TECHNEEDS - ML
WEEK 1

- Intro to Python
- Intro to github
- Intro to ML

Intro To Python

What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

- web development (server-side),
- software development,
- mathematics,
- system scripting.

What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).

- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

Execute Python Syntax

```
>>> print("Hello, World!")
Hello, World!
```

Python Indentation

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses indentation to indicate a block of code.

```
if 5 > 2:
  print("Five is greater than two!")
```

Syntax Error:

```
if 5 > 2:
print("Five is greater than two!")
```

Creating a Comment

Comments starts with a #, and Python will ignore them:

```
#print("Hello, World!")
print("Cheers, Mate!")
```

Multiline Comments

Python does not really have a syntax for multiline comments.

To add a multiline comment you could insert a # for each line:

```
#This is a comment
#written in
#more than just one line
print("Hello, World!")
```

Python Tokens

Tokens or lexical units are the smallest fractions in the python programme. A token is a set of one or more characters having a meaning together. There are 5 types of tokens in python which are listed below:

1. Keywords
2. Identifiers
3. Literals
4. Operators
5. Punctuators

1. Keywords

A keyword is a reserved word in a computer language that has a specific meaning. Python keywords form the vocabulary of the python language. Keywords aren't allowed to be used as identifiers. They are used to define the Python language's "Syntax" or "Structure."

There are as in all 33 keywords used in Python programming language version 3.7. Here are a few of them:

- and, not, or: logical Operators
- as: To create an alias
- assert: For debugging
- break: To break out of a loop
- if: To create a conditional statement
- while: To create a while loop

2. Identifiers

Just as identity refers to a characteristic that distinguishes a person, the same principle is a python identifier, a token in python. In Python, an identifier is a name given to a Class, Function, or Variable. It aids in distinguishing one entity from others.

Characteristics of Python Identifier

- The initial letter of the identifier should be any letter or underscore (_).
- Upper and lower case letters have distinct characteristics.
- Except for the initial letter, any digit from 0 to 9 can be part of the identification.
- It shouldn't be used as a keyword
- Except for the underscore (_), an identifier cannot contain any special characters.
- Identifiers can be as long as you want them to be.
- Case matters when it comes to identifier names. Myself and myself, for example, are not the same thing.

3. Operators

Operators are tokens that, when applied to variables and other objects in an expression, cause a computation or action to occur. Operands are the variables and objects to which the computation is applied. There are 7 different operators.

i)Arithmetic Operators

It performs all the mathematical calculations. Here are a few of them:

- (+) Operands on either right and left sides of the operator are added.
- (–) Subtract the right-hand operand from the left-hand operand with the subtraction operator.
- (✖️) operator – Multiplies both sides of the operator's operands.
- (➗) the left-hand operand by the right-hand operand with the division operator.
- (%) a percentage divides the left-hand operand by the right-hand operand and returns the remainder with the modulus operator.

ii) Relational Operators

A relational operator is a type of operator that examines the relationship between two operands. Some of the relational operators are:

- (==) Check if two operands' values are equal.
- (!=)Check if two operands' values are not equal.
- (>) Check if two operands' values are not identical (same as the!= operator).

iii) Assignment Operators

The assignment operators are employed to allocate a value to a variable. A few examples are:

- (+=)It adds the right side input to the left side input and then assigns the result to the left side input.
- (-=)Augmented assignment operator- It takes the right side operand and subtracts it from the left side operand, then assigns the result to the left side operand.

iv) Logical Operators

The logical operators compare two boolean expressions and yield a boolean result. Like

- The logical AND operator makes a condition true if both operands are true or non-zero.
- The logical OR operator returns true if one of the two operands is true or non-zero.

v) Bitwise Operators

The bitwise operator manipulates individual bits in one or more bit patterns or binary numbers. For example, If a binary XOR operator (^) is set in one input value but not both, it copies the matching binary 1 to the result.

vi) Membership Operators

The membership operator checks for membership in successions, such as a string, list, or tuple. Like in a membership operator that fetches a variable and if the variable is found in the supplied sequence, evaluate to true; otherwise, evaluate to false.

vii) Identity Operators

When comparing the memory locations of two objects, identity operators are used. If two variables point to separate objects, it does not return true; otherwise, it returns false.

4. Literals

Literals, tokens in Python, are data elements with a fixed value. Literals return a value for an object of the specified type. Python supports a variety of literals:

- String Literals
- Numeric Literals. These are further of three types, integer, float, and complex literals.
- Boolean Literals
- Literal Collection

Lists, tuples, dictionaries, and sets are all examples of literal collections in Python.

