Chapter 4
Naive Bayes Algorithm

Objectives

- What is working principle of Naïve Bayes algorithm?
- Mathematics behind it?
- Advantage, disadvantage and applications of Naïve Bayes algorithm
- Implementation of spam mail classifier
- Encoding of categorical data
- Evaluation metrics of classification models
- How to export a ML model as a file?

Probability

What is probability?

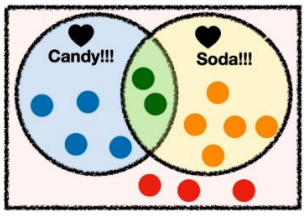
Probability is the extent to which an event is likely to occur, measured by the ratio of the favorable cases to the whole number of cases possible.

$$\text{Probablity of event E occur} = \frac{\text{Number of events}}{\text{Total number of samples}}$$

$$P(E) = \frac{N(E)}{N(S)}$$

Example: *There are 14 person in your family,*
7 of them love soda,
6 of them love candy,
2 of them love both,
3 of them love neither soda nor candy.

	Loves Candy	Doesn't Love Candy
Loves Soda	2	5
Doesn't Love Soda	4	3



What is probability of meeting someone who loves soda?

$$P(\text{loves soda}) = \frac{N(\text{loves soda})}{\text{Total number of the family}} = \frac{7}{14} = 0.5$$

What is conditional probability?

Conditional probability is the probability of an event (A), given that another (B) has already occur.

$$\text{Probablity of event A given that event B occur} = \frac{\text{Probability of both event A and B occur}}{\text{Probability of event B occur}}$$

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

$$P(\textit{what do we want to know} \mid \textit{what we know}) = \frac{P(\textit{what do we want to know})}{P(\textit{what we know})}$$

Example: *What is probability of meeting someone who loves both candy and soda given that he/she loves candy?*

$$P(\text{loves candy} \cap \text{loves soda}) = \frac{N(\text{loves candy and soda})}{\text{Total number of the family}} = \frac{2}{14} = 0.142857$$

$$P(\text{loves candy and soda} \mid \text{loves soda}) = \frac{P(\text{loves candy} \cap \text{loves soda})}{P(\text{loves soda})} = \frac{\frac{2}{14}}{\frac{7}{14}} = \frac{2}{7} = 0.2857$$

Can we solve the above problem without knowing P(loves candy ∩ loves soda) but you already know the probability of meeting someone who loves both candy and soda given that he/she loves soda?

Bayes' Theorem [*stated by Thomas Baye*]

$$P(A|B) = \frac{P(A \cap B)}{P(B)}, \quad P(B|A) = \frac{P(B \cap A)}{P(A)}$$

Since $P(A \cap B) = P(B \cap A)$, $P(A \mid B) * P(B) = P(B \mid A) * P(A)$, $P(A|B) = \frac{P(B|A)*P(A)}{P(B)}$

"

Probability of an event occurring based on prior knowledge of conditions that might be related to the event.

"

Formula: $P(A|B) = \frac{P(B|A)*P(A)}{P(B)}$

Where:

P(A|B) | **Posterior Probability** | is conditional probability of event A occurring given event B.

> *Definition of posterior: a Latin word "posterus" meaning coming after.*

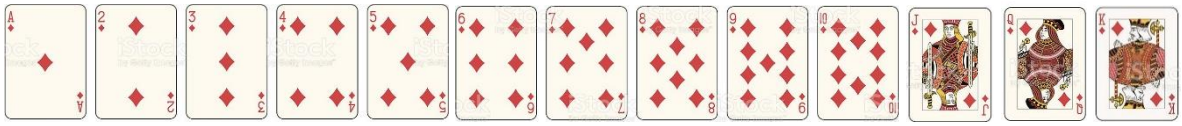
P(B|A) | **likelihood** | is conditional probability of event B occurring given the event A.

> *Definition of likelihood: likely (having the appearance of truth or fact) +*
> *hood (state or condition)*
> *likelihood meaning having appearance of truth in a certain condition.*

P(A) | **Prior Probability** | is probability of event A occurring.

> *Definition of prior: earlier; preceding, as in order of time.*

Example: *Pick a random card, you already know it is a diamond. Now what is the probability of that card being a queen?*



$$P(\text{diamond}) = \frac{N(\text{diamond})}{\text{Total number of cards}} = \frac{13}{52} = \frac{1}{4}$$

$$P(\text{queen}) = \frac{N(\text{queen})}{\text{Total number of cards}} = \frac{4}{52} = \frac{1}{4}$$

$$P(\text{diamond} \cap \text{queen}) = \frac{N(\text{queen diamond})}{\text{Total number of cards}} = \frac{1}{52} = \frac{1}{52}$$

$$P(\text{diamond} \mid \text{queen}) = \frac{P(\text{diamond} \cap \text{queen})}{P(\text{diamond})} = \frac{\frac{1}{52}}{\frac{1}{4}} = \frac{1}{13}$$

$$P(\text{queen} \mid \text{diamond}) = \frac{P(\text{diamond} \mid \text{queen}) * P(\text{queen})}{P(\text{diamond})} = \frac{\frac{1}{13} * \frac{1}{4}}{\frac{1}{4}} = \frac{1}{13}$$

Naive Bayes Classifier

A classifier under supervised ML group based on probabilistic logic that is Bayes theorem.

So, why it is called naïve?

Because we are making a naïve assumption that features or attributes are independent of each other. The effect of an attribute value on a given class is independent of the values of the other attributes. This assumption is made to reduce computational costs and hence is considered Naïve.

