KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY

(AN AUTONOMOUS INSTITUTE)

Accredited by NBA & NAAC, Approved by AICTE, Affiliated to JNTUH, Hyderabad

B.Tech. in COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

IV Year I Semester Syllabus (KR21)

WEB AND SOCIAL MEDIA ANALYTICS LAB (21CD732PE)

Pre-requisites/ Co-requisites:

1. PP207ES:Python Programming Lab Course
2. 21CS306PC: Machine Learning Using Python Lab

L	T	P	C
3	0	0	3

Course Objectives: The course will help to

1. Learn decision support systems.
2. Learn language processing concepts on text analytics.
3. Learn sentiment analysis.
4. Learn Web Analytics, Web Mining
5. Learn search engine optimization and web analytics.

Course Outcomes: After learning the concepts of this course, the student is able to

1. Understand decision support systems.
2. Apply natural language processing concepts on text analytics.
3. Understand sentiment analysis.
4. Understand Web Analytics, Web Mining
5. Knowledge on search engine optimization and web analytics.

List of Experiments

1. Preprocessing text document using NLTK of Python

a. Stop word elimination

b. Stemming

c. Lemmatization

d. POS tagging

e. Lexical analysis

2. Sentiment analysis on customer review on products

3. Web analytics

a. Web usage data (web server log data, clickstream analysis)

b. Hyperlink data

4. Search engine optimization- implements spamdexing

5. Use Google analytics tools to implement the following

a. Conversion Statistics

b. Visitor Profiles

6. Use Google analytics tools to implement the Traffic Sources.

8. Data Processing Techniques:

(i) Data cleaning (ii) Data transformation – Normalization (iii) Data integration

9. Implementation of Decision Tree algorithm

10. Classification of data using logistic regression

11. Classification of data using K – nearest neighbor approach

12. Implementation of K – means algorithm.

Resources:
1. Stanford core NLP package
2. GOOGLE.COM/ANALYTICS

TEXT BOOKS:

1. Ramesh Sharda, Dursun Delen, Efraim Turban, BUSINESS INTELLIGENCE AND ANALYTICS: SYSTEMS FOR DECISION SUPPORT, Pearson Education.

REFERENCE BOOKS:

1. Rajiv Sabherwal, Irma Becerra- Fernandez," Business Intelligence –Practice, Technologies and Management", John Wiley 2011.
2. Lariss T. Moss, Shaku Atre, "Business Intelligence Roadmap", Addison-Wesley It Service.
3. Yuli Vasiliev, "Oracle Business Intelligence: The Condensed Guide to Analysis and Reporting", SPD Shroff, 2012.

EXPERIMENT 1:

1. Preprocessing text document using NLTK of Python
a. Stop word elimination
b. Stemming
c. Lemmatization
d. POS tagging
e. Lexical analysis

1. Preprocessing text document using NLTK of Python

a. Stop word elimination b. Stemming c. Lemmatization d. POS tagging e. Lexical analysis **

Natural Language Toolkit (NLTK) is one of the largest Python libraries for performing various Natural Language Processing tasks. From rudimentary tasks such as text pre-processing to tasks like vectorized representation of text – NLTK's API has covered everything. In this article, we will accustom ourselves to the basics of NLTK and perform some crucial NLP tasks: Tokenization, Stemming, Lemmatization, and POS Tagging.

What is the Natural Language Toolkit (NLTK)? As discussed earlier, NLTK is Python's API library for performing an array of tasks in human language. It can perform a variety of operations on textual data, such as classification, tokenization, stemming, tagging, Leparsing, semantic reasoning, etc.

Installation: NLTK can be installed simply using pip or by running the following code.

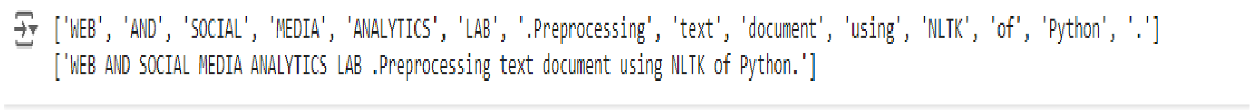
```
! pip install nltk
import nltk
nltk.download('all')
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Download necessary resources
nltk.download('stopwords')
nltk.download('punkt')
```

Tokenization Tokenization refers to break down the text into smaller units. It entails splitting paragraphs into sentences and sentences into words. It is one of the initial steps of any NLP pipeline. Let us have a look at the two major kinds of tokenization that NLTK provides:

Work Tokenization It involves breaking down the text into words.

```
# Tokenization using NLTK
from nltk import word_tokenize, sent_tokenize
sent = "WEB AND SOCIAL MEDIA ANALYTICS LAB .\
Preprocessing text document using NLTK of Python."
print(word_tokenize(sent))

+print(sent_tokenize(sent))
```

OUTPUT:

