SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

RAMAPURAM CAMPUS

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

ACADEMIC YEAR 2021-22(EVEN SEMESTER)

18CSE479T - Statistical Machine Learning

Unit - III

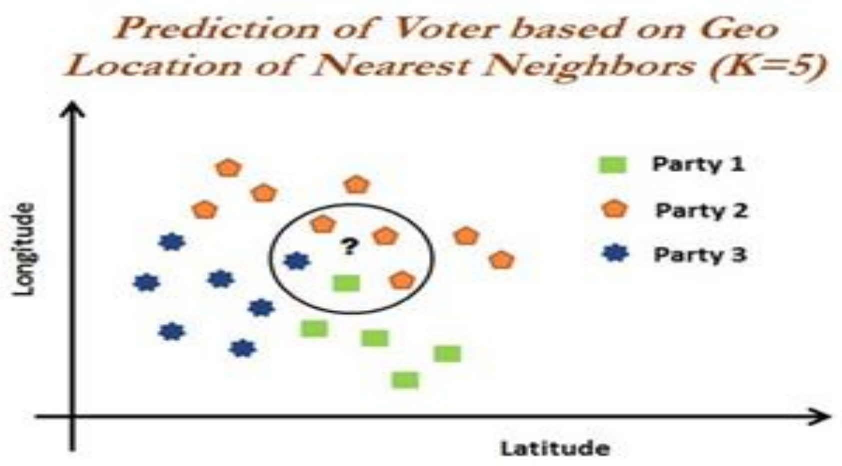
K-nearest neighbors-KNN voter example -Curse of dimensionality-Curse of dimensionality with 1D, 2D, and 3D example-Curse of dimensionality with 3D example-KNN classifier with breast cancer Wisconsin data example.Naive Bayes-Probability fundamentals - Joint probability-Understanding Bayes theorem with conditional probability-Naive Bayes classification -Laplace estimator-Naive Bayes SMS spam classification example

I K-nearest neighbors

- Definition - K-nearest neighbors is a **non-parametric machine learning model** in which the model memorizes the training observation for classifying the unseen test data.
- It is also called as **Instance-based learning**
- This model is often termed as **lazy learning**, as it does not learn anything during the training phase like regression, random forest, and so on.
- It starts working only during the testing/evaluation phase to compare the given test observations with nearest training observations

II KNN voter example

- Objective is to predict the party for which voter will vote based on their neighborhood, precisely geolocation (latitude and longitude).
- Assumption - Identify the potential voter to which political party they would be voting based on majority voters did voted for that particular party.
- Tuning the k-value (number to consider, among which majority should be counted) is the million- dollar question (as same as any machine learning algorithm)


Prediction of Voter based on Geo Location of Nearest Neighbors (K=5)
Party 1
Party 2
Party 3
?
Longitude
Latitude

- In the preceding diagram, we can see that the voter of the study will vote for **Party 2**.
- As within the vicinity, one neighbor has voted for **Party 1** and the other voter voted for P**arty 3**.
- But three voters voted for **Party 2**.
- In fact, by this way KNN solves any given classification problem.
- Regression problems are solved by taking mean of its neighbors within the given circle or vicinity or k-value

III Curse of dimensionality

Curse of dimensionality

- The curse of dimensionality refers to the phenomena that occur when classifying, organizing, and analyzing high dimensional data that does not occur in low dimensional spaces, specifically the issue of data sparsity and “closeness” of data.
- Points in high-dimensional spaces tend to be dispersing from each other more compared with the points in low-dimensional space.
- uniform random values between zero and one generated for 1D, 2D, and 3D space to validate this hypothesis

Example

- In the following code, mean distance between 1,000 observations have been calculated with the change in dimensions.
- It is apparent that with the increase in dimensions, distance between points increases logarithmically, which gives us the hint that we need to have exponential increase in data points with increase in dimensions in order to make machine learning algorithms work correctly

```
>>> import numpy as np
>>> import pandas as pd
>>> import random,math
```

- The following code generates random numbers between zero and one from uniformdistribution with the given dimension, which is equivalent of length of array or list:

```
>>> def random_point_gen(dimension):
...         return [random.random() for _ in range(dimension)]
```

- The following function calculates root mean sum of squares of Euclidean distances (2-norm) between points by taking the difference between points and sum the squares and finally takes square root of total distance:

```
>>> def distance(v,w):
...         vec_sub = [v_i-w_i for v_i,w_i in zip(v,w)]
...         sum_of_sqrs = sum(v_i*v_i for v_i in vec_sub)
...         return math.sqrt(sum_of_sqrs)
```

- Both dimension and number of pairs are utilized for calculating the distances with the following code:

```
>>> def random_distances_comparison(dimension,number_pairs):
... return[distance(random_point_gen(dimension),random_point_gen(dimension))
for _ in range(number_pairs)]
>>> def mean(x):
...                 return sum(x) / len(x)
```

- changing dimensions from 1 to 201 with the increase of 5 dimensions to check the increase in distance:

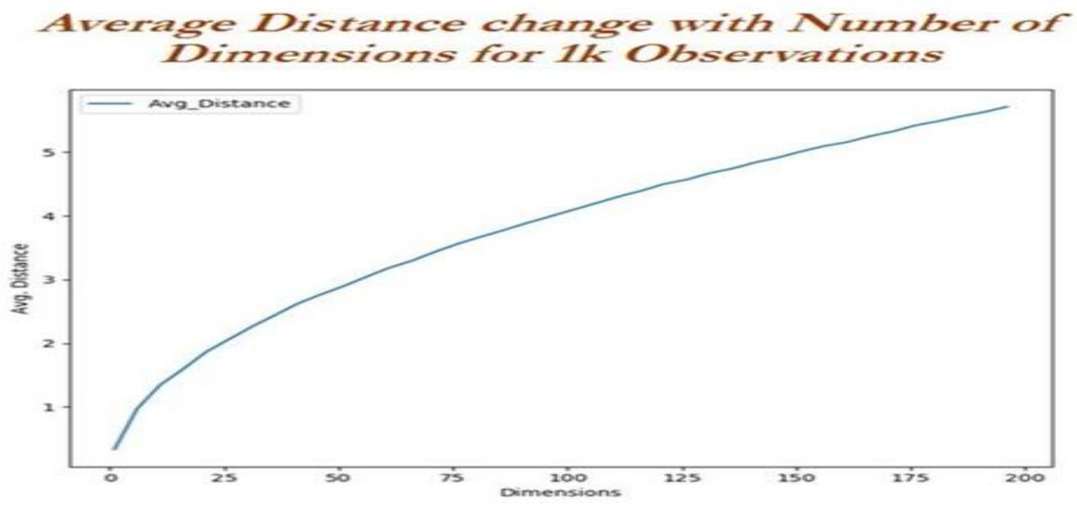
```
>>> dimensions = range(1, 201, 5)
```

- Both minimum and average distances have been calculated to check,

```
>>> avg_distances = []
>>> min_distances = []
>>> dummyarray = np.empty((20,4))
>>> dist_vals = pd.DataFrame(dummyarray)
>>> dist_vals.columns = ["Dimension","Min_Distance","Avg_Distance","Min/Avg_Distance”]
>>> random.seed(34)
>>> i = 0
>>> for dims in dimensions:
...     distances = random_distances_comparison(dims, 1000)
...     avg_distances.append(mean(distances))
...     min_distances.append(min(distances))
...     dist_vals.loc[i,"Dimension"] = dims
...     dist_vals.loc[i,"Min_Distance"] = min(distances)
...     dist_vals.loc[i,"Avg_Distance"] = mean(distances)
...     dist_vals.loc[i,"Min/Avg_Distance"] = min(distances)/mean(distances)
...     print(dims, min(distances), mean(distances), min(distances)*1.0 / mean( distances))
...     i = i+1
```

- **Plotting Average distances for Various Dimensions**

```
>>> import matplotlib.pyplot as plt
>>> plt.figure()
>>> plt.xlabel('Dimensions')
>>> plt.ylabel('Avg. Distance')
>>> plt.plot(dist_vals["Dimension"],dist_vals["Avg_Distance"])
>>> plt.legend(loc='best')
>>> plt.show()
```