> *Definition of naive: innocent, natural, simple , unsophisticated*

Types of Naïve Bayes classifiers

1. ***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".

```
# Bernoulli Naïve Bayes Classifier
from sklearn.naive_bayes import BernoulliNB
model = BernoulliNB()
```

2. ***Multinomial Naive Bayes :*** It 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.

```
# Multinomial Naïve Bayes Classifier
from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB()
```

3. ***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.

```
# Bernoulli Naïve Bayes Classifier
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
```

Advantages of Naïve Bayes Classifier

1. Simple to Implement because The conditional probabilities are easy to evaluate.
2. Very fast – no iterations since the probabilities can be directly computed. So this technique is useful where speed of training is important.
3. If the conditional Independence assumption holds true, it could give great results.
4. It requires small amount of training data to estimate the test data. So, the training period is less.
5. It performs well in Multi-class predictions as compared to the other Algorithms.

Disadvantages of Naïve Bayes Classifier

1. Conditional Independence Assumption does not always hold. In most situations, the feature show some form of dependency.
2. **Zero probability problem** : When we encounter words in the test data for a particular class that are not present in the training data, we might end up with zero class probabilities.

Applications of Naïve Bayes Classifier

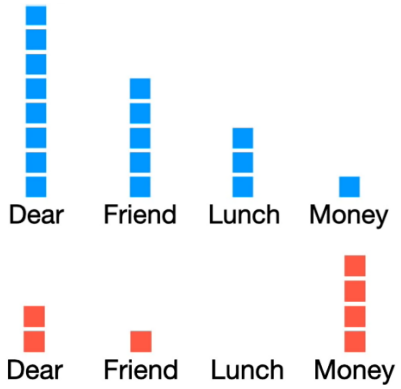
1. Email spam detection
2. Handwritten character recognition
3. Weather prediction
4. Face detection
5. News article categorization
6. Sentiment analysis
7. Credit scoring

Example (Spam email detection)

Imagine you received normal messages from friends and family, and you also received spam (unwanted messages that are usually scams or unsolicited advertisements), and you wanted to filter out the spam messages. For example, **8** of **12** messages are normal messages while the rest are spams and you just receive a mail with content "Dear friend", then classify this email whether it is normal or spam?

Step 1: Identify words frequently repeated in normal messages

Step 2: Make a histogram



Step 3: Calculate probability of individual words

Being normal message		Being spam message	
P(Dear | Normal)	= 8/17	P(Dear | Spam)	= 2/7
P(Friend | Normal)	= 5/17	P(Friend | Spam)	= 1/7
P(Lunch | Normal)	= 3/17	P(Lunch | Spam)	= 0/7
P(Money | Normal)	= 1/17	P(Money | Spam)	= 4/7

Step 4: Calculate probability of normal and spam email

$P(normal) = \frac{8}{8+4} = \frac{8}{12} = \frac{2}{3}$ $P(spam) = \frac{4}{4+8} = \frac{4}{12} = \frac{1}{3}$

Step 5: Calculate probability of normal or spam email for Dear friend

P(normal | 'Dear Friend') = P(normal) * P('Dear' | normal) * P('Friend' | normal)

= 2/3 * 8/17 * 5/17 = 80/867 = 0.0922722

P(spam | 'Dear Friend') = P(spam) * P('Dear' | spam) * P('Friend' | spam)

= 1/3 * 2/7 * 1/7 = 2/147 = 0.0136

Step 6: If P(normal | E) < P(spam | E), the email is spam else it is normal mail

0.09 > 0.01, Therefore 'Dear Friend' content message is normal email.

Python Implementation of Spam Email Classifier

Pre-requisite: We need **Jupyter Notebook** for a working environment.

Jupyter Notebook is an open source web application that you can use to create and share live code.

How to install Jupyter Notebook?

- Download Anaconda
  - ✓ Visit the link http://anaconda.org
  - ✓ Click Download Anaconda
  - ✓ Select Window / Mac
  - ✓ Download python 3.6 version
- Open downloaded file
- Click continue
- Click install for me only
- You get notification 'The installation was completed successfully'
- Click close

How to write and run a code on Jupyter Notebook?

PS. Make sure your internet connection is working.

- Open anaconda prompt
- Select desired folder from the given list of directories
- Click new
- Click python 3
- Write your code inside the cells

Example: Program to print hello world

```
In [ ]:  1  print("Hello World")
```

- Rename file name
- Click file
- Click save as
- Click file path (This is optional)
- Click save
- Click 'Shift + Enter' to run your code
- Finally you will get an output 'Hello World' on next line to your code

```
In [1]:  1  print("Hello World")

Out[1]:  Hello World
```

- Else you made a mistake, so recheck it once again.

Pandas

Pandas is an open-source library that provides high-performance data manipulation in Python. It is used for data analysis in Python and developed by **Wes McKinney** in 2008.

> *Definition of pandas: The name derived from the word panel data.*

Data analysis requires lots of processing, such as **reading, restructuring, cleaning** or **merging**, etc. There are different tools are available for fast data processing, such as **Numpy**, **Scipy**, **Cython**, and **Panda**. But we prefer Pandas because working with Pandas is fast, simple and more expressive than other tools.

To read and process our dataset we need to install and import pandas library, Python pip is the package manager for Python packages. We can use pip to install packages or libraries that do not come with Python.

> *Definition of pip: Pip Installs Packages*

In [1]:
```
# install pandas /make sure your internet connection is working fine/
pip install pandas
```

```
Requirement already satisfied:
```

Let us import pandas library as pd because whenever we want to use pandas library we will easily call pd instead of pandas.