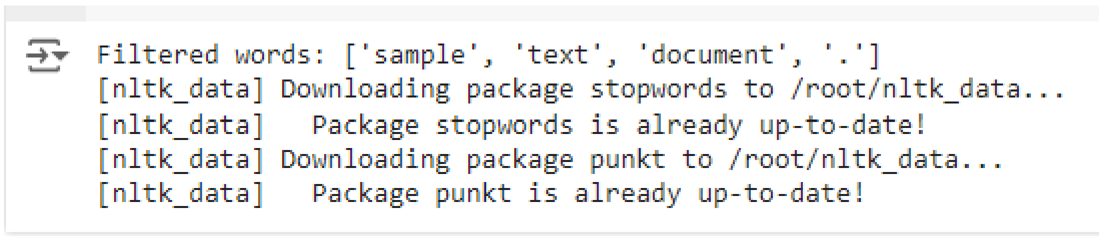


```
['WEB', 'AND', 'SOCIAL', 'MEDIA', 'ANALYTICS', 'LAB', '.Preprocessing', 'text', 'document', 'using', 'NLTK', 'of', 'Python', '.']
['WEB AND SOCIAL MEDIA ANALYTICS LAB .Preprocessing text document using NLTK of Python.']
```

A. Stop Word Elimination: Remove common words that don't contribute significant meaning.

```
# Sample text
text = "This is a sample text document."

# Tokenize and remove stop words
words = word_tokenize(text)
filtered_words = [word for word in words if word.lower() not in
stopwords.words('english')]

print("Filtered words:", filtered_words)
```

OUTPUT:

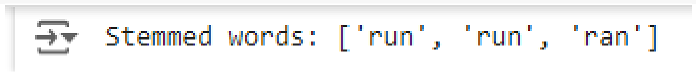


```
Filtered words: ['sample', 'text', 'document', '.']
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

B. Stemming: Reduce words to their root form using algorithms like the Porter Stemmer.For typical stemming results using Porter Stemmer, "running" will be reduced to "run," which is the expected behavior.

```
from nltk.stem import PorterStemmer
```

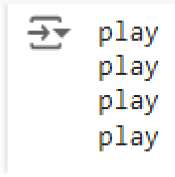
```
# Create stemmer object
stemmer = PorterStemmer()

# Sample words
words = ["running", "runs", "ran"]

# Apply stemming
stemmed_words = [stemmer.stem(word) for word in words]

print("Stemmed words:", stemmed_words)
```
OUTPUT:



```
Stemmed words: ['run', 'run', 'ran']
```

C. Lemmatization: Reduce words to their base or dictionary form using the WordNet Lemmatizer.

```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.tokenize import word_tokenize
import nltk

# Download necessary resources
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
from nltk.stem import WordNetLemmatizer
# create an object of class WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize("plays", 'v'))
print(lemmatizer.lemmatize("played", 'v'))
print(lemmatizer.lemmatize("play", 'v'))
print(lemmatizer.lemmatize("playing", 'v'))
```
OUTPUT:



```
play
play
play
play
```

```
Key Differences
Purpose:

Stemming: Focuses on reducing words to a base form quickly but may create non-
standard or inaccurate root forms.
Lemmatization: Aims to convert words to their correct base forms (lemmas) using
context and POS, resulting in more accurate and meaningful base forms.
Output:
```

```
Stemming: Results may be non-words (e.g., "runn" from "running").
Lemmatization: Results in valid dictionary words (e.g., "run" from "running").
Complexity:

Stemming: Less complex, faster, and more suited for simple tasks.
Lemmatization: More complex and slower but produces more accurate results.
```

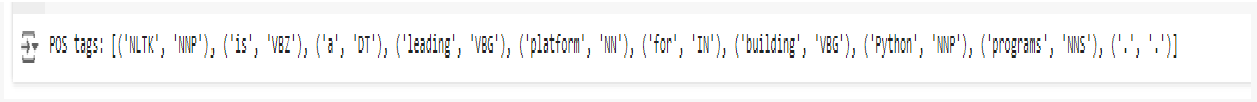
```
POS (Part-of-Speech) tags represent the grammatical categories of words in a
sentence. Each tag denotes a specific role that a word plays within the
structure of a sentence. Here’s a brief explanation of the POS tags you
mentioned:

Common POS Tags
1. *NNP (Proper Noun, Singular):*

Represents names of specific people, places, or organizations.
Example: "NLTK" and "Python" in your example.

2. *VBZ (Verb, 3rd Person Singular Present):*

Indicates a verb in the present tense that is used with third-person singular
subjects.
Example: "is" in your example.

3. *DT (Determiner):*

Used to introduce nouns and specify which noun is being referred to.
Example: "a" in your example.

4. *NN (Noun, Singular or Mass):*

Represents a singular noun that refers to a general, non-specific item or
concept.
Example: "platform" in your example.

5. *VBG (Verb, Gerund or Present Participle):*

Represents a verb form ending in -ing that functions as a noun or as part of a
continuous tense.
Example: "leading" and "building" in your example.

6. *IN (Preposition or Subordinating Conjunction):*

Represents words used to show relationships between other words or phrases.
Example: "for" in your example.
```

```
7. *NNS (Noun, Plural):*

Represents a noun in its plural form.
Example: "programs" in your example.
```

D. POS Tagging: Label words with their part of speech.

```
import nltk
from nltk.tokenize import word_tokenize

# Download necessary resources
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')
# Sample text
text = "NLTK is a leading platform for building Python programs."