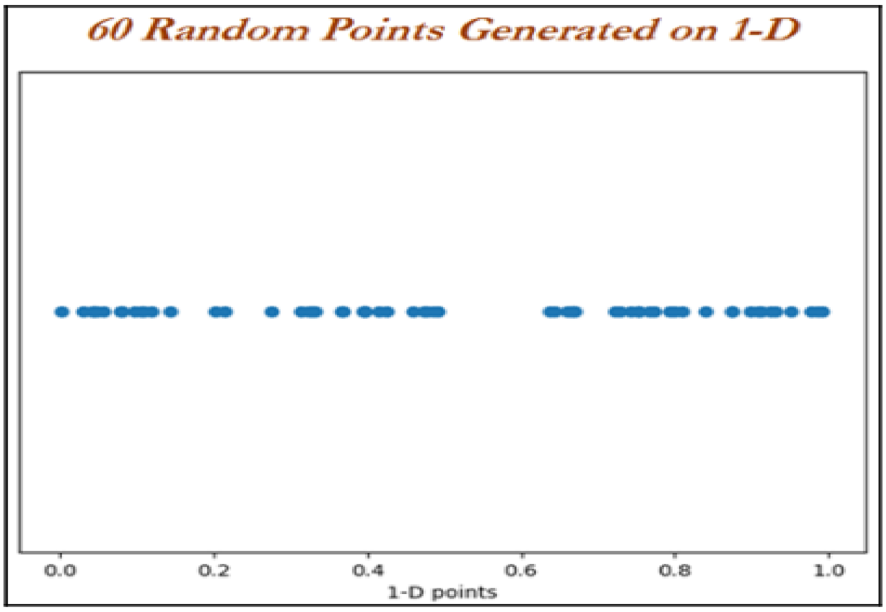
- From the graph, it is proved that with the increase in dimensions, mean distance increases logarithmically.
- Hence the higher the dimensions, the more data is needed to overcome the curse of dimensionality!

IV Curse of dimensionality with 1D

- distance of 60 random points are expanding with the increase in dimensionality.

# 1-Dimension Plot

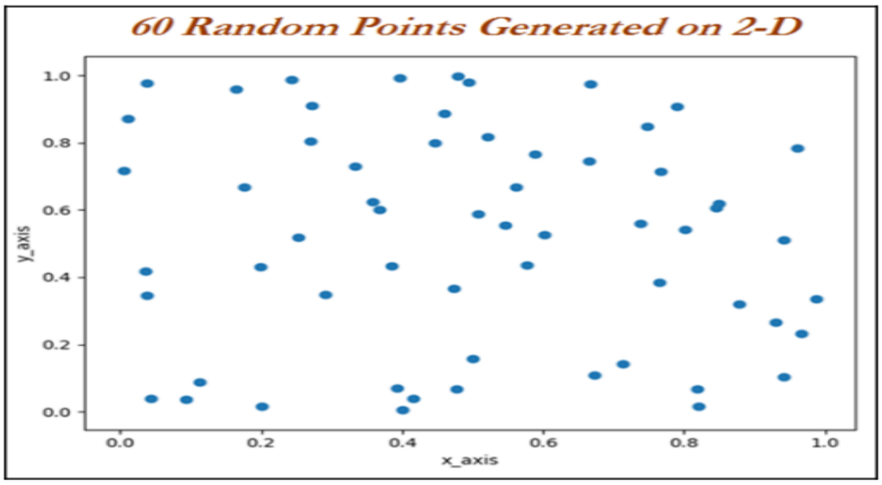
```
>>> import numpy as np
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>> one_d_data = np.random.rand(60,1)
>>> one_d_data_df = pd.DataFrame(one_d_data)
>>> one_d_data_df.columns = ["1D_Data"]
>>> one_d_data_df["height"] = 1
>>> plt.figure()
>>> plt.scatter(one_d_data_df['1D_Data'],one_d_data_df["height"])
>>> plt.yticks([])
>>> plt.xlabel("1-D points")
>>> plt.show()
```



V Curse of dimensionality with 2D

- 60 random numbers with x and y co-ordinate space

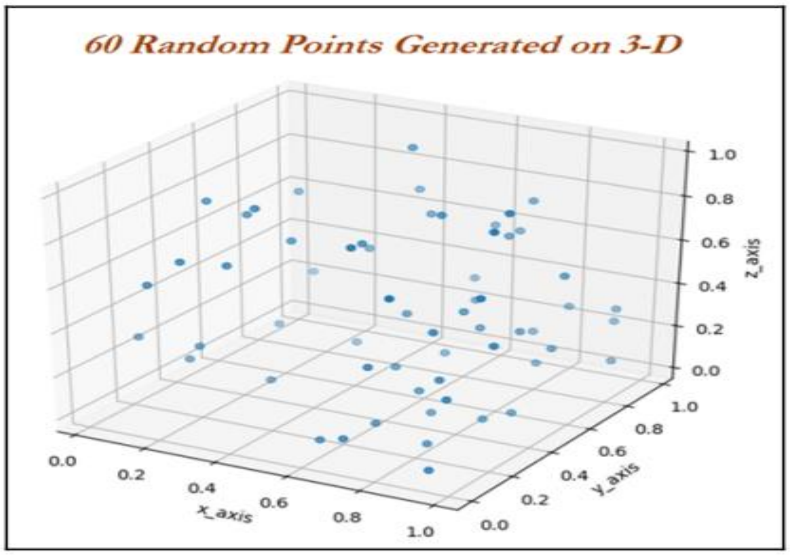
```
# 2- Dimensions Plot
>>> two_d_data = np.random.rand(60,2)
>>> two_d_data_df = pd.DataFrame(two_d_data)
>>> two_d_data_df.columns = ["x_axis","y_axis"]
 >>> plt.figure()
>>> plt.scatter(two_d_data_df['x_axis'],two_d_data_df["y_axis"])
>>> plt.xlabel("x_axis");plt.ylabel("y_axis")
>>> plt.show()
```



VI Curse of dimensionality with 3D example

- 60 data points are drawn for 3D space
- increase in spaces, which is very apparent.
- This has proven to us visually that with the increase in dimensions, it creates lot of space, which makes a classifier weak to detect the signal

```
# 3- Dimensions Plot
>>> three_d_data = np.random.rand(60,3)
>>> three_d_data_df = pd.DataFrame(three_d_data)
>>> three_d_data_df.columns = ["x_axis","y_axis","z_axis"]
 >>> from mpl_toolkits.mplot3d import Axes3D
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111, projection='3d')
>>>
ax.scatter(three_d_data_df['x_axis'],three_d_data_df["y_axis"],three_d_data
_df ["z_axis"])
>>> plt.show()
```



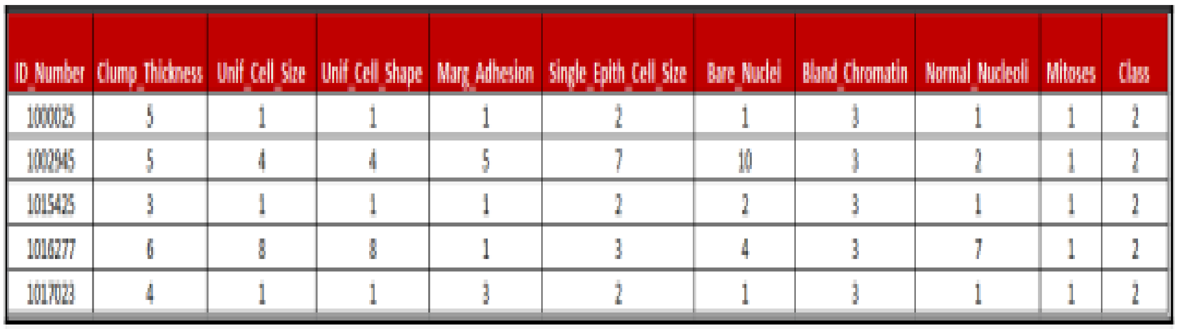
VII KNN classifier with breast cancer Wisconsin data example