In [2]:
```
# import pandas library to our working environment
import pandas as pd
# check pandas version
pd.__version__
```

Out[2]: `'1.3.4'`

Now let us import our dataset that we are going to use for training and testing our classifier model.

download the dataset from here:

https://github.com/bemnetdev/Mini-Machine-Learning/blob/main/Naive%20Bayes/spam.csv

In [3]:
```
# read our dataset then save as a dataframe
df = pd.read_csv("file path/spam.csv")
```

NB: df is a variable that we store our spam.csv dataset as a pandas dataframe

Now let us look 3 sample data from our dataframe

In [4]:
```
# view three sample data
df.sample(3)
```

Out[4]:

	Category	Message
13	ham	I've been searching for the right words to tha...
2145	spam	FreeMsg: Hey - I'm Buffy. 25 and love to satis...
2043	ham	Me not waking up until 4 in the afternoon, sup

We can explore general setting of our dataset using the following codes

In [5]:
```
# overview of each column
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Category  5572 non-null   object
 1   Message   5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

In [6]:
```
# dimension of dataset
df.shape()
```

Out[6]:
```
(5572, 2)
```

In [7]:
```
# number of non-empty records in a column
df.count()
```

Out[7]:
```
Category    5572
Message     5572

dtype: int64
```

In [8]:
```
# statistical summary of our dataset
df.describe()
```

Out[8]:

	Category	Message
count	5572	5572
unique	2	5157
top	ham	Sorry, I'll call later
freq	4825	30

In [9]:
```
# distribution of categorical data
df["Category"].value_counts()
```

Out[9]:
```
ham     4825
Spam     747
Name: Category, dtype: int64
```

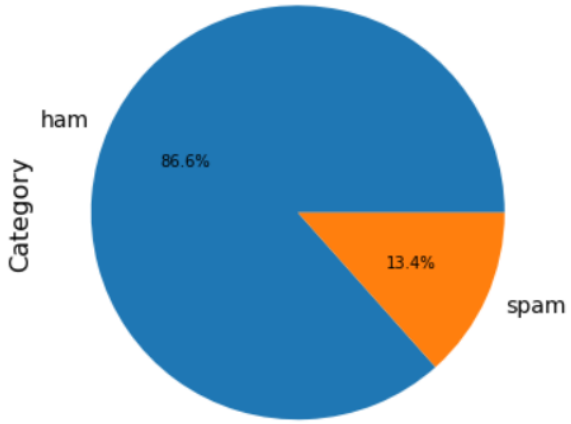
In [10]:
```
# proportion of categorical data
df["Category"].value_counts(normalize=True)
```

Out[10]:
```
ham      0.865937
Spam     0.134063
Name: Category, dtype: float64
```

In [11]:
```
# install matplotlib for visualization
pip install matplotlib
```

```
Requirement already satisfied:
```

In [12]:
```
# visualize proportion of categorical data
%matplotlib inline
import matplotlib.pyplot as plt
prop = df.Category.value_counts(normalize=True)
prop.plot(kind="pie", autopct='%1.1f%%', figsize=(6,6))
plt.show()
```



We encode categorical data numerically because math is generally done using numbers. A big part of encoding is converting text to numbers. Just like that, our algorithms cannot run and process data if that data is not numerical.

Encoding categorical data

Encoding is the conversion of categorical features to numeric values as machine Learning models cannot handle the text data directly.

Label / Ordinal Encoding

Label Encoding refers to converting the labels into a numeric form. It is recommended for ordinal categorical data.

Example: Bachelors, Masters, Phd

```
# create a dataframe using a dictionary
data = { 'Degree': ['Masters',
                    'Bachelors',
                    'Bachelors',
                    'Masters',
                    'Phd']}
# encoding categorical attribute using ordinal encoding
df = pd.DataFrame(data)
df
```

	Degree
0	Masters
1	Bachelors
2	Bachelors
3	Masters
4	Phd

```
# encoding categorical attribute using ordinal encoding
import category_encoders as ce

# create object of Ordinalencoding (oe)
encoder= ce.OrdinalEncoder(cols = ['Degree'], mapping = [{'col': 'Degree',
'mapping':{'None': 0,
           'Bachelors': 1,
           'Masters': 2,
           'Phd': 3}}])

# fit and transform the data
df_oe = encoder.fit_transform(df)
df_oe
```

	Degree
0	2
1	1
2	1
3	2
4	3

OneHot Encoding

One hot encoding is a process of converting categorical data variables so they can be provided to machine learning algorithms to improve predictions. It is recommended for Nominal categorical data

Example: green, yellow, red, pink, black

Based on the above example create a dataframe like down below using animal column

	Color
0	Green
1	Yellow
2	Red
3	Pink
4	Black

```
# create an object for one-hot encoding (ohe)
cat_encoder = ce.OneHotEncoder(cols = 'Color', use_cat_names=True)

# fit and transform Data
df_ohe = cat_encoder.fit_transform(df)
df_ohe
```

	Color_Green	Color_Yellow	Color_Red	Color_Pink	Color_Black
0	0.0	1.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0

Here we have nominal categorical data so we use One Hot Encoding

In [13]:
```
# install category_encoders library
pip install category_encoders
```