# Tokenize and POS tagging
words = word_tokenize(text)
pos_tags = nltk.pos_tag(words)

print("POS tags:", pos_tags)
```

OUTPUT:

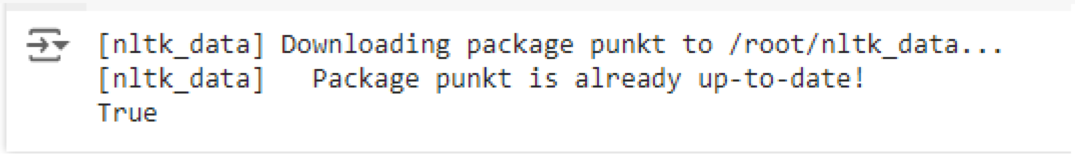


```
POS tags: [('NLTK', 'NNP'), ('is', 'VBZ'), ('a', 'DT'), ('leading', 'VBG'), ('platform', 'NN'), ('for', 'IN'), ('building', 'VBG'), ('Python', 'NNP'), ('programs', 'NNS'), ('.', '.')]
```

E. Lexical Analysis: Breaks text into tokens for analysis.Tokens are the individual units into which text is broken down during the process of text analysis. In natural language processing (NLP), tokens are typically the smallest meaningful elements of text and can be words, phrases, or even punctuation marks

```
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize

# Download necessary resources
nltk.download('punkt')
```

OUTPUT:

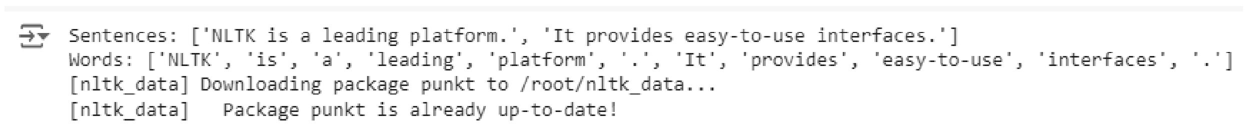


```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
# Sample text
text = "NLTK is a leading platform. It provides easy-to-use interfaces."
```

```
# Tokenize sentences and words
sentences = sent_tokenize(text)
words = word_tokenize(text)

print("Sentences:", sentences)
print("Words:", words)
```

OUTPUT:



```
Sentences: ['NLTK is a leading platform.', 'It provides easy-to-use interfaces.']
Words: ['NLTK', 'is', 'a', 'leading', 'platform', '.', 'It', 'provides', 'easy-to-use', 'interfaces', '.']
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

import spacy

# Load the small English model

nlp = spacy.load("en_core_web_sm")

# Sample text

text = "spaCy is a powerful NLP library."

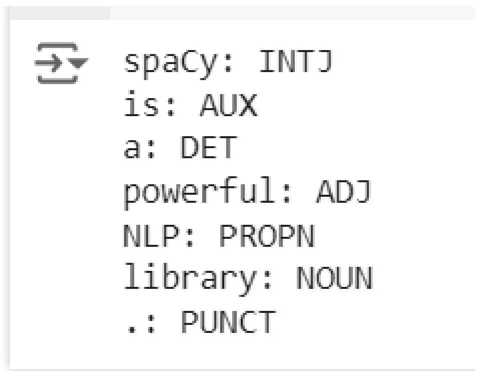
# Process the text

doc = nlp(text)

# Extract and print tokens and their POS tags

for token in doc:

    print(f"{token.text}: {token.pos_}")

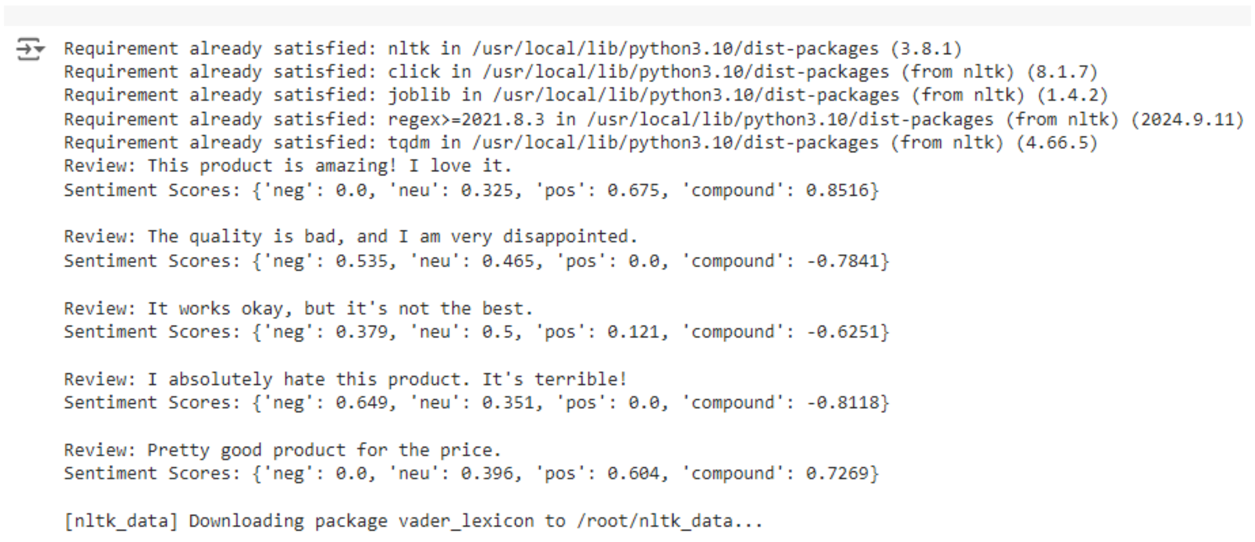


```
spaCy: INTJ
is: AUX
a: DET
powerful: ADJ
NLP: PROPN
library: NOUN
.: PUNCT
```