Breast cancer data has been utilized from the UCI machine learning repository http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

To find whether the cancer is malignant or benign based on various collected features such as clump thickness and so on using the KNN classifier

```
# KNN Classifier - Breast Cancer
>>> import numpy as np
>>> import pandas as pd
>>> from sklearn.metrics import accuracy_score,classification_report
>>> breast_cancer = pd.read_csv("Breast_Cancer_Wisconsin.csv")
```

- The Class value has class 2 and 4. Value 2 and 4 represent benign and malignant class, respectively.
- Whereas all the other variables do vary between value 1 and 10, which are very much categorical in nature



ID_Number	Clump_Thickness	Unif_Cell_Size	Unif_Cell_Shape	Marg_Adhesion	Single_Epith_Cell_Size	Bare_Nuclei	Bland_Chromatin	Normal_Nucleoli	Mitoses	Class
1000025	5	1	1	1	2	1	3	1	1	2
1002945	5	4	4	5	7	10	3	2	1	2
1015425	3	1	1	1	2	2	3	1	1	2
1016277	6	8	8	1	3	4	3	7	1	2
1017023	4	1	1	3	2	1	3	1	1	2

- the Bare_Nuclei variable has some missing values, here we are replacing them with the most frequent value (category value 1) in the following code:

  ```
  >>>breast_cancer['Bare_Nuclei']= breast_cancer['Bare_Nuclei'].replace('?', np.NAN)
  >>>breast_cancer['Bare_Nuclei'] = breast_cancer['Bare_Nuclei'].fillna(breast_cancer[
  'Bare_Nuclei'].value_counts().index[0])
  ```

- Use the following code **to convert the classes to a 0 and 1** indicator for using in the classifier:

  ```
  >>> breast_cancer['Cancer_Ind'] = 0
  >>>breast_cancer.loc[breast_cancer['Class']==4,'Cancer_Ind '] = 1
  ```

- we are dropping non-value added variables from analysis:

  ```
  >>> x_vars = breast_cancer.drop(['ID_Number','Class','Cancer_Ind'],axis=1)
  >>> y_var = breast_cancer['Cancer_Ind']
  >>> from sklearn.preprocessing import StandardScaler
  >>> x_vars_stdscle = StandardScaler().fit_transform(x_vars.values)
  >>> from sklearn.model_selection import train_test_split
  ```

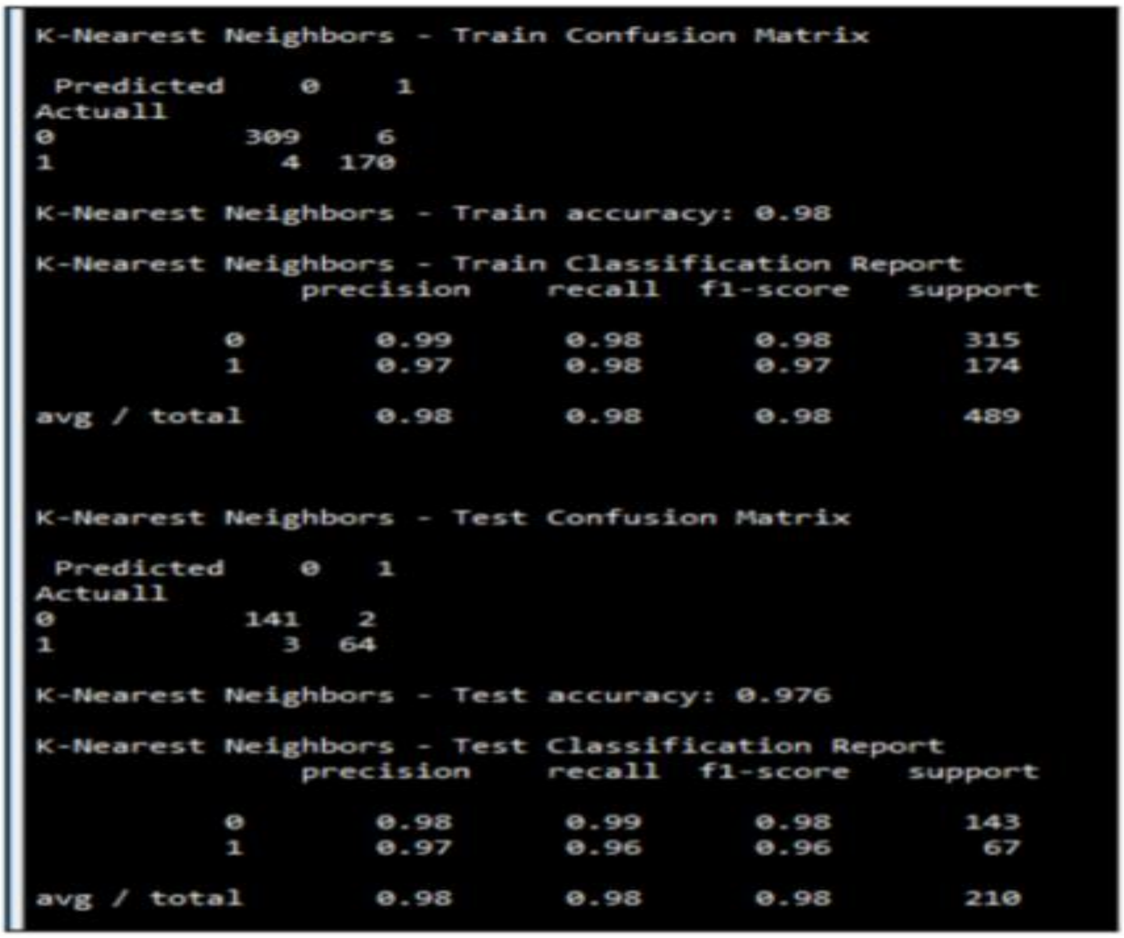
- As KNN is very sensitive to distances, here we are standardizing all the columns before applying algorithms:

  ```
  >>> x_vars_stdscle_df = pd.DataFrame(x_vars_stdscle, index=x_vars.index,
  columns=x_vars.columns)
  >>> x_train,x_test,y_train,y_test =
  train_test_split(x_vars_stdscle_df,y_var, train_size = 0.7,random_state=42)
  ```

- KNN classifier is being applied with neighbor value of 3 and p value indicates it is 2-norm, also known as Euclidean distance for computing classes:

```
>>> from sklearn.neighbors import KNeighborsClassifier
>>> knn_fit = KNeighborsClassifier(n_neighbors=3,p=2,metric='minkowski')
```

```
>>> knn_fit.fit(x_train,y_train)
>>> print ("\nK-Nearest Neighbors - Train Confusion Matrix\n\n",pd.crosstab(y_train, knn_fit.predict(x_train),rownames = ["Actuall"],colnames = ["Predicted"]) )
>>> print ("\nK-Nearest Neighbors - Train accuracy:",round(accuracy_score(y_train, knn_fit.predict(x_train)),3))
>>> print ("\nK-Nearest Neighbors - Train Classification Report\n", classification_report( y_train,knn_fit.predict(x_train)))
>>> print ("\n\nK-Nearest Neighbors - Test Confusion Matrix\n\n",pd.crosstab(y_test, knn_fit.predict(x_test),rownames = ["Actuall"],colnames = ["Predicted"]))
>>> print ("\nK-Nearest Neighbors - Test accuracy:",round(accuracy_score( y_test,knn_fit.predict(x_test)),3))
>>> print ("\nK-Nearest Neighbors - Test Classification Report\n", classification_report(y_test,knn_fit.predict(x_test)))
```



```
K-Nearest Neighbors - Train Confusion Matrix

 Predicted     0     1
Actuall
0            309     6
1              4   170

K-Nearest Neighbors - Train accuracy: 0.98

K-Nearest Neighbors - Train Classification Report
              precision    recall  f1-score   support

          0       0.99      0.98      0.98       315
          1       0.97      0.98      0.97       174

avg / total       0.98      0.98      0.98       489


K-Nearest Neighbors - Test Confusion Matrix

 Predicted     0   1
Actuall
0            141   2
1              3  64

K-Nearest Neighbors - Test accuracy: 0.976

K-Nearest Neighbors - Test Classification Report
              precision    recall  f1-score   support

          0       0.98      0.99      0.98       143
          1       0.97      0.96      0.96        67

avg / total       0.98      0.98      0.98       210
```

- From the results, it is appearing that KNN is working very well in classifying malignant and benign classes well
- Obtains test accuracy of 97.6 percent with 96 percent of recall on malignant class.