- A list is a collection of elements enclosed in square brackets and separated by commas. These variables can be of any data type, and their values can be altered.
- Tuple: A comma-separated list of elements or values in round brackets is also known as a tuple. Values can be of any data type, but they cannot be modified.
- Dictionary: It’s an unsorted collection of key-value pairs.
- The “set” is an unordered collection of objects enclosed in curly braces.

5. Punctuators

Punctuators are tokens in python employed to put the grammar and structure of syntax into practice. Punctuators are symbols that are used to structure programming sentences in a computer language. Some commonly used punctuators are: ‘, ‘ ,#, \ ,( ) ,{ },[ ] ,@ ,: , =

Python Data Types

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

Text Type:	`str`
Numeric Types:	`int`, `float`, `complex`
Sequence Types:	`list`, `tuple`, `range`
Mapping Type:	`dict`
Set Types:	`set`, `frozenset`
Boolean Type:	`bool`
Binary Types:	`bytes`, `bytearray`, `memoryview`
None Type:	`NoneType`

Setting the Data Type

Tec

Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	memoryview
`x = None`	NoneType

Python Numbers

There are three numeric types in Python:

- `int`
- `float`
- `complex`

Variables of numeric types are created when you assign a value to them:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

To verify the type of any object in Python, use the `type()` function:

```
print(type(x))
print(type(y))
print(type(z))
```

Type Conversion

You can convert from one type to another with the `int()`, `float()`, and `complex()` methods:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)
```

Random Number

Python does not have a `random()` function to make a random number, but Python has a built-in module called `random` that can be used to make random numbers:

Introduction to NumPy

What is NumPy?

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It is open-source software.

Features of NumPy

NumPy has various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy in Python can also be used as an efficient multi-dimensional container of generic data. Arbitrary data types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Install Python NumPy

Numpy can be installed for **Mac** and **Linux** users via the following pip command:

```
pip install numpy
```

Windows does not have any package manager analogous to that in Linux or Mac. Please download the pre-built Windows installer for NumPy from here (according to your system configuration and Python version). And then install the packages manually.

Note: All the examples discussed below will not run on an **online IDE.**

Arrays in NumPy

NumPy&#x2019s main object is the homogeneous multidimensional array.

- It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.
- In NumPy, dimensions are called *axes*. The number of axes is *rank*.
- NumPy&#x2019s array class is called **ndarray**. It is also known by the alias **array**.

Example:

In this example, we are creating a two-dimensional array that has the **rank** of 2 as it has 2 **axes**. The first axis(dimension) is of length 2, i.e., the number of rows, and the second axis(dimension) is of length 3, i.e., the number of columns. The overall shape of the array can be represented as (2, 3)

Sure, here are some comprehensive notes on NumPy, a fundamental library for numerical computing in Python:

Importing NumPy

```
import numpy as np
```

Creating Arrays

1. **From Lists:**

```python
array = np.array([1, 2, 3, 4, 5])
```

2. **From Tuples:**

```python
array = np.array((1, 2, 3, 4, 5))
```

Array Attributes

- **ndim**: Number of dimensions

```python
array.ndim  # Example: 1 for a 1D array
```

- **shape**: Tuple of array dimensions

```python
array.shape  # Example: (5,) for a 1D array with 5 elements
```

- **size**: Total number of elements

```python
array.size  # Example: 5 for an array with 5 elements
```

- **dtype**: Data type of the elements

```python
array.dtype  # Example: dtype('int64')
```

Array Creation Routines

- **Zeros**:

```python
np.zeros((3, 4))  # Creates a 3x4 array filled with zeros
```

- **Ones**:

```python
np.ones((2, 3))  # Creates a 2x3 array filled with ones
```

Techn

- **Full**:

```python
np.full((2, 2), 7)  # Creates a 2x2 array filled with 7
```

- **Eye** (Identity Matrix):

```python
np.eye(3)  # Creates a 3x3 identity matrix
```

- **Arange**:

```python
np.arange(0, 10, 2)  # Creates an array [0, 2, 4, 6, 8]
```

- **Linspace**:

```python
np.linspace(0, 1, 5)  # Creates an array [0, 0.25, 0.5, 0.75, 1]
```

Array Indexing and Slicing

- **Indexing**:

```python
array[2]  # Access third element
```

- **Slicing**:

```python
array[1:4]  # Access elements from index 1 to 3
```

Array Operations

- Element-wise Operations:

```python
array + 2  # Adds 2 to each element
array * 3  # Multiplies each element by 3
array / 4  # Divides each element by 4
```

- Universal Functions (ufuncs):

```python
np.sqrt(array)  # Square root of each element
np.exp(array)   # Exponential of each element
np.sin(array)   # Sine of each element
```

Array Methods

Techn

- Sum:

```python
array.sum()  # Sum of all elements
```

- Mean:

```python
array.mean()  # Mean of all elements
```

- Standard Deviation:

```python
array.std()  # Standard deviation of all elements
```

Linear Algebra

Techn

- Dot Product:

```python
np.dot(array1, array2)  # Dot product of two arrays
```

- Matrix Multiplication:

```python
np.matmul(matrix1, matrix2)  # Matrix multiplication
```

- Determinant:

```python
np.linalg.det(matrix)  # Determinant of a matrix
```

- Inverse:

```python
np.linalg.inv(matrix)  # Inverse of a matrix
```

Random Module

- Random Samples:

```python
np.random.random((2, 3))  # 2x3 array of random samples from [0.0, 1.
```

- Random Integers:

```python
np.random.randint(0, 10, (2, 3))  # 2x3 array of random integers from
```

- Normal Distribution:

```python
np.random.normal(0, 1, (2, 3))  # 2x3 array of samples from a normal
```

Techne

Reshaping and Resizing

- Reshape:

```python
array.reshape((2, 5))  # Reshape array to 2x5
```

- Resize:

```python
array.resize((3, 2))  # Resize array to 3x2
```

Broadcasting

- **Broadcasting**: NumPy's ability to perform operations on arrays of different shapes.

```python
array = np.array([1, 2, 3])
array2 = np.array([[0], [1], [2]])
result = array + array2
```

Tech

Aggregation Functions

- Max:

```python
array.max()  # Maximum value
```

- Min:

```python
array.min()  # Minimum value
```

- Argmax:

```python
array.argmax()  # Index of maximum value
```

Tech

- Argmin:

```python
array.argmin()  # Index of minimum value
```

Practical Examples

- Mean of Rows/Columns:

```python
matrix.mean(axis=0)  # Mean of each column
matrix.mean(axis=1)  # Mean of each row
```

- Cumulative Sum:

```python
array.cumsum()  # Cumulative sum of elements
```

Introduction to Pandas

- **Pandas**: An open-source data analysis and manipulation library for Python.
- **Key Data Structures**: Series (1D) and DataFrame (2D).

Importing Pandas

```
import pandas as pd
```

Creating Data Structures

Series

- From a List:

```python
s = pd.Series([1, 3, 5, 7, 9])
```

- From a Dictionary:

```python
s = pd.Series({'a': 1, 'b': 2, 'c': 3})
```

Techne

DataFrame

- From a Dictionary of Lists:

python

```
df = pd.DataFrame({
    'A': [1, 2, 3],
    'B': [4, 5, 6],
    'C': [7, 8, 9]
})
```

- From a List of Dictionaries:

python

```
df = pd.DataFrame([
    {'A': 1, 'B': 2},
    {'A': 3, 'B': 4}
])
```

DataFrame Attributes

- Shape:

```python
df.shape  # (rows, columns)
```

- Columns:

```python
df.columns  # Column names
```

- Index:

```python
df.index  # Row indices
```

Tech

DataFrame Operations

Viewing Data

- Head and Tail:

```python
df.head()  # First 5 rows
df.tail()  # Last 5 rows
```

- Describe:

```python
df.describe()  # Summary statistics
```

Tech

- Info:

```python
df.info()  # Information about DataFrame
```

Selecting Data

- Single Column:

```python
df['A']  # Access column 'A'
```

- Multiple Columns:

```python
df[['A', 'B']]  # Access columns 'A' and 'B'
```

Tech

- Row by Label:

```python
df.loc[0]  # First row
```

- Row by Index:

```python
df.iloc[0]  # First row
```

Filtering Data

- Conditional Selection:

```python
df[df['A'] > 2]  # Rows where column 'A' is greater than 2
```

Techi

Data Manipulation

Adding Columns

- New Column:

```python
df['D'] = df['A'] + df['B']
```

Dropping Data

- Drop Rows:

```python
df.drop(0, axis=0)  # Drop first row
```

Techn

- Drop Columns:

```python
df.drop('D', axis=1)  # Drop column 'D'
```

Missing Data

- Detect Missing Values:

```python
df.isnull()  # Boolean DataFrame indicating missing values
```

- Drop Missing Values:

```python
df.dropna()  # Drop rows with any missing values
```

Techi

- Fill Missing Values:

```python
df.fillna(0)  # Replace missing values with 0
```

Grouping and Aggregating

Group By

- Group and Aggregate:

```python
df.groupby('A').sum()  # Group by column 'A' and sum
```

- Group and Apply:

```python
df.groupby('A').apply(lambda x: x.sum())  # Group by 'A' and apply
```

Merging and Joining

Merge

- Merge DataFrames:

```python
pd.merge(df1, df2, on='key')  # Merge df1 and df2 on column 'key'
```

Join

- Join DataFrames:

```python
df1.join(df2, lsuffix='_left', rsuffix='_right')  # Join on index
```

Input and Output

Reading Data

- CSV:

```python
df = pd.read_csv('file.csv')
```

- Excel:

```python
df = pd.read_excel('file.xlsx')
```

- SQL:

```python
import sqlite3
conn = sqlite3.connect('database.db')
df = pd.read_sql('SELECT * FROM table', conn)
```

Writing Data

- To CSV:

```python
df.to_csv('file.csv', index=False)
```

- To Excel:

```python
df.to_excel('file.xlsx', index=False)
```

Practical Examples

Time Series Data

- Creating Time Series DataFrame:

```python
dates = pd.date_range('20230101', periods=6)
df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list('ABCD'))
```

Pivot Table

- Creating Pivot Table:

```python
df.pivot_table(values='D', index='A', columns='B', aggfunc='sum')
```

Handling Duplicate Data

- Detect Duplicates:

```python
df.duplicated()
```

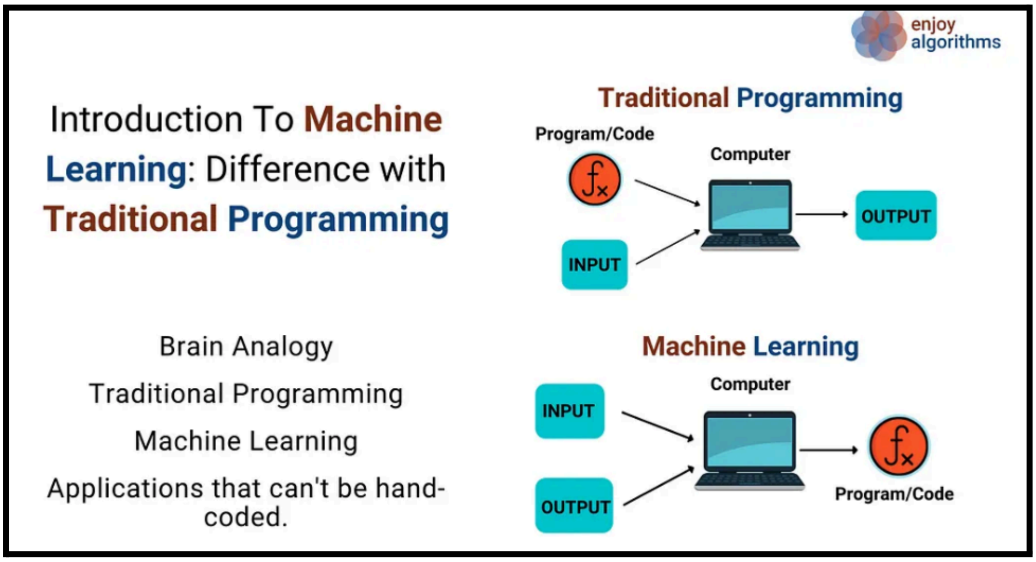
- Drop Duplicates:

```python
df.drop_duplicates()
```

Techne

INTRODUCTION TO ML

Machine learning (ML) is a branch of artificial intelligence (AI) that enables computers to “self-learn” from training data and improve over time, without being explicitly programmed. Machine learning algorithms are able to detect patterns in data and learn from them, in order to make their own predictions.



Classification of Machine Learning

Machine learning implementations are classified into four major categories, depending on the nature of the learning “signal” or “response” available to a learning system which are as follows:

A. Supervised learning:

Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. The given data is labeled. Both *classification* and *regression* problems are supervised learning problems.

- Example — Consider the following data regarding patients entering a clinic . The data consists of the gender and age of the patients and each patient is labeled as “healthy” or “sick”.

Supervised learning can be grouped further in two categories of algorithms:

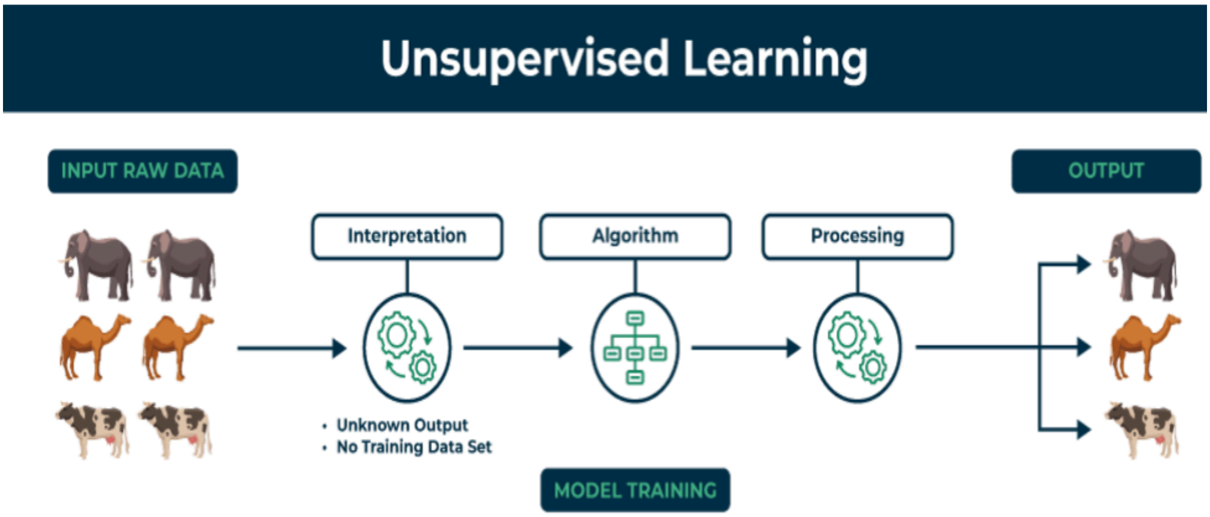
- **Classification**
- **Regression**

Gender	Age	Label
M	48	sick
M	67	sick
F	53	healthy
M	49	sick
F	32	healthy
M	34	healthy
M	21	healthy

B. Unsupervised Learning

In artificial intelligence, machine learning that takes place in the absence of human supervision is known as unsupervised machine learning. Unsupervised machine learning models, in contrast to supervised learning, are given unlabeled data and allow discover patterns and insights on their own—without explicit direction or instruction.

Unsupervised machine learning analyzes and clusters unlabeled datasets using machine learning algorithms. These algorithms find hidden patterns and data without any human intervention, i.e., we don't give output to our model. The training model has only input parameter values and discovers the groups or patterns on its own.



How does unsupervised learning work?

Unsupervised learning works by analyzing unlabeled data to identify patterns and relationships. The data is not labeled with any predefined categories or outcomes, so the algorithm must find these patterns and relationships on its own. This can be a challenging task, but it can also be very rewarding, as it can reveal insights into the data that would not be apparent from a labeled dataset.

Unsupervised Learning Algorithms

There are mainly 3 types of Algorithms which are used for Unsupervised dataset.

- Clustering
- Association Rule Learning
- Dimensionality Reduction

Clustering

Clustering in unsupervised machine learning is the process of grouping unlabeled data into clusters based on their similarities. The goal of clustering is to identify patterns and relationships in the data without any prior knowledge of the data's meaning.

Broadly this technique is applied to group data based on different patterns, such as similarities or differences, our machine model finds. These algorithms are used to process raw, unclassified data objects into groups. For example, in the above figure, we have not given output parameter values, so this technique will be used to group clients based on the input parameters provided by our data.

Some common clustering algorithms

- K-means Clustering: Partitioning Data into K Clusters
- Hierarchical Clustering: Building a Hierarchical Structure of Clusters
- Density-Based Clustering (DBSCAN): Identifying Clusters Based on Density
- Mean-Shift Clustering: Finding Clusters Based on Mode Seeking
- Spectral Clustering: Utilizing Spectral Graph Theory for Clustering

Association Rule Learning

Association rule learning is also known as association rule mining is a common technique used to discover associations in unsupervised machine learning. This technique is a rule-based ML technique that finds out some very useful relations between parameters of a large data set. This technique is basically used for market basket analysis that helps to better understand the relationship between different products. For e.g. shopping stores use algorithms based on this technique to find out the relationship between the sale of one product w.r.t to another's sales based on customer behavior. Like if a customer buys milk, then he may also buy bread, eggs, or butter. Once trained well, such models can be used to increase their sales by planning different offers.

- Apriori Algorithm: A Classic Method for Rule Induction
- FP-Growth Algorithm: An Efficient Alternative to Apriori
- Eclat Algorithm: Exploiting Closed Itemsets for Efficient Rule Mining
- Efficient Tree-based Algorithms: Handling Large Datasets with Scalability

Dimensionality Reduction