EXPERIMENT 2:
Sentiment analysis on customer review on products

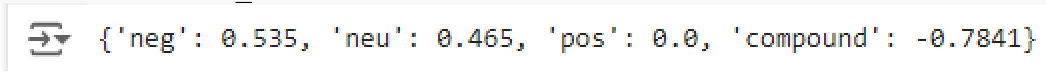
```
#Install the necessary libraries:
!pip install nltk
#Download NLTK resources (only once):
import nltk
nltk.download('vader_lexicon')



# Import necessary libraries
from nltk.sentiment import SentimentIntensityAnalyzer

# Initialize the Sentiment Analyzer
sia = SentimentIntensityAnalyzer()

# Sample customer reviews
reviews = [
    "This product is amazing! I love it.",
    "The quality is bad, and I am very disappointed.",
    "It works okay, but it's not the best.",
    "I absolutely hate this product. It's terrible!",
    "Pretty good product for the price."
]

# Analyze sentiment for each review
for review in reviews:
    sentiment_score = sia.polarity_scores(review)
    print(f"Review: {review}")
    print(f"Sentiment Scores: {sentiment_score}\n")
```

OUTPUT:



```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.8.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2024.9.11)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.5)
Review: This product is amazing! I love it.
Sentiment Scores: {'neg': 0.0, 'neu': 0.325, 'pos': 0.675, 'compound': 0.8516}

Review: The quality is bad, and I am very disappointed.
Sentiment Scores: {'neg': 0.535, 'neu': 0.465, 'pos': 0.0, 'compound': -0.7841}

Review: It works okay, but it's not the best.
Sentiment Scores: {'neg': 0.379, 'neu': 0.5, 'pos': 0.121, 'compound': -0.6251}

Review: I absolutely hate this product. It's terrible!
Sentiment Scores: {'neg': 0.649, 'neu': 0.351, 'pos': 0.0, 'compound': -0.8118}

Review: Pretty good product for the price.
Sentiment Scores: {'neg': 0.0, 'neu': 0.396, 'pos': 0.604, 'compound': 0.7269}

[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

```python
review = "The quality is bad, and I am very disappointed."
sentiment_score = sia.polarity_scores(review)
print(sentiment_score)
```



```
{'neg': 0.535, 'neu': 0.465, 'pos': 0.0, 'compound': -0.7841}
```

EXPERIMENT 9

Implementation of Decision Tree algorithm

Decision Tree Algorithm

• Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems.

• It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**

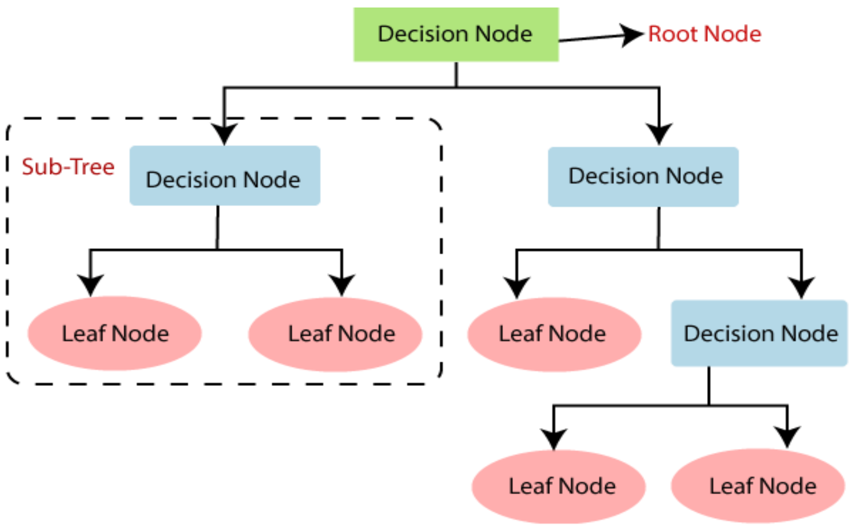
• In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

• **It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.**

• It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

• In order to build a tree, we use the **CART algorithm,** which stands for **Classification and Regression Tree algorithm.**

- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.
- The general structure of a decision tree



- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

Why use Decision Trees?

- There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:
- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

How does the Decision Tree algorithm Work?

- In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.
- For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:
- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Algorithms

- **CART (Gini Index)**
- **ID3 (Entropy, Information Gain)**
- While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:
    - **Information Gain**
    - **Gini Index**

Information Gain

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:
- Information Gain= Entropy(S)-[(Weighted Avg) *Entropy(each feature)
- **Entropy:** Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:
- $\text{Entropy}(s) = -P(\text{yes})\log_2 P(\text{yes}) - P(\text{no}) \log_2 P(\text{no})$

Gini Index

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

$$\text{Gini Index} = 1 - \sum_j P_j^2$$

Pruning

- Pruning: Getting an Optimal Decision tree
- Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.
- A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:
  - Cost Complexity Pruning
  - Reduced Error Pruning**.**

ID3 ALGORITHM

- ID3 stands for Iterative Dichotomiser 3 and is named such because the algorithm iteratively (repeatedly) dichotomizes(divides) features into two or more groups at each step.
- Invented by Ross Quinlan, ID3 uses a **top-down greedy** approach to build a decision tree. In simple words, the **top-down** approach means that we start building the tree from the top and the **greedy** approach means that at each iteration we select the best feature at the present moment to create a node.
- Most generally ID3 is only used for classification problems with nominal features only.
- ID3 algorithm selects the best feature at each step while building a Decision tree.
  How does ID3 select the best feature?
- ID3 uses **Information Gain** or just **Gain** to find the best feature.
- Information Gain calculates the reduction in the entropy and measures how well a given feature separates or classifies the target classes. The feature with the **highest Information Gain** is selected as the **best** one
- In simple words, **Entropy** is the measure of disorder and the Entropy of a dataset is the measure of disorder in the target feature of the dataset.
  In the case of binary classification (where the target column has only two types of classes) entropy is **0** if all values in the target column are homogenous(similar) and will be **1** if the target column has equal number values for both the classes.

Program :(: tennis.csv)

