO

- We have got a train dataset, with some input features "Day", "Outlook", "Temperature", "Humidity", "Wind" and two classes in output feature "PlayTennis" (Y_k) "YES", "NO"
- We have to predict "PlayTennis" - "YES" or "NO" for below records. Which is nothing but **calculate posterior probabilities** of each class and pick the class which has got more probability

D15	Sunny	Cool	Normal	Weak	?
D16	Rain	Mild	Normal	Strong	?

Prior Probabilities of Y_k - ('YES', 'NO')

- Probability(Play Tennis) = P(play) = 9/14
- Probability(Not Play Tennis) = P(Not-Play) = 5/14

To calculate posterior probability we have to follow three steps

- 1) Constructing a **frequency table** for each attribute against the target.

Frequency Table		PlayTennis	
		YES	NO
Outlook	Sunny	2	3
	Overcast	4	0
	Rain	3	2

Frequency Table		PlayTennis	
		YES	NO
Temperature	Hot	2	2
	Mild	4	2
	Cool	3	1

Frequency Table		PlayTennis	
		YES	NO
Humidity	High	3	4
	Normal	6	1

Frequency Table		PlayTennis	
		YES	NO
Wind	Weak	5	2
	Strong	4	3

- 2) Transforming the frequency tables to **likelihood tables**.

Likelihood Table		PlayTennis	
		YES	NO
Outlook	Sunny	2/9.	3/5.
	Overcast	4/9.	0/5.
	Rain	3/9.	2/5.

Likelihood Table		PlayTennis	
		YES	NO
Temperature	Hot	2/9.	2/5.
	Mild	4/9.	2/5.
	Cool	3/9.	1/5.

Likelihood Table		PlayTennis	
		YES	NO
Humidity	High	3/9.	4/5.
	Normal	6/9.	1/5.

Likelihood Table		PlayTennis	
		YES	NO
Wind	Weak	5/9.	2/5.
	Strong	4/9.	3/5.

- 3) Use the Naive Bayesian equation to calculate the **posterior probability** for each class. The class with the highest posterior probability is the outcome of prediction.
-

$$P(YES|[Outlook = Sunny], [Temperature = Cool], [Humidity = Normal], [$$
$$= \frac{P(Outlook = Sunny|YES) * P(Temperature = Cool|YES) * P(Humidity = Normal|YES) *}{P([Outlook = Sunny], [Temperature = Cool], [Humidity = Normal], [Wind =}$$
$$= \frac{2/9 * 3/9 * 6/9 * 5/9 * 9/14}{P([Outlook = Sunny], [Temperature = Cool], [Humidity = Normal], [Wind = Week])}$$

- $$P(NO|[Outlook = Sunny], [Temperature = Cool], [Humidity = Normal], [Wi$$
$$= \frac{P(Outlook = Sunny|NO) * P(Temperature = Cool|NO) * P(Humidity = Normal|NO) * P([}{P([Outlook = Sunny], [Temperature = Cool], [Humidity = Normal], [Wind = V}$$
$$= \frac{3/5 * 1/5 * 1/5 * 2/5 * 5/14}{P([Outlook = Sunny], [Temperature = Cool], [Humidity = Normal], [Wind = Week])}$$

- In above two poserior probabilities, the denominator is same, there is no need of calculating it as we have to compare these two probabilities.
- Hence P(YES | X) approximately equal to 0.017636684
- P(NO | X) is approximately equal to 0.0034285714285714
- **P(YES | X) is greater than P(NO | X), hence the prediction is PlayTennis = YES**

Variations of the Naive Bayes algorithm

- There are multiple variations of the Naive Bayes algorithm depending on the distribution of P(x_j|C_i). Three of the commonly used variations are
  - **Gaussian:** The Gaussian Naive Bayes algorithm assumes distribution of features to be Gaussian or normal, i.e., $P(x_j|C_i) = \frac{1}{\sqrt{2\pi\sigma_{C_i}^2}}\exp\left(-\frac{(x_j-\mu_{C_j})^2}{2\sigma_{C_i}^2}\right)$ Read more about it here (http://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes).
  - **Multinomial:** The Multinomial Naive Bayes algorithm is used when the data is distributed multinomially, i.e., multiple occurrences matter a lot. You can read more here (http://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes).
  - **Bernoulli:** The Bernoulli algorithm is used when the features in the data set are binary-valued. It is helpful in spam filtration and adult content detection techniques. For more details, click here (http://scikit-learn.org/stable/modules/naive_bayes.html#bernoulli-naive-bayes).

Pros and Cons of Naive Bayes algorithm

- **Pros**
  - It is a relatively easy algorithm to build and understand.
  - It is faster to predict classes using this algorithm than many other classification algorithms.
  - It can be easily trained using a small data set.
- **Cons**
  - **Zero Conditional Probability Problem :** If a given **class and a feature have 0 frequency**, then the **conditional probability estimate for that category will come out as 0**. This problem is known as the **"Zero Conditional Probability Problem"**. This is a problem because it wipes out all the information in other probabilities too. There are several sample correction techniques to fix this problem such as **"Laplacian Correction"**.

- Another disadvantage is the very strong assumption of independence class features that it makes. It is near to impossible to find such data sets in real life.