- The only deficiency of KNN classifier would be, it is computationally
- intensive during test phase, as each test observation will be compared with all the available observations in train data, which practically KNN does not learn a thing from training data. Hence, we are also calling it a lazy classifier!

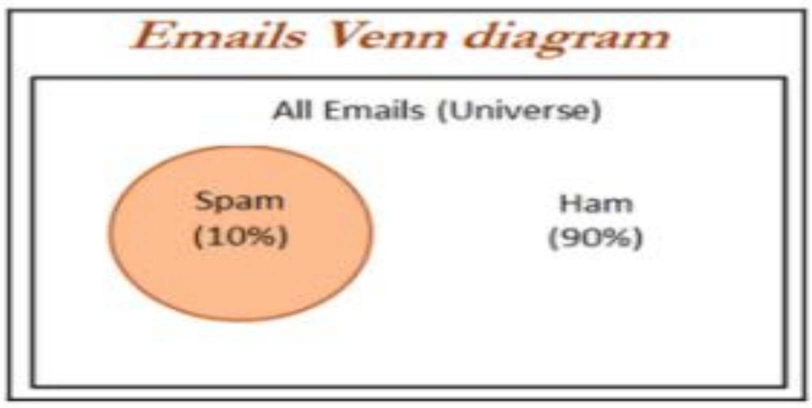
VIII Naive Bayes - Introduction

- The Naive Bayes is a **classification algorithm** that is suitable for **binary and multiclass classification.**
- Naïve Bayes performs well in cases of **categorical input variables** compared to numerical variables.
- It is useful for making predictions and forecasting data based on historical results.
- The distinction between Bayes theorem and Naive Bayes is that **Naive Bayes assumes conditional independence** where **Bayes theorem does not.**
- This means **the relationship between all input features are independent**. Maybe not a great assumption, but this is is why the algorithm is called "naive
- **Thomas** developed the foundational mathematical principles for **determining the probability of unknown events from the known events.**
- For example, if all apples are red in color and average diameter would be about 4 inches then, if at random one fruit is selected from the basket with red color and diameter of 3.7 inch, what is the probability that the particular fruit would be an apple?
- **Naive term does assume independence of particular features in a class with respect to others**.
- In this case, there would be no dependency between color and diameter.
- This independence assumption makes the Naive Bayes classifier most effective in terms of computational ease for particular tasks
- **Bayesian classifiers are best applied to problems in which information from a very high number of attributes should be considered simultaneously to estimate the probability of final outcome.**

IX Probability fundamentals

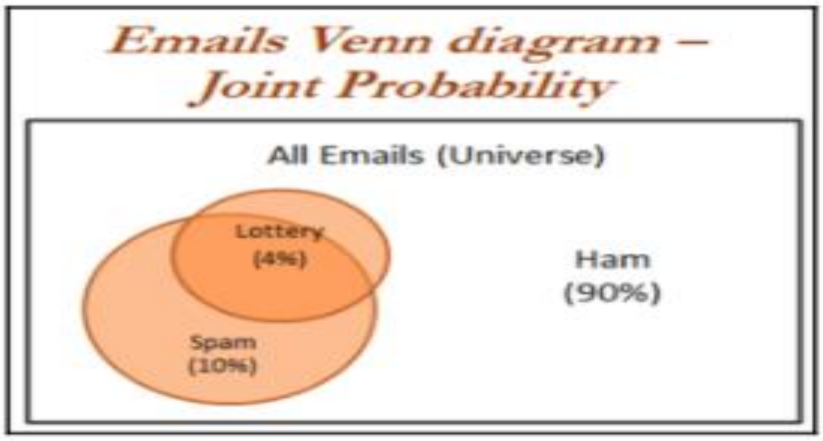
- Probability of an event can be estimated from observed data by dividing the number of trails in which an event occurred with total number of trails.

- For instance, if a bag contains red and blue balls and randomly picked 10 balls one by one with replacement and out of 10, 3 red balls appeared in trails we can say that
- probability of red is 0.3, pred = 3/10 = 0.3.
- Total probability of all possible outcomes must be 100 percent.
- **If a trail has two outcomes such as email classification either it is spam or ham and both cannot occur simultaneously, these events are considered as mutually exclusive with each other.**
- In addition, if those outcomes cover all possible events, it would be called as **exhaustive events**.
- For example, in email classification if *P (spam) = 0.1*, we will be able to calculate *P (ham) = 1- 0.1 = 0.9*, these two events are mutually exclusive

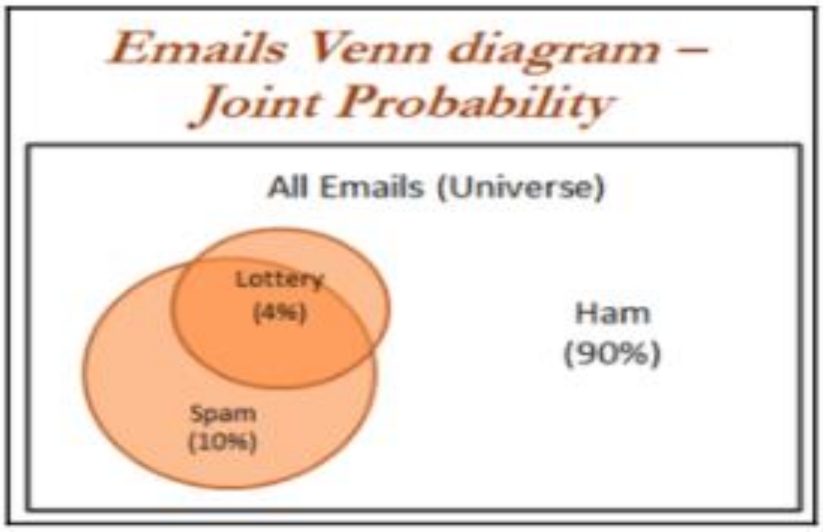


X Joint probability

- Though mutually exclusive cases are simple to work upon, most of the actual problems do fall under the category of non-mutually exclusive events.
- **By using the joint appearance, we can predict the event outcome.**
- For example, if emails messages present the word like lottery, which is very highly likely of being spam rather than ham.
- The spam messages contain the word lottery and not every email with the word lottery is spam.



- the joint probability of both *p(spam)* and *p(lottery)* occurring, which can be written as *p(spam ∩ lottery)*.
- In case if both the events are totally unrelated, they are called **independent events** and their respective value is *p(spam ∩ lottery) = p(spam) * p(lottery) = 0.1 * 0.04 = 0.004*, which is 0.4 percent of all messages are spam containing the word Lottery.
- In general, for independent events *P(A∩ B) = P(A) * P(B)*.