```
import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

#numpy and pandas initialization

import numpy as np  import
pandas as pd

PlayTennis = pd.read_csv("tennis.csv")

#numerical values into numerical values using LabelEncoder
from sklearn.preprocessing import LabelEncoder

Le = LabelEncoder()

PlayTennis['outlook'] = Le.fit_transform(PlayTennis['outlook'])
PlayTennis['temp'] = Le.fit_transform(PlayTennis['temp'])
PlayTennis['humidity'] = Le.fit_transform(PlayTennis['humidity'])
PlayTennis['windy'] = Le.fit_transform(PlayTennis['windy'])
PlayTennis['play'] = Le.fit_transform(PlayTennis['play'])

features_cols=['outlook','temp','humidity','windy']
x=PlayTennis[features_cols]

y=PlayTennis.play

from sklearn.model_selection import train_test_split

x_train, x_test, y_train,y_test=train_test_split(x,y,test_size=0.2)  from
sklearn.tree import DecisionTreeClassifier

#classifier=DecisionTreeClassifier(criterion='gini')
classifier=DecisionTreeClassifier(criterion='entropy')
classifier.fit(x_train,y_train)
DecisionTreeClassifier(criterion='entropy')
classifier.predict(x_test)  classifier.score(x_test,y_test)

from sklearn import tree

clf = tree.DecisionTreeClassifier(criterion = 'entropy')

 clf = clf.fit(x, y)

tree.plot_tree(clf)
```

10. Classification of data using logistic regression

```
#Import required libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O  import os
#Read dataset into pandas dataframe.
data=pd.read_csv('Iris.csv')
data.head()
# Display the datatype of each column  data.dtypes
# Create a column cat_code which represents the numerical values for Species column.
data['cat_code']=data.Species.astype('category').cat.codes
data.head()  data.columns
#Select the input features as x
x=data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm','PetalWidthCm']]
#Select the target
y=data['cat_code']


# Visualize the 3 categories of iris flower with respective to  Sepal Length
and Sepal Width
import matplotlib.pyplot as plt
plt.scatter(x['SepalLengthCm'],x['SepalWidthCm'],c=y)
# Visualize the 3 categories of iris flower with respective to  Petal Length and Petal Width

plt.scatter(x['PetalLengthCm'],x['PetalWidthCm'],c=y)  x.shape,y.shape
# Splitting train and test datasets
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,shuffle=True)
x_train.shape,x_test.shape,y_train.shape,y_test.shape


# Fit the logistic regression model with SKlearn library
```

```
from sklearn.linear_model import LogisticRegression
lm=LogisticRegression()
lm.fit(x_train,y_train)