```
Requirement already satisfied:
```

In [14]:
```
# initiate encoder object for one hot encoding
import category_encoders as ce
encoder = ce.OneHotEncoder(cols= 'Category', use_cat_names=True)
# fit and transform Data
df = encoder.fit_transform(df)
# view transformed three sample data
df.sample(3)
```

Out[14]:

	Category_ham	Category_spam	Message
13	1	0	I've been searching for the right...
2145	0	1	FreeMsg: Hey - I'm Buffy. 25 and...
2043	1	0	Me not waking up until 4 in the...

Dummy variables

Dummy variables are variables containing values 0 or 1 representing the presence or absence of the categorical value such as ham = 1, spam = 0.

Dummy variables trap

The dummy variable trap is a scenario in which two or more variables are highly correlated. In simple terms, one variable can be predicted from the others. Intuitively, there is a duplicated category: if we drop the ham category it is inherently defined in the spam category (spam value indicated ham, and vice versa).

The solution to the dummy variable trap is to drop one of the categorical variables – if there are m number of categories. Use m - 1 in the model

In [15]:
```
# drop Category_ham column
df.drop(['Category_ham'], axis=1)
```

Out[15]:

	Category_spam	Message
13	0	I've been searching for the right...
2145	1	FreeMsg: Hey - I'm Buffy. 25 and...
2043	0	Me not waking up until 4 in the...

Train-Test Split

Now to train and evaluate our classifier we need to randomly split our dataframe into training and test set. Training set will be 80% of the whole dataset and test set will be the rest 20%. The model is trained using the training dataset and then evaluated using the test dataset. This method is called Holdout method. To split our dataset we need to install a python library called “scikit-learn”.

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modelling including classification, regression, clustering…etc.

Here our feature (input) is the message and label (output) is Category_spam, the following figure illustrates how our dataset will be split.

Category spam	Message	
0	Message 1	80% of the dataset (Training set)
1	Message 2	
0	Message 3	
. . .	. . .	
0	Message 5571	20% of the dataset (Test set)
1	Message 5572	
Label (Output) X	Feature (Input) y	X_train y_train X_test y_test

In [16]:
```
# install sklearn library
pip install sklearn
```

```
Requirement already satisfied:
```

In [17]:
```
# Split the dataframe into X_train, X_test, y_train and y_test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df.Message,
df.Category_spam, test_size = 0.2, random_state = 10)
print("Size of X_train = ", X_train.shape[0])
print("Size of X_test = ", X_test.shape[0])
print("Size of y_train = ", y_train.shape[0])
print("Size of y_test = ", y_test.shape[0])
```

Out[17]:
```
Size of X_train =  4457
Size of X_test =  1115
Size of y_train =  4457
Size of y_test =  1115
```

But, in the above train test split we have considered purely random sampling methods. This is generally fine if your dataset is large enough (especially relative to the number of attributes) and balanced with respect to class labels, but if it is not, you run the risk of introducing a significant sampling bias.

What is sampling bias?

Sampling bias is a bias that occurs when some members of the intended population have a higher or lower probability of being selected than others. This results in a biased sample of a population in which not all individuals were equally likely to have been selected. Although a bias in the sample may not always invalidate the data gathered.

When a survey company decides to call 1,000 people to ask them a few questions, they don't just pick 1,000 people randomly in a phone book. They try to ensure that these 1,000 people are representative of the whole population. For example, the US population is 51.3% females and 48.7% males, so a well-conducted survey in the US would try to maintain this ratio in the sample: 513 female and 487 male. This is called **stratified sampling**:

In [18]:
```
# stratified split
X_train, X_test, y_train, y_test = train_test_split
(df.Message, df.Category_spam, test_size = 0.2, random_state = 10,
stratify = df.Category_spam)
y_train.value_counts(normalize=True)
```

Out[18]:
```
0    0.865829
1    0.134171
Name: Category_spam, dtype: float64
```

In [19]:
```
# ham/spam proportion of full dataset
df["Category_spam"].value_counts(normalize=True)
```

Out[19]:
```
0    0.865937
1    0.134063
Name: Category_spam, dtype: float64
```

Previously, we encode our label (output) data to integer 0 and 1, but how about our Message column data, these texts also need to encode (convert to integer) using Count Vectorizer.

CountVectorizer

CountVectorizer is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text. This is helpful when we have multiple such texts, and we wish to convert each word in each text into vectors (for using in further text analysis). Let us consider a few sample texts from a document (each as a list element):

document = ["Dear friend are you free now",
"Free gift you can win now",
"Hello friend you have class"]

	are	can	class	dear	free	friend	gift	have	hello	now	win	you
document[0]	1	0	0	1	1	1	0	0	0	1	0	1
document[1]	0	1	0	0	1	0	1	0	0	1	1	1
document[2]	0	0	1	0	0	1	0	1	1	0	0	1

Key Observations:

1. There are 12 unique words in the document, represented as columns of the table.
2. There are 3 text samples in the document, each represented as rows of the table.
3. Every cell contains a number, that represents the count of the word in that particular text.
4. All words have been converted to lowercase.
5. The words in columns have been arranged alphabetically.

Inside CountVectorizer, these words are not stored as strings. Rather, they are given a particular index value. In this case, 'are' would have index 0, 'can' would have index 1, 'class' would have index 2 and so on. The actual representation has been shown in the table below – this is called **sparse matrix**.

0	1	2	3	4	5	6	7	8	9	10	11
1	0	0	1	1	1	0	0	0	1	0	1
0	1	0	0	1	0	1	0	0	1	1	1
0	0	1	0	0	1	0	1	1	0	0	1

Python implementation of CountVectorizer

```
# create a list of text elements
document = [ "Dear friend are you free now",
             "Free gift you can win now",
             "Hello friend you have class" ]