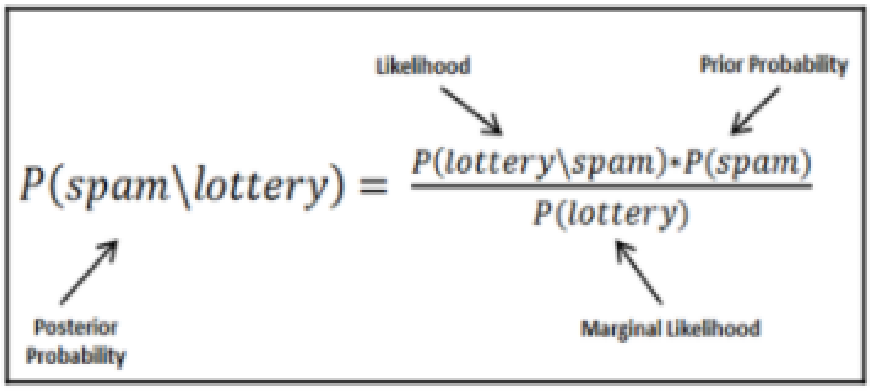


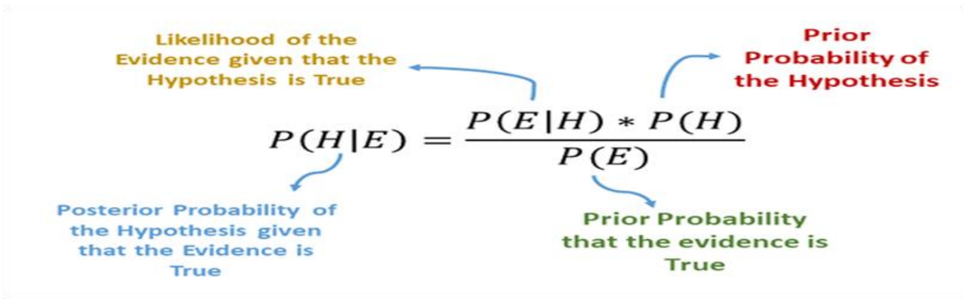
XI Understanding Bayes theorem with conditional probability

- **Conditional probability provides a way of calculating relationships between dependent events using Bayes theorem.**
- For example, A and B are two events and we would like to calculate **P(A\B) can be read as the probability of event occurring A given the fact that event B already occurred**, in fact this is known as conditional probability,
- The equation can be written as follows:

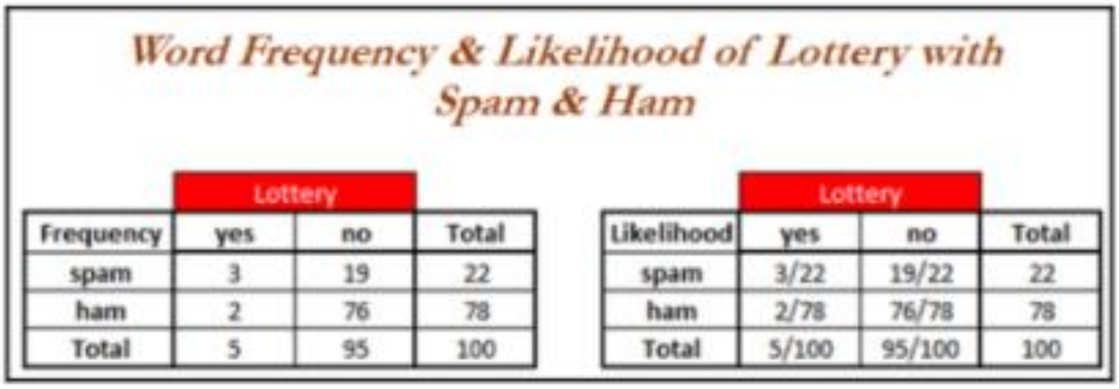
$$P(A\backslash B) = \frac{P(B\backslash A) * P(A)}{P(B)} = \frac{P(A \cap B)}{P(B)}$$

- Email classification example - Objective is to predict whether email is spam given the word lottery and some other clues.
- In this case we already knew the overall probability of spam, which is 10 percent also known as prior probability.
- Now suppose you have obtained an additional piece of information that probability of word lottery in all messages, which is 4 percent, also known as marginal likelihood.
- Now, we know the probability that lottery was used in previous spam messages and is called the likelihood.


Likelihood
Prior Probability
$P(spam\backslash lottery) = \frac{P(lottery\backslash spam)*P(spam)}{P(lottery)}$
Posterior Probability
Marginal Likelihood


Likelihood of the Evidence given that the Hypothesis is True
Prior Probability of the Hypothesis
$P(H|E) = \frac{P(E|H) * P(H)}{P(E)}$
Posterior Probability of the Hypothesis given that the Evidence is True
Prior Probability that the evidence is True

- By applying the Bayes theorem to the evidence, we can calculate the posterior probability that calculates the probability that the message is how likely being a spam; given the fact that lottery was appearing in message.
- On average if the probability is greater than 50 percent it indicates that the message is spam rather than ham.



Word Frequency & Likelihood of Lottery with Spam & Ham

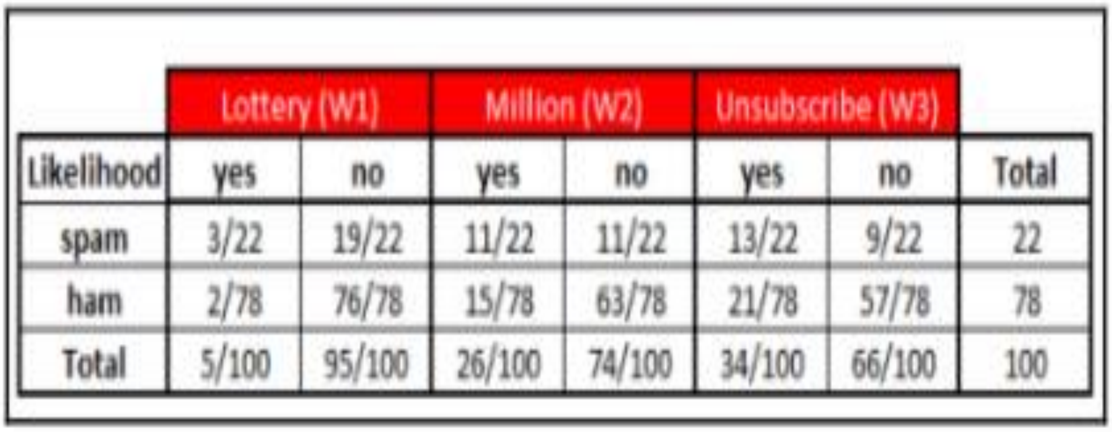
	Lottery		
Frequency	yes	no	Total
spam	3	19	22
ham	2	76	78
Total	5	95	100

	Lottery		
Likelihood	yes	no	Total
spam	3/22	19/22	22
ham	2/78	76/78	78
Total	5/100	95/100	100

- sample frequency table that records the number of times *Lottery* appeared in spam and ham messages and its respective likelihood has been shown.
- **Likelihood table reveals that *P(Lottery\Spam)*= *3/22 = 0.13*,** indicating that probability is 13 percent that a spam message contains the term *Lottery*.
- Subsequently we can calculate the ***P(Spam ∩ Lottery) = P(Lottery\Spam) * P(Spam) = (3/22) * (22/100) = 0.03*.**
- In order to **calculate the posterior probability, we divide *P(Spam ∩ Lottery)* with *P(Lottery)*, which means *(3/22)*(22/100) / (4/100) = 0.75***
- Therefore, the probability is 75 percent that a message is spam, given that message contains the word *Lottery*.