# Prediction for train set
y_pred = lm.predict(x_train)
y_train.values
y_pred


# Accuracy for train dataset
from sklearn.metrics import accuracy_score
# Prediction for test dataset and Accuracy
y_pred = lm.predict(x_test)
accuracy_score(y_test,y_pred)
```

EXPERIMENT 11

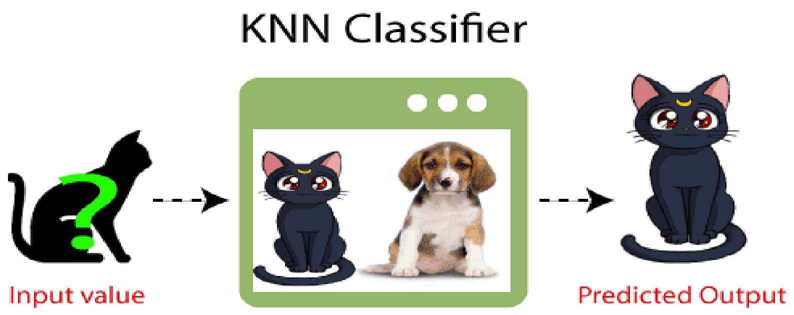
Classification of data using K – nearest neighbor approach

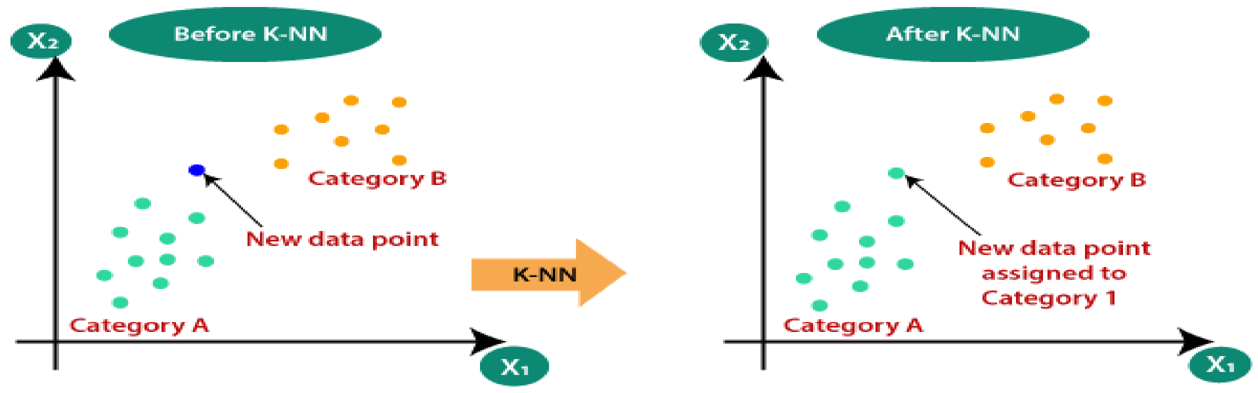
KNN algorithm

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- **K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.**
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Lazy learning algorithm** − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.
- **Non-parametric learning algorithm** − KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

Example

- Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

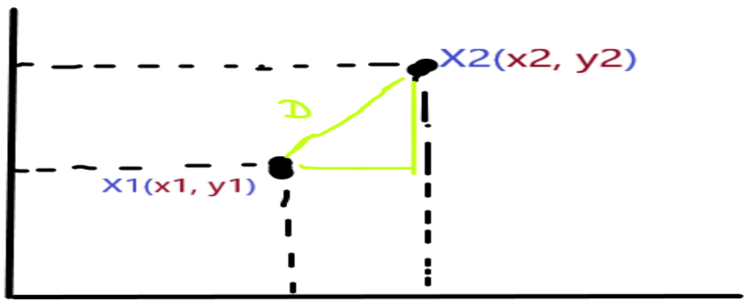




Euclidean distance

We mostly use this distance measurement technique to find the distance between consecutive points. It is generally used to find the distance between two real-valued vectors. Euclidean distance is used when we have to calculate the distance of real values like integer, float, etc… One issue with this distance measurement technique is that we have to normalize or standard the data into a balance scale otherwise the outcome will not preferable.

Let's take an example of a 2-Dimensional vector and take a geometrical intuition of distance measures on 2-Dim data for a better understanding.



From the above image, you can see that there are 2-Dim data X_1 and X_2 placed at certain coordinates in 2 dimensions, suppose X_1 is at (x_1,y_1) coordinates and X_2 is at (x_2,y_2) coordinates.

We have 2-dim data so we considered F1 and F2 two features and D is considered as the shortest line from X_1 and X_2, If we find the distance between these data points that can be found by the Pythagoras theorem and can be written as:

$$D = \sqrt{[(x2 - x1)^2 + (y2 - y1)^2]}$$

You already know that we are finding the distance or we can say the length of any 2 points, so we in vectorize from we can write that as $d = \| X1 - X2 \|$

Suppose we take D-dimensional vector it means X_1 belongs to R^d ($X_1 \in R^d$) and also X_2 belongs to R^d ($X_2 \in R^d$) so the distance between X1 and X2 is written as:

$$d = \sqrt{\sum_{i=1}^{N} (Xi - Yi)^\wedge 2}$$

Example: K nearest neighbor classifier to predict the diabetic patient with the given features BMI, Age. If the training examples are

BMI	Age	Sugar
33.6	50	1
26.6	30	0
23.4	40	0
43.1	67	0
35.3	23	1
35.9	67	1
36.7	45	1
25.7	46	0
23.3	29	0
31	56	1

0-Non-diabetic

1-Diabetic

Test Example BMI=43.6, Age=40, Sugar=?

The given training dataset has 10 instances with two features BMI (Body Mass Index) and Age. Sugar is the target label. The target label has two possibilities 0 and 1. 0 means the diabetic patient has no sugar and 1 means the diabetic patient has sugar.