Text Analytics - Bag-of-Words model

- **Stop Words** are words which are filtered out before or after processing of natural language data (text). Though "stop words" usually refers to the most common words in a language, there is no single universal list of stop words used by all natural language processing tools, and indeed not all tools even use such a list. Some tools specifically avoid removing these stop words to support phrase search.
- **Tokenization** Given a character sequence and a defined document unit, tokenization is the task of chopping it up into pieces, called tokens , perhaps at the same time throwing away certain characters, such as punctuation. Here is an example of tokenization:
  - Input: Friends, Romans, Countrymen, lend me your ears;
  - Output: **Friends, Romans, Countrymen, lend, me, your, ears**
- **Stemming** algorithms work by cutting off the end of the word, and in some cases also the beginning while looking for the root. This indiscriminate cutting can be successful in some occasions, but not always, that is why we affirm that this is an approach that has some limitations.
  - The best-known and most popular stemming approach for English is the Porter stemming algorithm, also known as the Porter stemmer. It is a collection of rules (or, if you prefer, heuristics) designed to reflect how English handles inflections. **For example, the Porter stemmer chops both apple and apples down to appl, and it stems berry and berries to berri.**
- **Lemmatization** on the other hand takes into consideration the morphological analysis of the words. To do so it is necessary to have detailed dictionaries the algorithm can look back at to link the form back to its lemma.
  - Lemmatization does not simply chop off inflections, but instead relies on a lexical knowledge base like , WordNet (https://wordnet.princeton.edu/) to obtain the correct base forms of words.
  - For example, WordNet lemmatizes **geese** to **goose**, **meanness to meanness** and **meaning to meaning**. In these examples, it outperforms than the Porter stemmer.
- **Stemming VS Lemmatization :** Lemmatization or Stemming has limits. For example, Porter stems both happiness and happy to happi, while WordNet lemmatizes the two words to themselves. The WordNet lemmatizer also requires specifying the word's part of speech—otherwise, it assumes the word is a noun. Finally, lemmatization cannot handle unknown words: for example, Porter stems both iphone and iphones to iphon, while WordNet lemmatizes both words to themselves. In general, lemmatization offers better precision than stemming, but at the expense of recall.

In [16]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [17]:

```
reviews = pd.read_csv('../data/Restaurant_Reviews.tsv', delimiter = '\t', quoting = 3)
```

In [18]:

```
reviews.shape
```

Out[18]:

```
(1000, 2)
```

In [19]:

```
reviews.head()
```

Out[19]:

	Review	Liked
0	Wow... Loved this place.	1
1	Crust is not good.	0
2	Not tasty and the texture was just nasty.	0
3	Stopped by during the late May bank holiday of...	1
4	The selection on the menu was great and so wer...	1

In [25]:

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
```

In [6]:

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\thisi\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[6]:

```
True
```

In [26]:

```
print(type(stopwords.words('english')))
```

```
<class 'list'>
```

In [27]:

```
my_stop_words = stopwords.words('english')
```

In [29]:

```
my_stop_words.remove('not')
```

In [30]:

```
print(my_stop_words)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

In [31]:

```
corpus = []
for i in range(0, 1000):
    review = re.sub('[^a-zA-Z]', ' ', reviews['Review'][i]) #substitute non alphabets with
    review = review.lower()
    review = review.split()
    ps = PorterStemmer()
    review = [ps.stem(word) for word in review if not word in set(my_stop_words)]
    review = ' '.join(review)
    corpus.append(review)
```

In [ ]:

```

```

In [32]:

```
corpus
```

Out[32]:

```
['wow love place',
 'crust not good',
 'not tasti textur nasti',
 'stop late may bank holiday rick steve recommend love',
 'select menu great price',
 'get angri want damn pho',
 'honeslti tast fresh',
 'potato like rubber could tell made ahead time kept warmer',
 'fri great',
 'great touch',
 'servic prompt',
 'would not go back',
 'cashier care ever say still end wayyy overpr',
 'tri cape cod ravoli chicken cranberri mmmm',
 'disgust pretti sure human hair',
 'shock sign indic cash',
 'highli recommend',
 'waitress littl slow servic',
```

In [49]:

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features = 20)
X = cv.fit_transform(corpus).toarray()
y = reviews.iloc[:, 1].values
```

In [50]:

```
cv.vocabulary_   #A mapping of terms to feature indices.
```

Out[50]:

```
{'love': 10,
 'place': 13,
 'not': 11,
 'good': 7,
 'great': 8,
 'get': 5,
 'like': 9,
 'time': 17,
 'servic': 16,
 'would': 19,
 'go': 6,
 'back': 0,
 'ever': 3,
 'food': 4,
 'restaur': 15,
 'order': 12,
 'realli': 14,
 'wait': 18,
 'disappoint': 2,
 'best': 1}
```

In [52]:

```
X[:5, : ]
```

Out[52]:

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

In [53]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20, random_state =
```

In [54]:

```
X_train.shape
```

Out[54]:

```
(800, 20)
```

In [55]:

```
X_test.shape
```

Out[55]:

```
(200, 20)
```

In [56]:

```
from sklearn.naive_bayes import BernoulliNB
classifier = BernoulliNB()
classifier.fit(X_train, y_train)
```

Out[56]:

```
BernoulliNB()
```

In [57]:

```
y_pred = classifier.predict(X_test)
```

In [58]:

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

In [59]:

```
cm
```

Out[59]:

```
array([[39, 58],
       [19, 84]], dtype=int64)
```

In [ ]:

```

```