# import CountVectorizer class from sklearn python library
from sklearn.feature_extraction import CountVectorizer

# create an object for CountVectorizer
vectorizer = CountVectorizer()

# fit (ready parameters for transformation)
vectorizer.fit(document)

# print identified unique words along with their indices
print(vectorizer.vocabulary_)

# transform (perform calculation and transform the input data)
Vector = vectorizer.transform(document)

# print sparse matrix of encoded text
print("Encoded Document is:\n", vector.toarray())
```

In [20]:
```
# import CountVectorizer class from sklearn python library
from sklearn.feature_extraction import CountVectorizer
# create an object for CountVectorizer
vectorizer = CountVectorizer()
# transform our data
X_train_count = v.fit_transform(X_train.values)
# convert transformed data to an array
X_train_count.toarray()
# print three sample array elements from transformed data
X_train_count[:3]
```

Out[20]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

In [21]:
```
# import Multinomial Naïve Bayes Classifier ML algorithm
from sklearn.naive_bayes import MultinomialNB
# build our model using Multinomial Naïve Bayes Classifier
model = MultinomialNB()
# train our model using train set data
model.fit(X_train_count, y_train)
```

Out[21]:
```
MultinomialNB()
```

In [22]:
```
# transform our test set data
X_test_count = vectorizer.transform(X_test)
# predict using our test set data
model.predict(X_test_count)
```

Out[22]: array([0., 0., 0., ..., 0., 0., 0.])

Evaluation Metrics of Classification Algorithms

1. **Accuracy**: we can measure accuracy of our model using accuracy metric

In [23]:
```
# training accuracy
model.score(X_train_count, y_train)
```

Out[23]: 0.9932690150325331

Train-Test Split problem

The Data we have selected for training and testing is randomly. Due to random selection, some kind of data that is present in Test data may not present in Training data. So what will happen?

whenever we perform Train-Test split, we use a random_state variable. And then we define the value of random_state. And based on this random_state variable, data is selected randomly. So What's the problem here?.

Suppose, first we have chosen random_state =5. And Split the data in 70% and 30%. So the accuracy we got is around 80%. But, again when we choose random_state=10. And Train-Test split at 70% and 30%. So the Training data and Testing Data gets shuffled. That means the data which we had in the previous round, is now shuffled. Now we have different data in Training Data and in Testing Data.

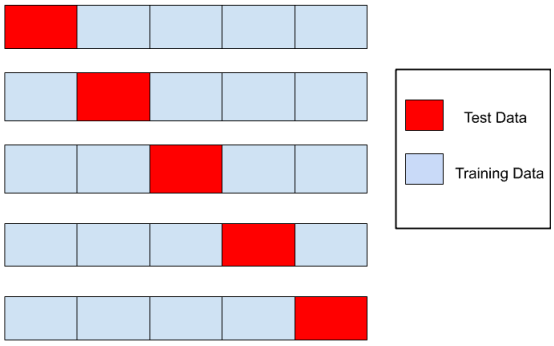
And now we got an accuracy of 85%. So, the problem is our accuracy fluctuates with different random_state. And we can't say that what accuracy our model has. Therefore to prevent this problem, Cross-Validation is used.

Why do we use cross validation?

We solve K-fold Cross-validation to evaluate our model accuracy. And to solve the Train-Test Split problem. Cross-Validation helps you to find a perfect model or algorithm for your problem. So yes, Cross-Validation is needed. If you want to choose the best algorithm who has high accuracy for your problem.

How Cross-Validation works?

Cross-Validation split the dataset into different segments. And use each segment for training as well as testing one by one. You can understand with the help of this image-



Types of Cross-Validation

Cross-Validation has following types-

1. Leave One Out Cross-Validation
2. K-Fold Cross-Validation.

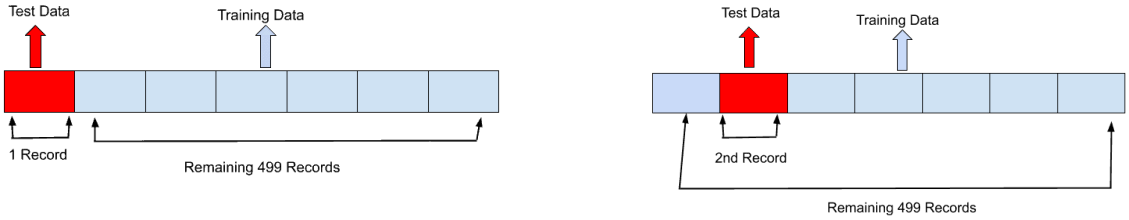
1. LOOCV (Leave One Out Cross Validation)

As its name suggests, "Leave One Out", therefore it leave only one record for testing. In this method, we perform training on the whole data-set but leaves only one data-point of the available data-set and then iterates for each data-point.

Example

Suppose, we have 500 records in our dataset. So what Leave One Out Cross-validation does?. It basically takes one record for the Test dataset and the remaining record for the Training dataset.

Let's understand with the help of an example-



Advantages of LOOCV?

An advantage of using this method is that we make use of all data points and hence it is low bias.

Disadvantages of LOOCV?

The major drawback of this method is that it leads to higher variation in the testing model as we are testing against one data point. If the data point is an outlier it can lead to higher variation. Another drawback is it takes a lot of execution time as it iterates over 'the number of data points' times.

2. K fold cross validation

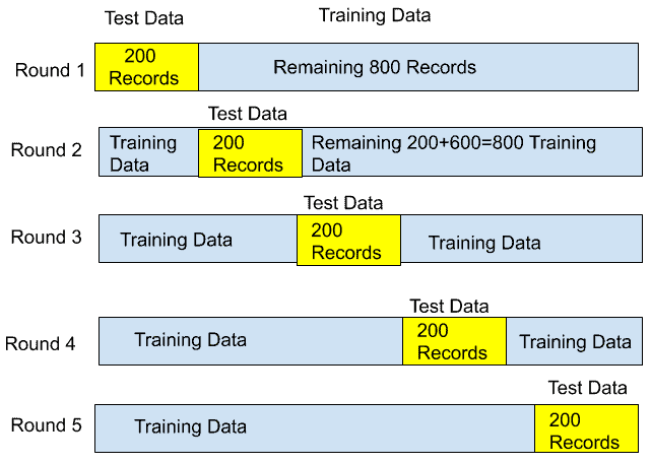
Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into.

How it works?

In this method, we split the data-set into k number of subsets(known as folds) then we perform training on the all the subsets but leave one(k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time.

Example

Suppose we have 1000 records in our dataset. And we select the value of K as 5. So K value means, the number of rounds we perform Training and Testing. And here, our k value is 5, that means we have to perform 5 round of Training and Testing.



We can take all Accuracies, and find out the Mean. And that Mean of all 5 accuracies is the actual accuracy of your model.

By doing so, your accuracy will not fluctuate as in Train-Test Split. Moreover, you will get your model minimum accuracy and maximum accuracy.

Does Cross-Validation improve accuracy?

K Fold Cross Validation is all about estimating the accuracy, not improving the accuracy. If you increase the K value, it will increase the accuracy of the measurement of your accuracy. But it will not improve the original accuracy.