Given the dataset and new test instance, we need to find the distance from the new test instance to every training example. Here we use the euclidean distance formula to find the distance.

$$Distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

If we choose K=3 then we need to select three neighbours which are close to the given test data point i.e**,(43.6,40**).

BMI	Age	Sugar	Formula	Distance
33.6	50	1	√((43.6-33.6)^2+(40-50)^2)	14.14
26.6	30	0	√((43.6-26.6)^2+(40-30)^2)	19.72
23.4	40	0	√((43.6-23.4)^2+(40-40)^2)	20.20
43.1	67	0	√((43.6-43.1)^2+(40-67)^2)	27.00
35.3	23	1	√((43.6-35.3)^2+(40-23)^2)	18.92
35.9	67	1	√((43.6-35.9)^2+(40-67)^2)	28.08
36.7	45	1	√((43.6-36.7)^2+(40-45)^2)	8.52
25.7	46	0	√((43.6-25.7)^2+(40-46)^2)	18.88
23.3	29	0	√((43.6-23.3)^2+(40-29)^2)	23.09
31	56	1	√((43.6-31)^2+(40-56)^2)	20.37

Select three data points based on minimum Euclidean distance

BMI	Age	Sugar
36.7	45	1
33.6	50	1
25.7	46	0

Now, we need to apply the majority voting technique to decide the resulting label for the test data point,**(43.6,40**).

As majority is classified as **1** then the **test data point** is classified as **1**(i.e, patient has sugar)

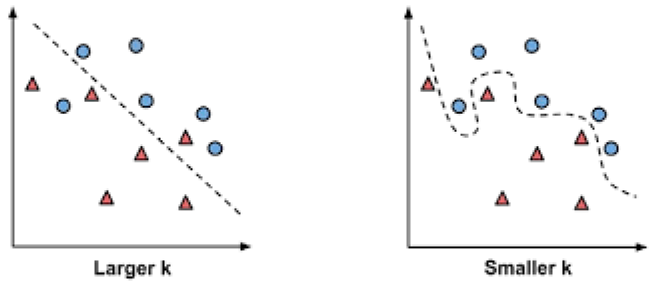
Test Example BMI=43.6, Age=40, Sugar=1

How to select the value of K in the K-NN Algorithm

- There is no particular way to determine the best value for "K", Choosing a right value of K is a process called **Hyperparameter Tuning.**
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.
- The impact of selecting a smaller or larger K value on the model

- **Larger K value:** The case of underfitting occurs when the value of k is increased. In this case, the model would be unable to correctly learn on the training data.

- **Smaller k value:** The condition of overfitting occurs when the value of k is smaller. The model will capture all of the training data, including noise. The model will perform poorly for the test data in this scenario.



Few ways of selecting K value

1. **Square Root Method**: Take square root of the number of samples in the training dataset.
2. **Cross Validation Method:** We should also use cross validation to find out the optimal value of K in KNN. Start with K=1, run cross validation (5 to 10 fold), measure the accuracy and keep repeating till the results become consistent.
   K=1, 2, 3... As K increases, the error usually goes down, then stabilizes, and then raises again. Pick the optimum K at the beginning of the stable zone. This is also called **Elbow Method.**
3. **Domain Knowledge** also plays a vital role while choosing the optimum value of K.
4. K should be an **odd number**

Advantages of KNN Algorithm

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm

- Always needs to determine the value of K which may be complex some time.

- The computation cost is high because of calculating the distance between the data points for all the training samples.

Program :

(IRIS DATASET)