XII Naive Bayes classification

- Let us construct the likelihood table for the appearance of the three words (W1, W2, and W3), as shown in the following table for 100 emails:



	Lottery (W1)		Million (W2)		Unsubscribe (W3)		
Likelihood	yes	no	yes	no	yes	no	Total
spam	3/22	19/22	11/22	11/22	13/22	9/22	22
ham	2/78	76/78	15/78	63/78	21/78	57/78	78
Total	5/100	95/100	26/100	74/100	34/100	66/100	100

- When a new message is received, the posterior probability will be calculated to determine that email message is spam or ham. Let us assume that we have an email with terms *Lottery* and *Unsubscribe*, but it does not have word *Million* in it, with this details, what is the probability of spam?
- By using Bayes theorem, we can define the problem as *Lottery = Yes*, *Million = No* and *Unsubscribe = Yes*:
- Solving the preceding equations will have high computational complexity due to the dependency of words with each other. As more number of words are added, this will even explode and also huge memory will be needed for processing all possible intersecting events. This finally leads to intuitive turnaround with independence of words (**cross- conditional independence**)

$$P(Spam|W_1 \cap \neg W_2 \cap W_3) = \frac{P(W_1 \cap \neg W_2 \cap W_3 \mid Spam) * P(Spam)}{P(W_1 \cap \neg W_2 \cap W_3)}$$

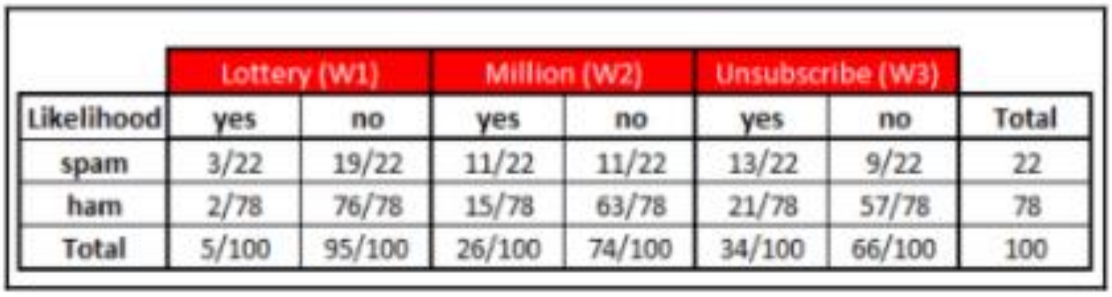
- When both events are independent we can write *P(A ∩ B) = P(A) * P(B)*. In fact, this equivalence is much easier to compute with less memory requirement:

$$P(Spam \mid W_1 \cap \neg W_2 \cap W_3) = \frac{P(W_1 \mid Spam) * P(\neg W_2 \mid Spam) * P(W_3 \mid Spam) * P(Spam)}{P(W_1) * P(\neg W_2) * P(W_3)}$$

- In a similar way, we will calculate the probability for ham messages as well, as follows:

$$P(Ham \mid W_1 \cap \neg W_2 \cap W_3) = \frac{P(W_1 \mid Ham) * P(\neg W_2 \mid Ham) * P(W_3 \mid Ham) * P(Ham)}{P(W_1) * P(\neg W_2) * P(W_3)}$$

- By substituting the preceding likelihood table in the equations, due to the ratio of spam/ham we can just simply ignore the denominator terms in both the equations. Overall likelihood of spam is:



	Lottery (W1)		Million (W2)		Unsubscribe (W3)		
Likelihood	yes	no	yes	no	yes	no	Total
spam	3/22	19/22	11/22	11/22	13/22	9/22	22
ham	2/78	76/78	15/78	63/78	21/78	57/78	78
Total	5/100	95/100	26/100	74/100	34/100	66/100	100

$$P(Spam|W_1 \cap \neg W_2 \cap W_3) = \left(\frac{3}{22}\right) * \left(\frac{11}{22}\right) * \left(\frac{13}{22}\right) * \left(\frac{22}{100}\right) = 0.008864$$

$$P(Ham|W_1 \cap \neg W_2 \cap W_3) = \left(\frac{2}{78}\right) * \left(\frac{63}{78}\right) * \left(\frac{21}{78}\right) * \left(\frac{78}{100}\right) = 0.004349$$

- After calculating ratio, *0.008864/0.004349 = 2.03*, which means that this message is two times more likely to be spam than ham. But we can calculate the probabilities as follows

$$P(Spam) = 0.008864/(0.008864+0.004349) = 0.67$$

$$P(Ham) = 0.004349/(0.008864+0.004349) = 0.33$$

XIII Laplace estimator

- In the previous calculation, all the values are nonzeros, which makes calculations well. Whereas in practice some words never appear in past for specific category and suddenly appear at later stages, which makes entire calculations as zeros
- For example, in the previous equation *W3* did have a *0* value instead of *13*, and it will convert entire equations to *0* altogether

$$P(Spam\ |W_1 \cap \neg W_2 \cap W_3) = \left(\frac{3}{22}\right) * \left(\frac{11}{22}\right) * \left(\frac{0}{22}\right) * \left(\frac{22}{100}\right) = 0$$

- In order to avoid this situation, Laplace estimator essentially adds a small number to each of the counts in the frequency table, which ensures that each feature has a nonzero probability of occurring with each class. Usually Laplace estimator is set to *1*, which ensures that each class-feature combination is found in the data at least once

$$P(Spam|W_1 \cap \neg W_2 \cap W_3) = \left(\frac{4}{25}\right) * \left(\frac{12}{25}\right) * \left(\frac{1}{25}\right) * \left(\frac{22}{100}\right) = 0$$

XIV Naive Bayes SMS spam classification example

- Data set : SMS spam collection
- NLP techniques to preprocess prior to build the Naive Bayes model

>>> import csv

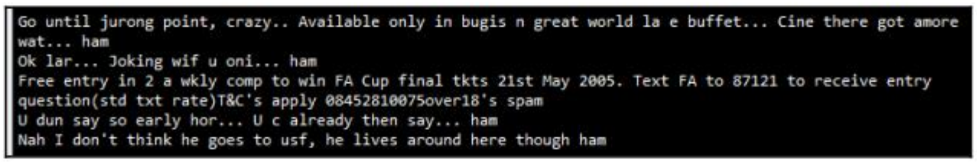
>>> smsdata = open('SMSSpamCollection.txt','r')

>>> csv_reader = csv.reader(smsdata,delimiter='\t')

```
>>> smsdata_data = []
>>> smsdata_labels = []

>>> for line in csv_reader:
...     smsdata_labels.append(line[0])
...     smsdata_data.append(line[1])

>>> smsdata.close()
```

The following code prints the top 5 lines:

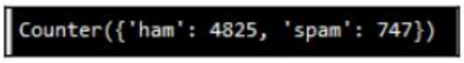
```
>>> for i in range(5):
...     print (smsdata_data[i],smsdata_labels[i])
```



```
Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore
wat... ham
Ok lar... Joking wif u oni... ham
Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry
question(std txt rate)T&C's apply 08452810075over18's spam
U dun say so early hor... U c already then say... ham
Nah I don't think he goes to usf, he lives around here though ham
```

After getting preceding output run following code:

```
>>> from collections import Counter
>>> c = Counter( smsdata_labels )
>>> print(c)
```



```
Counter({'ham': 4825, 'spam': 747})
```

Out of 5,572 observations, 4,825 are ham messages, which are about 86.5 percent and 747 spam messages are about remaining 13.4 percent.

Preprocessing stages are

- **Removal of punctuations**
- **Word tokenization**: Words are chunked from sentences based on white space for further processing
- **Converting words into lower case**: Converting to all lower case provides removal of duplicates, such as *Run* and *run*, where the first one comes at start of the sentence and the later one comes in the middle of the sentence, and so on
- **Stop word removal**: Stop words are the words that repeat so many times in literature and yet are not much differentiator in explanatory power of sentences. For example: *I*, *me*, *you*, *this*, *that*, and so on, which needs to be removed before further processing
- **of length at least three**: Here we have removed words with length less than three

- Stemming of words: Stemming process stems the words to its respective root words. Example of stemming is bringing down running to run or runs to run. By doing stemming we reduce duplicates and improve the accuracy of the model
- **Part-of-speech (POS) tagging**: This applies the speech tags to words, such as noun, verb, adjective, and so on. For example, POS tagging for *running* is verb, whereas for *run* is noun. In some situation *running* is noun and lemmatization will not bring down the word to root word *run*, instead it just keeps the *running* as it is.
- **Lemmatization of words**: Lemmatization is another different process to reduce the dimensionality. In lemmatization process, it brings down the word to root word rather than just truncating the words. For example, bring *ate* to its root word as *eat* when we pass the *ate* word into lemmatizer with the POS tag as verb.