It helps you to choose the accurate machine learning algorithm for your problem. You can check the accuracy of each Machine learning algorithm by estimating its performance, and then based on its accuracy, you can choose the best one for your problem.

How to Choose the value of K?

- The value of k should be chosen carefully. If you choose poor k value, it will result in high variance or high bias.
- The value of K depends upon the size of the data. Along with that, How much your system is capable to afford the computational cost. The high K value, the more rounds or folds you need to perform.
- So, before selecting the K value, look at your data size and your system computation power.

There are some common strategies for choosing the k value-

- **K = 10** → This is the value found by after various experiments. k=10 will result in a model with low bias and moderate variance. So if you are struggling to choose the value of k for your dataset, you can choose k=10. The value of k as 10 is very common in the field of machine learning.
- **K = n** → The value of k is n, where n is the size of the dataset. That means using each record in a dataset to test the model. That is nothing but Leave One Out Approach.
- **5 < K > 10** → There is no formal rule but the value of k should be 5 or 10.

NB: Time complexity of K fold cross-validation is **O(Kn)**. Where, n is the sample size and K is constant.

Advantages of K fold Cross-validation

1. No randomness for Train Test Split.
2. It has less bias.
3. It is not as computationally expensive as Leave One Out.

Disadvantages of K fold Cross-validation

It doesn't check about proper proportions in the test dataset and Training Dataset. Suppose in Round 1, we have 200 Test Dataset. But what if all these test datasets are of the same type? That means, suppose we are working on classification problem, and we have dataset only in binary form, like 0 and 1.

And we have only 0's in the test dataset. So, this is an Imbalanced dataset. And that can be a problem. In that case, you will not get accurate results.

Stratified K-fold cross validation

In order to solve the above problem, Stratified Cross-Validation is used. This method is similar to k-fold cross validation but with a slight difference. Here it ensured that while splitting the dataset into k folds, each fold should contain the same proportion of instances with a given categorical value.

Measuring Accuracy Using Cross-Validation

In [24]:
```
# import cross validation score class
from sklearn.model_selection import cross_val_score
cross = cross_val_score(model, X_train_count, y_train,
                        cv = 10, scoring = "accuracy")
print("Accuracy of 10 folds\n", cross)
print("Average Accuracy =", cross.mean())
```

Out[24]:
```
Accuracy of 10 folds
[0.97309417 0.97309417 0.98206278 0.98654709 0.98430493
 0.95515695 0.98206278 0.97752809 0.97752809 0.98426966]
Average Accuracy = 0.9775648712651787
```

The Problem Measuring with Accuracy

Suppose, you have 1000 imbalanced training data i.e. 900 of them are spam and the rest 100 belong to ham. Your classifier model classify all 1000 message as a spam. Therefore the accuracy will be calculated as follows:

$$\boldsymbol{Accuracy} = \frac{\mathbf{900}}{\mathbf{1000}} = \mathbf{90\%}$$

However your model is predicting ham messages incorrectly and whenever new message is coming it predict as a spam that is not what we want because the user will miss important messages. So to solve this problem we will use additional performance metrics using a confusion matrix.

Confusion Matrix

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of data. It can only be determined if the true values for test data are known. The matrix itself can be easily understood, but the related terminologies may be confusing. Since it shows the errors in the model performance in the form of a matrix, hence also known as an error matrix. Some features of Confusion matrix are given below:

- For binary classifier, the matrix is of 2*2 table, for 3 classes classifier, it is 3*3 table, and so on
- The matrix is divided into two dimensions, that are predicted values and actual values along with the total number of predictions.
- Predicted values are those values, which are predicted by the model, and actual values are the true values for the given observations. It looks like the below table:

Total Predictions	**Actual: Yes**	**Actual: No**
Predicted: Yes	True Positive	False Positive
Predicted: No	False Negative	True Negative

✓ ***True Positive:*** The model has predicted yes, and the actual value was also true.

✓ ***True Negative:*** Model has given prediction No, and the real or actual value was also No.

× ***False Negative:*** The model has predicted no, but the actual value was Yes, it is also called as Type-II error.

× ***False Positive:*** The model has predicted Yes, but the actual value was No. It is also called a Type-I error.

Need for Confusion Matrix in Machine learning

It evaluates the performance of the classification models, when they make predictions on the data, and tells how good our classification model is.

It not only tells the error made by the classifiers but also the type of errors such as it is either type-I or type-II error.

With the help of the confusion matrix, we can calculate the different parameters for the model, such as accuracy, precision, etc.

In [25]:
```
# import confusion matrix class
from sklearn.metrics import confusion_matrix
# predictions for training data
y_train_pred = model.predict(X_train_count)
# create a confusion matrix
cm = confusion_matrix(y_train, y_train_pred)
cm
```

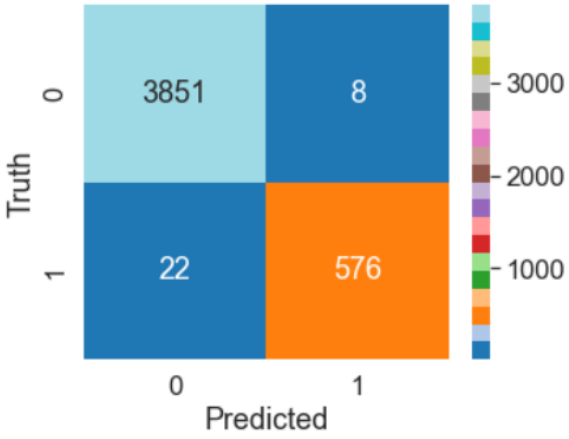
Out[25]:
```
array([[3851,    8],
       [  22,  576]], dtype=int64)
```

In [26]:
```
# install seaborn library for visualization of confusion matrix
pip install seaborn
```