```
# Import libraries

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import pandas as pd

import warnings

warnings.filterwarnings("ignore")

import seaborn as sns

import matplotlib.pyplot as plt

data=pd.read_csv("Iris.csv")

data.head()

data.info()

data.describe()

data.head()

data.tail()

data.isnull().sum()

data['Species'].value_counts()

data['cat_code']=data.Species.astype('category').cat.codes

#Select the input features as x

x=data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm','PetalWidthCm']]

#Select the target

y=data['cat_code']

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics

# Splitting train and test datasets

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,shuffle=True)

KNN=KNeighborsClassifier(n_neighbors=3)

KNN.fit(x_train,y_train)

prediction=KNN.predict(x_test)

print('The accuracy of the KNN

is',metrics.accuracy_score(prediction,y_test)*100, 'percent')

#Get accuracy. Note: In case of classification algorithms score method represents accuracy.

KNN.score(x_test,y_test)

print("KNN SCORE",KNN.score(x_test,y_test))

#import confusion_matrix

from sklearn.metrics import confusion_matrix

prediction=KNN.predict(x_test)

confusion_matrix(y_test,prediction)
```

Result:

The accuracy for KNN is

EXPERIMENT 12. Implementation of K – means algorithm

What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

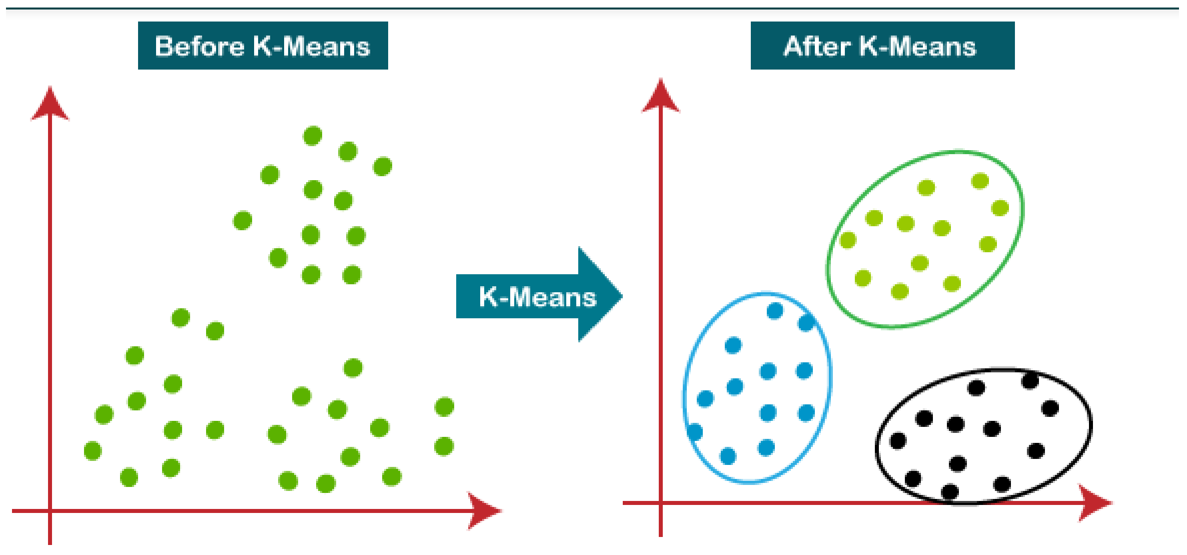
The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning#google_vignette

How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

The `make_blobs` function from the `sklearn.datasets` module is used to generate synthetic data for clustering tasks, especially to test clustering algorithms like K-means.

https://codinginfinite.com/k-means-clustering-explained-with-numerical-example/

Key Features of `make_blobs`:

1. **Creates Clustered Data**: It generates data points grouped into distinct clusters, which makes it useful for testing clustering algorithms.
2. **Customizable**: You can control the number of samples, centers (clusters), dimensionality, standard deviation (spread) of each cluster, and randomness.

Key Parameters:

- `n_samples`: The total number of data points to generate.
- `centers`: Number of clusters (or actual coordinates of the centers) to generate. If an integer is provided, it creates random cluster centers.
- `cluster_std`: Standard deviation (spread) of the clusters. A larger value makes the clusters more dispersed.
- `random_state`: Fixes the random generation of the dataset so you get the same output every time.

CODE

#import the KMeans class from the `sklearn.cluster` module

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs

# Step 1: Generate synthetic data

# **X**: Contains the coordinates of the generated data points (features).

# **y_**: Contains the cluster labels for each data point, indicating which cluster each point belongs to.

X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=0)

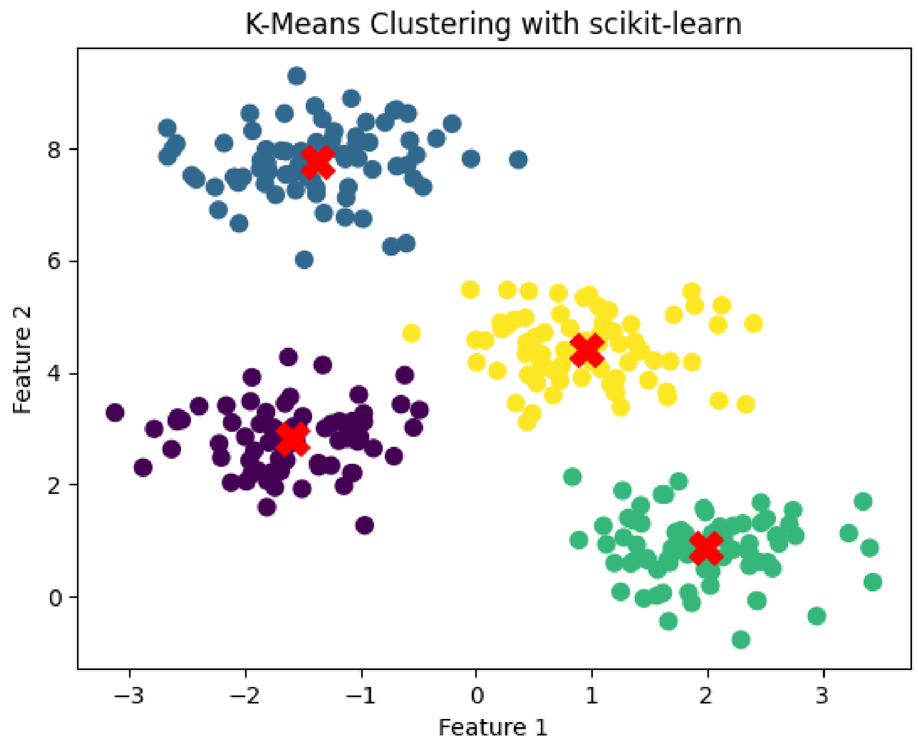
# Step 2: Use the KMeans class from scikit-learn

kmeans = KMeans(n_clusters=4) # Define the number of clusters, **Creating the KMeans Object**:

kmeans.fit(X) # Fit the model

```
# Step 3: Get the labels and centroids

labels = kmeans.labels_

centroids = kmeans.cluster_centers_


# Step 4: Visualization of the clusters

plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')

plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='red', marker='X')

plt.title('K-Means Clustering with scikit-learn')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()
```

•


K-Means Clustering with scikit-learn
Feature 2
Feature 1