The `nltk` package has been utilized for all the preprocessing steps, as it consists of all the necessary NLP functionality in one single roof:

```
>>> import nltk
>>> from nltk.corpus import stopwords
>>> from nltk.stem import WordNetLemmatizer
>>> import string
>>> import pandas as pd
>>> from nltk import pos_tag
>>> from nltk.stem import PorterStemmer
```

Function has been written (preprocessing) consists of all the steps for convenience. However, we will be explaining all the steps in each section:

```
>>> def preprocessing(text):
```

The following line of the code splits the word and checks each character if it is in standard punctuations, if so it will be replaced with blank and or else it just does not replace with blanks:

```
...      text2 = " ".join("".join([" " if ch in string.punctuation else ch
for ch in text]).split())
```

The following code tokenizes the sentences into words based on white spaces and put them together as a list for applying further steps:

```
...      tokens = [word for sent in nltk.sent_tokenize(text2) for word in
               nltk.word_tokenize(sent)]
```

Converting all the cases (upper, lower, and proper) into lowercase reduces duplicates in corpus:

```
...     tokens = [word.lower() for word in tokens]
```

As mentioned earlier, stop words are the words that do not carry much weight in understanding the sentence; they are used for connecting words, and so on. We have removed them with the following line of code:

```
...     stopwds = stopwords.words('english')
...     tokens = [token for token in tokens if token not in stopwds]
```

Keeping only the words with length greater than `3` in the following code for removing small words, which hardly consists of much of a meaning to carry:

```
...     tokens = [word for word in tokens if len(word)>=3]
```

Stemming is applied on the words using `PorterStemmer` function, which stems the extra suffixes from the words:

```
...     stemmer = PorterStemmer()
...     tokens = [stemmer.stem(word) for word in tokens]
```

POS tagging is a prerequisite for lemmatization, based on whether the word is noun or verb, and so on, it will reduce it to the root word:

```
...     tagged_corpus = pos_tag(tokens)
```

The `pos_tag` function returns the part of speed in four formats for noun and six formats for verb. `NN` (noun, common, singular), `NNP` (noun, proper, singular), `NNPS` (noun, proper, plural), `NNS` (noun, common, plural), `VB` (verb, base form), `VBD` (verb, past tense), `VBG` (verb, present participle), `VBN` (verb, past participle), `VBP` (verb, present tense, not third person singular), `VBZ` (verb, present tense, third person singular):

```
...     Noun_tags = ['NN','NNP','NNPS','NNS']
...     Verb_tags = ['VB','VBD','VBG','VBN','VBP','VBZ']
...     lemmatizer = WordNetLemmatizer()


...     def prat_lemmatize(token,tag):
...         if tag in Noun_tags:
...             return lemmatizer.lemmatize(token,'n')
...         elif tag in Verb_tags:
...             return lemmatizer.lemmatize(token,'v')
...         else:
...             return lemmatizer.lemmatize(token,'n')
```

After performing tokenization and applied all the various operations, we need to join it back to form stings and the following function performs the same:

```
...     pre_proc_text =  " ".join([prat_lemmatize(token,tag) for token,tag
in tagged_corpus])
...     return pre_proc_text
```

The following step applies the preprocessing function to the data and generates new corpus:

```
>>> smsdata_data_2 = []
>>> for i in smsdata_data:
...     smsdata_data_2.append(preprocessing(i))
```

```
>>> import numpy as np
>>> trainset_size = int(round(len(smsdata_data_2)*0.70))
>>> print ('The training set size for this classifier is ' +
str(trainset_size) + '\n')
>>> x_train = np.array([''.join(rec) for rec in
smsdata_data_2[0:trainset_size]])
>>> y_train = np.array([rec for rec in smsdata_labels[0:trainset_size]])
>>> x_test = np.array([''.join(rec) for rec in
smsdata_data_2[trainset_size+1:len( smsdata_data_2)]])
>>> y_test = np.array([rec for rec in smsdata_labels[trainset_size+1:len(
smsdata_labels)]])
```

The following code converts the words into a vectorizer format and applies **term frequency-inverse document frequency** (**TF-IDF**) weights, which is a way to increase weights to words with high frequency and at the same time penalize the general terms such as *the*, *him*, *at*, and so on. In the following code, we have restricted to most frequent 4,000 words in the vocabulary, none the less we can tune this parameter as well for checking where the better accuracies are obtained:

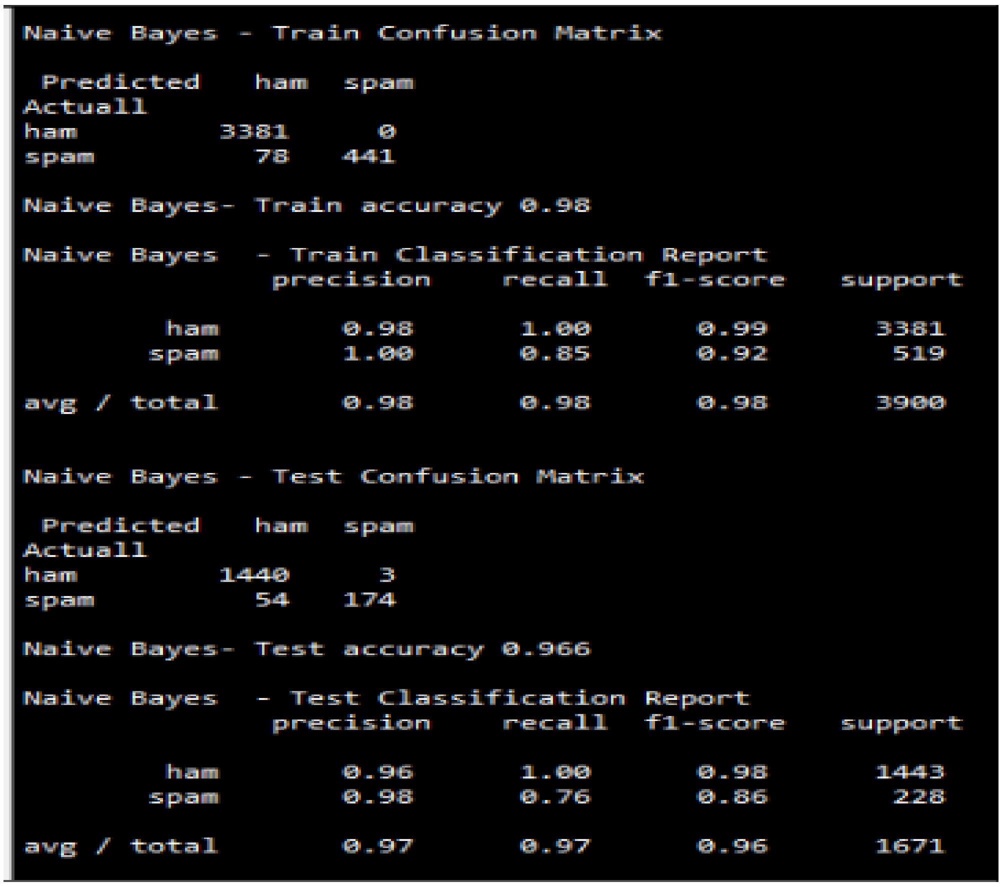
```
# building TFIDF vectorizer
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> vectorizer = TfidfVectorizer(min_df=2, ngram_range=(1, 2),
stop_words='english',
    max_features= 4000,strip_accents='unicode',  norm='l2')
```

The TF-IDF transformation has been shown as follows on both train and test data. The `todense` function is used to create the data to visualize the content:

```
>>> x_train_2 = vectorizer.fit_transform(x_train).todense()
>>> x_test_2 = vectorizer.transform(x_test).todense()
```

```
>>> from sklearn.naive_bayes import MultinomialNB
>>> clf = MultinomialNB().fit(x_train_2, y_train)

>>> ytrain_nb_predicted = clf.predict(x_train_2)
>>> ytest_nb_predicted = clf.predict(x_test_2)

>>> from sklearn.metrics import classification_report,accuracy_score
```