```
Requirement already satisfied:
```

In [27]:
```
# visualize confusion matrix
import seaborn as sn
plt.figure(figsize = (5, 4))
sn.set(font_scale = 1.5)
sn.heatmap(cm, cmap="tab20", annot = True, fmt="d")
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[27]: Text(16.5, 0.5, 'Truth')



Calculations using Confusion Matrix

We can perform various calculations for the model, such as the model's accuracy, using this matrix. These calculations are given below:

1. ***Classification Accuracy:*** It is one of the important parameters to determine the accuracy of the classification problems. It defines how often the model predicts the correct output. It can be calculated as the ratio of the number of correct predictions made by the classifier to all number of predictions made by the classifiers. The formula is given below:

$$\boldsymbol{Accuracy} = \frac{\boldsymbol{TP} + \boldsymbol{TN}}{\boldsymbol{TP} + \boldsymbol{FP} + \boldsymbol{FN} + \boldsymbol{TN}}$$

In [28]:
```
# training accuracy
from sklearn.metrics import accuracy_score
accuracy_score(y_train, y_train_pred)
```

Out[28]: 0.9932690150325331

2. ***Misclassification rate:*** It is also termed as Error rate, and it defines how often the model gives the wrong predictions. The value of error rate can be calculated as the number of incorrect predictions to all number of the predictions made by the classifier. The formula is given below:

$$\boldsymbol{Error\ Rate} = \frac{\boldsymbol{FP} + \boldsymbol{FN}}{\boldsymbol{TP} + \boldsymbol{FP} + \boldsymbol{FN} + \boldsymbol{TN}}$$

In [29]:
```
# training error rate
1 - accuracy_score(y_train, y_train_pred)
```

Out[29]: 0.006730984967466935

In spam detection scenario False Positive means the mail is normal but it is predicted as a spam. So we need minimum False Positive value (Type II error) because the user might miss an important mail but predicted as a spam. Whenever FP is much more important, we need to evaluate precision of our model.

3. ***Precision / Positive Prediction Value:*** It can be defined as out of all positive classes that have predicted correctly by the model, how many of them were actually true. It can be calculated using the below formula:

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FP}}$$

In [29]:
```
# training precision
from sklearn.metrics import precision_score
precision_score(y_train, y_train_pred)
```

Out[29]: 0.9863013698630136

Suppose we are predicting whether a patient is having cancer or not, Here False Negative means the patient is having cancer but it is predicted as not having cancer. So we need minimum False Negative value (Type I error) because the patient might lost due to cancer but predicted as cancer-free. Whenever FP is much more important, we need to evaluate precision of our model.

4. ***Recall / Sensitivity / True Positive Rate:*** It is defined as the out of total positive classes, how our model predicted correctly. The recall must be as high as possible.

$$\boldsymbol{Recall\ (TPR)} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FN}}$$

In [30]:
```
# training recall
from sklearn.metrics import recall_score
recall_score(y_train, y_train_pred)
```

Out[30]: 0.9632107023411371

Whenever both FP and FN are highly important, we need to evaluate both precision and recall of our model using harmonic mean of them called f1-score.

5. ***F-measure:*** If two models have low precision and high recall or vice versa, it is difficult to compare these models. So, for this purpose, we can use F-score. This score helps us to evaluate the recall and precision at the same time. The F-score is maximum if the recall is equal to the precision. It can be calculated using the below formula:

$$\boldsymbol{F1\_score} = \frac{\boldsymbol{2 * Recall * Precision}}{\boldsymbol{Recall + Precision}}$$

In [31]:
```
# training f1-score
from sklearn.metrics import f1_score
f1_score(y_train, y_train_pred)
```

Out[31]: 0.9746192893401014

Classification Report

A classification report is an overall performance evaluation metric in machine learning. It is used to show the accuracy, precision, recall, F1 Score, and support for each target class.

You can refer the following article to know more about classification report:

https://towardsdatascience.com/understanding-data-science-classification-metrics-in-scikit-learn-in-python-3bc336865019

In [32]:
```
# classification report
from sklearn.metrics import classification_report
print(classification_report(y_train, y_train_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      3859
           1       0.99      0.96      0.97       598

    accuracy                           0.99      4457
   macro avg       0.99      0.98      0.99      4457
weighted avg       0.99      0.99      0.99      4457
```

Evaluation of our classification model using Test data

So far we were analysing performance of our trained model using only training data. Now we will evaluate the model using our test data.