Dimensionality reduction is the process of reducing the number of features in a dataset while preserving as much information as possible. This technique is useful for improving the performance of machine learning algorithms and for data visualization. Examples of dimensionality reduction algorithms includeDimensionality reduction is the process of reducing the number of features in a dataset while preserving as much information as possible.

- Principal Component Analysis (PCA): Linear Transformation for Reduced Dimensions

- Linear Discriminant Analysis (LDA): Dimensionality Reduction for Discrimination
- Non-negative Matrix Factorization (NMF): Decomposing Data into Non-negative Components
- Locally Linear Embedding (LLE): Preserving Local Geometry in Reduced Dimensions
- Isomap: Capturing Global Relationships in Reduced Dimensions

Challenges of Unsupervised Learning

Here are the key challenges of unsupervised learning

- **Evaluation:** Assessing the performance of unsupervised learning algorithms is difficult without predefined labels or categories.
- **Interpretability:** Understanding the decision-making process of unsupervised learning models is often challenging.
- **Overfitting:** Unsupervised learning algorithms can overfit to the specific dataset used for training, limiting their ability to generalize to new data.
- **Data quality:** Unsupervised learning algorithms are sensitive to the quality of the input data. Noisy or incomplete data can lead to misleading or inaccurate results.
- **Computational complexity:** Some unsupervised learning algorithms, particularly those dealing with high-dimensional data or large datasets, can be computationally expensive.

Advantages of Unsupervised learning

- **No labeled data required:** Unlike supervised learning, unsupervised learning does not require labeled data, which can be expensive and time-consuming to collect.
- **Can uncover hidden patterns:** Unsupervised learning algorithms can identify patterns and relationships in data that may not be obvious to humans.
- **Can be used for a variety of tasks:** Unsupervised learning can be used for a variety of tasks, such as clustering, dimensionality reduction, and anomaly detection.
- **Can be used to explore new data:** Unsupervised learning can be used to explore new data and gain insights that may not be possible with other methods.

Disadvantages of Unsupervised learning

- **Difficult to evaluate:** It can be difficult to evaluate the performance of unsupervised learning algorithms, as there are no predefined labels or categories against which to compare results.
- **Can be difficult to interpret:** It can be difficult to understand the decision-making process of unsupervised learning models.
- **Can be sensitive to the quality of the data:** Unsupervised learning algorithms can be sensitive to the quality of the input

data. Noisy or incomplete data can lead to misleading or inaccurate results.

- **Can be computationally expensive:** Some unsupervised learning algorithms, particularly those dealing with high-dimensional data or large datasets, can be computationally expensive

Applications of Unsupervised learning

- **Customer segmentation:** Unsupervised learning can be used to segment customers into groups based on their demographics, behavior, or preferences. This can help businesses to better understand their customers and target them with more relevant marketing campaigns.
- **Fraud detection:** Unsupervised learning can be used to detect fraud in financial data by identifying transactions that deviate from the expected patterns. This can help to prevent fraud by flagging these transactions for further investigation.
- **Recommendation systems:** Unsupervised learning can be used to recommend items to users based on their past behavior or preferences. For example, a recommendation system might use unsupervised learning to identify users who have similar taste in movies, and then recommend movies that those users have enjoyed.

- **Natural language processing (NLP):** Unsupervised learning is used in a variety of NLP tasks, including topic modeling, document clustering, and part-of-speech tagging.
- **Image analysis:** Unsupervised learning is used in a variety of image analysis tasks, including image segmentation, object detection, and image pattern recognition.

Git and Github

Git is the free and open source distributed version control system that's responsible for everything GitHub related that happens locally on your computer. This cheat sheet features the most important and commonly used Git commands for easy reference.

Cheat Sheet:

https://education.github.com/git-cheat-sheet-education.pdf

References

Supervised Learning
Unsupervised Learning
Introduction to Machine learning
Machine Learning