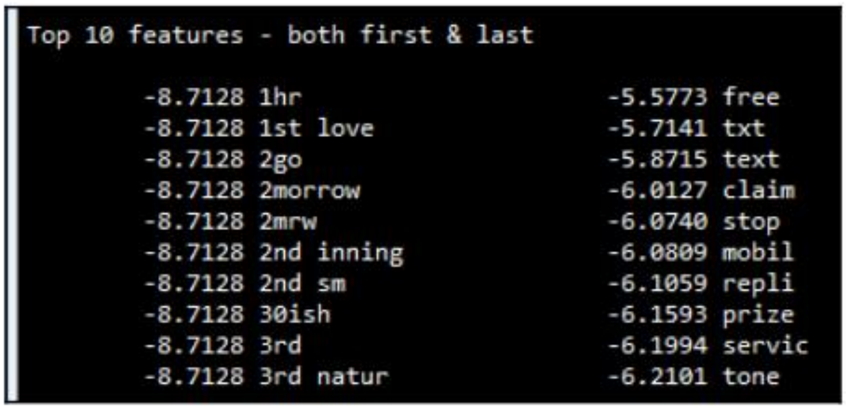
```
>>> print ("\nNaive Bayes - Train Confusion
Matrix\n\n",pd.crosstab(y_train, ytrain_nb_predicted,rownames =
["Actuall"],colnames = ["Predicted"]))
>>> print ("\nNaive Bayes- Train accuracy",round(accuracy_score(y_train,
ytrain_nb_predicted),3))
>>> print ("\nNaive Bayes  - Train Classification
Report\n",classification_report(y_train, ytrain_nb_predicted))

>>> print ("\nNaive Bayes - Test Confusion Matrix\n\n",pd.crosstab(y_test,
ytest_nb_predicted,rownames = ["Actuall"],colnames = ["Predicted"]))
>>> print ("\nNaive Bayes- Test accuracy",round(accuracy_score(y_test,
ytest_nb_predicted),3))
>>> print ("\nNaive Bayes  - Test Classification
Report\n",classification_report( y_test, ytest_nb_predicted))
```



```
Naive Bayes  - Train Confusion Matrix

 Predicted    ham   spam
Actuall
ham         3381      0
spam          78    441

Naive Bayes- Train accuracy 0.98

Naive Bayes   - Train Classification Report
              precision    recall  f1-score   support

        ham       0.98      1.00      0.99      3381
       spam       1.00      0.85      0.92       519

avg / total       0.98      0.98      0.98      3900


Naive Bayes  - Test Confusion Matrix

 Predicted    ham   spam
Actuall
ham         1440      3
spam          54    174

Naive Bayes- Test accuracy 0.966

Naive Bayes   - Test Classification Report
              precision    recall  f1-score   support

        ham       0.96      1.00      0.98      1443
       spam       0.98      0.76      0.86       228

avg / total       0.97      0.97      0.96      1671
```

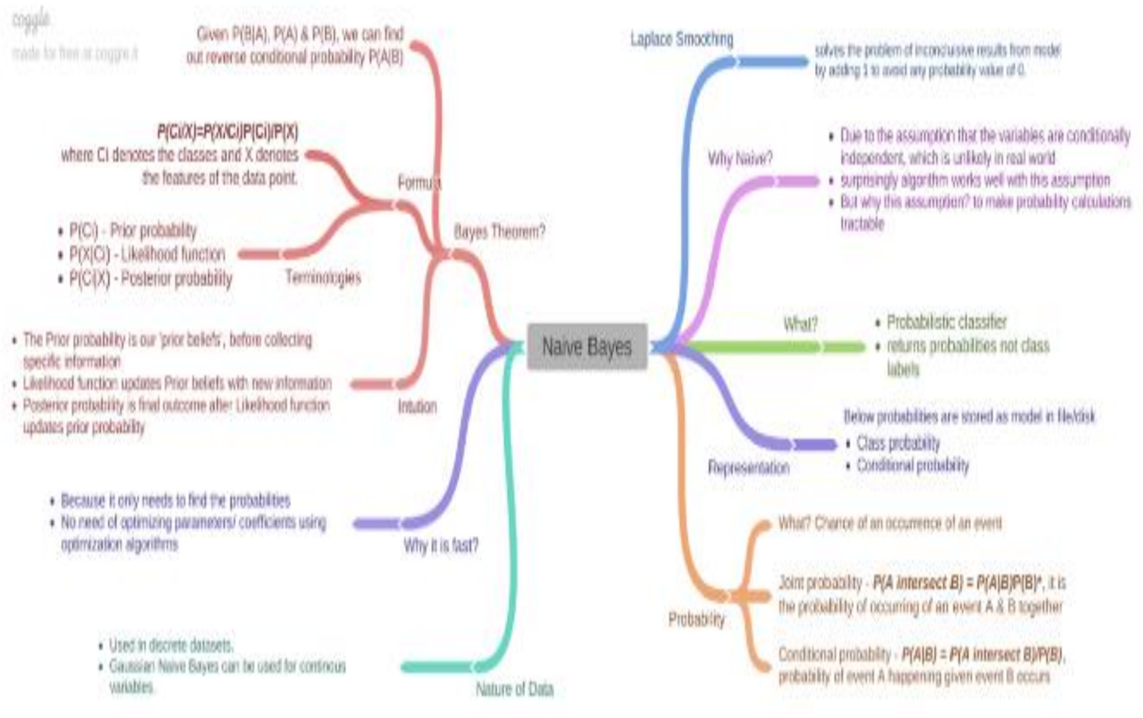
However, if we would like to check what are the top 10 features based on their coefficients from Naive Bayes, the following code will be handy for this:

```
# printing top features
>>> feature_names = vectorizer.get_feature_names()
>>> coefs = clf.coef_
>>> intercept = clf.intercept_
>>> coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))

>>> print ("\n\nTop 10 features - both first & last\n")
>>> n=10
>>> top_n_coefs = zip(coefs_with_fns[:n], coefs_with_fns[:-(n + 1):-1])
>>> for (coef_1, fn_1), (coef_2, fn_2) in top_n_coefs:
...     print('\t%.4f\t%-15s\t\t%.4f\t%-15s' % (coef_1, fn_1, coef_2,
fn_2))
```



```
Top 10 features - both first & last

        -8.7128 1hr                        -5.5773 free
        -8.7128 1st love                   -5.7141 txt
        -8.7128 2go                        -5.8715 text
        -8.7128 2morrow                    -6.0127 claim
        -8.7128 2mrw                       -6.0740 stop
        -8.7128 2nd inning                 -6.0809 mobil
        -8.7128 2nd sm                     -6.1059 repli
        -8.7128 30ish                      -6.1593 prize
        -8.7128 3rd                        -6.1994 servic
        -8.7128 3rd natur                  -6.2101 tone
```

Naive Bayes - Mind Map


coggle
made for free at coggle.it
Naive Bayes
Bayes Theorem?
Given P(B|A), P(A) & P(B), we can find out reverse conditional probability P(A|B)
Formula
P(Ci/X)=P(X/Ci)P(Ci)/P(X)
where Ci denotes the classes and X denotes the features of the data point.
Terminologies
• P(Ci) - Prior probability
• P(X|Ci) - Likelihood function
• P(Ci|X) - Posterior probability
Intution
• The Prior probability is our 'prior beliefs', before collecting specific information
• Likelihood function updates Prior beliefs with new information
• Posterior probability is final outcome after Likelihood function updates prior probability
Why it is fast?
• Because it only needs to find the probabilities
• No need of optimizing parameters/ coefficients using optimization algorithms
Nature of Data
• Used in discrete datasets.
• Gaussian Naive Bayes can be used for continous variables.
Laplace Smoothing
solves the problem of inconclusive results from model by adding 1 to avoid any probability value of 0.
Why Naive?
• Due to the assumption that the variables are conditionally independent, which is unlikely in real world
• surprisingly algorithm works well with this assumption
• But why this assumption? to make probability calculations tractable
What?
• Probabilistic classifier
• returns probabilities not class labels
Representation
Below probabilities are stored as model in file/disk
• Class probability
• Conditional probability
Probability
What? Chance of an occurrence of an event
Joint probability - P(A intersect B) = P(A|B)P(B)*, it is the probability of occuring of an event A & B together
Conditional probability - P(A|B) = P(A intersect B)/P(B), probability of event A happening given event B occurs