In [33]:
```
# count vectorise or transform test data
X_test_count = v.transform(X_test.values)
# convert transformed data to an array
X_test_count.toarray()
# print three sample array elements from transformed data
X_test_count[:3]
```

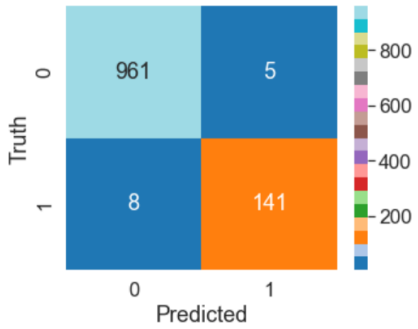
Out[33]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

In [34]:
```
# import confusion matrix class
from sklearn.metrics import confusion_matrix
# predictions for test data
y_test_pred = model.predict(X_test_count)
# create a confusion matrix
cm = confusion_matrix(y_test, y_test_pred)
cm
```

Out[34]:
```
array([[961,    5],
       [  8,  141]], dtype=int64)
```

In [35]:
```
# visualize confusion matrix for predictions of test data
import seaborn as sn
plt.figure(figsize = (5, 4))
sn.set(font_scale = 1.5)
sn.heatmap(cm, cmap="tab20", annot = True, fmt="d")
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

```
Out[35]: Text(16.5, 0.5, 'Truth')
```




```
In [36]:
1 # test accuracy
2 accuracy_score(y_test, y_test_pred)
```

```
Out[36]: 0.9883408071748879
```

```
In [37]:
1 # test error rate
2 1 - accuracy_score(y_test, y_test_pred)
```

```
Out[37]: 0.011659192825112075
```

```
In [38]:
1 # test precision
2 precision_score(y_test, y_test_pred)
```

```
Out[38]: 0.9657534246575342
```

```
In [39]:
1 # test recall
2 recall_score(y_test, y_test_pred)
```

```
Out[39]: 0.9463087248322147
```

```
In [40]:
1 # test f1-score
2 f1_score(y_test, y_test_pred)
```

```
Out[40]: 0.9559322033898304
```

```
In [41]:
1 # classification report
2 print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       966
           1       0.97      0.95      0.96       149

    accuracy                           0.99      1115
   macro avg       0.98      0.97      0.97      1115
weighted avg       0.99      0.99      0.99      1115
```

There are other advanced performance evaluation metrics of classification model, since those are beyond our scope we can't include it here. please go through the following articles to know more.

ROC Curve: *https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5*

Lift and Gain Chart: *https://towardsdatascience.com/model-benefit-evaluation-with-lift-and-gain-analysis-4b69f9288ab3*

Exporting our model as a file

We can simply export our model using pickle or joblib library then load for our prediction later on.

Save model using pickle – Recommended for smaller dataset

In [42]:
```
# exporting model as a pickle file
import pickle
with open ('spam_detector', 'wb') as f:
      pickle.dump(model, f)
```

In [43]:
```
# Load model using pickle
with open ('spam_detector', 'rb') as f:
      model_object = pickle.load(f)
model_object
```

Out[43]: MultinomialNB()

Save model using joblib – Recommended for larger dataset

In [44]:
```
# exporting model using joblib
import joblib
joblib.dump(model, "spam_detector_v2")
```

Out[44]: ['spam_detector_v2']

In [45]:
```
# Load model using joblib
model_object = joblib.load('spam_detector_v2')
model_object
```

Out[45]: MultinomialNB()

Implement our model using code snippet ready for production

In [46]:
```
# classify new mail in to ham and spam.
sent = input("Write your mail...\n")
emails = []
emails.append(sent)
emails_count = v.transform(emails)
pred = model.predict(emails_count)
if pred[0] == 0:
    print("This mail is ham.")
elif pred[0] == 1:
    print("This mail is spam.")
```

```
Write your mail...
  win now free prize gift up to $100.
This mail is spam.
```

Key takeaways

- Probability is the chance that a given event will occur.
- Conditional probability is the probability of an event (A), given that another (B) has already occur.
- Naïve Bayes classifier is a ML classifier which is based on probabilistic logic that is Bayes theorem.
- Bernoulli NB, Multinomial NB and Gaussian NB are types of Naïve Bayes classifier.

- Pandas is an open-source library that provides high-performance data manipulation in Python.
- Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python.

- Encoding is the conversion of categorical features to numeric values.
- Label (Ordinal) Encoding refers to converting ordinal categorical data into a numeric form.
- OneHot Encoding refers to converting nominal categorical data into a numeric form.
- Dummy variables are values 0 or 1 representing the presence or absence of categorical data.
- The dummy variable trap is a scenario in which two or more variables are highly correlated.

- To train and evaluate our classifier we need to split our dataframe into training and test set.
- Sampling bias is a bias that occurs when some members of the intended population are not selected.
- CountVectorizer is a tool to transform a given text into a vector based on frequency of each word.

- Cross-validation is a resampling procedure used to evaluate ML models on a limited data sample.
- Confusion matrix is a matrix used to determine the performance of the classification models.
- Accuracy, precision, recall and f1-score are main classification performance metrics.

- Precision is the ratio of actual true & all positive classes that model predicted correctly.
- Recall is defined as out of total positive classes, how many of them predicted correctly.
- F1-score helps us to evaluate the recall and precision at the same time.
- ROC curve, Lift and Gain chart are other advanced performance metrics for classification models.
- A classification report is overall performance (accuracy, precision, recall, F1, support) for each class.

- We can simply export your model using pickle or joblib then use it later.