

**TRIBHUVAN UNIVERSITY**

**Faculty of management**

**National College of Computer Studies**

Paknajol, Kathmandu

**PYTHON REPORT ON MOTION POSE CLASSIFICATION**

**Submitted by: Submitted To:**

Name: Jubisha Prajapati Mausam Rajbanshi

BIM 5th SEM

Section: B

Roll No:7

Submission Date

**Acknowledgement**

I would like to extend my heartfelt gratitude for the guidance and support provided by academic and professional mentor, ‘Mr. Mausam Rajbanshi’ for his constant support and guidance during making this report. His insightful feedback and valuable suggestions were crucial in shaping the structure and content of the report. I am also grateful to my friends whose valuable contributions and feedback helped me to enhance the quality of this report.

I would like to express my appreciation to NCCS college management for giving me this opportunity and providing me with the necessary resources and facilities to carry out this project. Their support was indispensable in ensuring the success of this project. Additionally, I want to thank the scikit-learn library's developers, contributors, and online resources that I have referred to while completing this project. Their contribution has been instrumental in providing us with the necessary information and data to complete this report.

Lastly, I would like to thank my family for their support and encouragement throughout the project process. Their encouragement and belief in my abilities have been a source of motivation and inspiration.

**Table of Contents**

[Introduction 1](#_Toc145173176)

[Machine Learning in python 2](#_Toc145173177)

[ Background 3](#_Toc145173178)

[Objectives 5](#_Toc145173179)

[Implementation 6](#_Toc145173180)

[Conclusion 15](#_Toc145173181)

# Introduction

In the ever-evolving landscape of computer vision and artificial intelligence, the accurate classification of human motion poses plays a pivotal role in numerous applications, ranging from gesture recognition and human-computer interaction to healthcare and sports analysis. Understanding and categorizing human motion allows machines to comprehend and respond to human actions, fostering innovation in areas such as robotics, gaming, and healthcare monitoring.

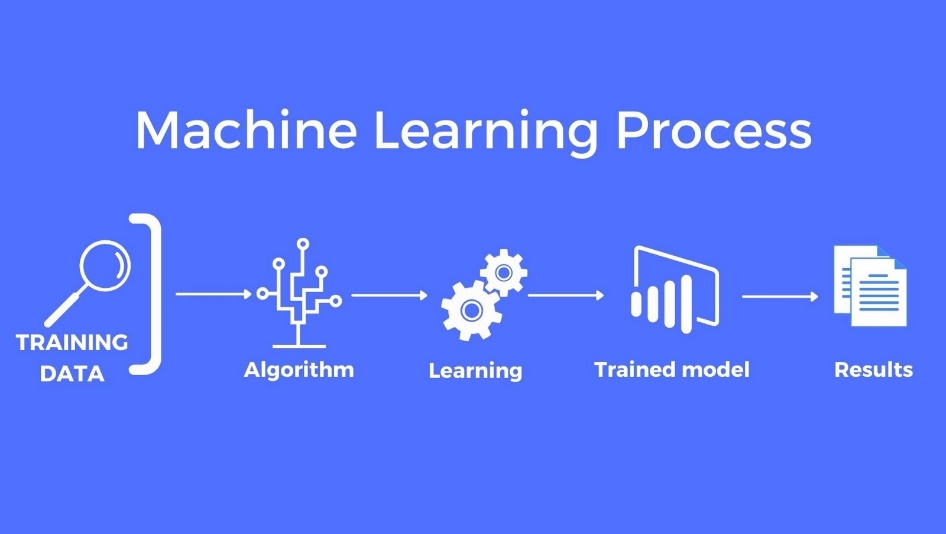
Motion pose classification is the process of recognizing and categorizing specific human body poses or movements from data, such as images, videos, or sensor readings. It is a fundamental task in computer vision and machine learning that involves identifying and assigning a label or class to a given human pose or motion based on a set of predefined categories. The primary goal of motion pose classification is to teach a computer system to understand and interpret human movements,

This report delves into the exciting field of motion pose classification, showcasing the utilization of Python as a powerful tool for developing sophisticated models and algorithms. We will explore the fundamental concepts, challenges, and techniques involved in classifying human motion poses, with a focus on how Python, coupled with popular libraries and frameworks, enables us to build robust and accurate motion pose classification systems.

# Machine learning in python

Machine learning in Python is a popular and powerful combination for developing and implementing machine learning models and algorithms. Python's simplicity, readability, extensive libraries, and vibrant community make it an ideal choice for working on machine learning projects. Python's rich library ecosystem, combined with its readability and ease of use, makes it an excellent choice for data scientists looking to effectively explore, analyze, and model data. Depending on the task, you can use a variety of libraries and tools to achieve your data science objectives.

Machine learning is a subset of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computer systems to improve their performance on a specific task through learning from data, without being explicitly programmed. In other words, machine learning allows computers to learn from experience or examples and make predictions or decisions based on that learned knowledge.



Here are some key concepts and components of machine learning:

1. **Data**: Machine learning heavily relies on data. It requires a substantial amount of data to train models effectively. This data can come in various forms, including text, images, numerical values, and more.
2. **Algorithms**: Machine learning algorithms are the mathematical and computational techniques used to discover patterns, relationships, and insights from data. These algorithms can be categorized into different types, including supervised learning, unsupervised learning, and reinforcement learning.
3. **Model Training**: To train a machine learning model, you provide it with a dataset and use an algorithm to adjust the model's parameters so that it can make accurate predictions or decisions. The training process involves optimizing the model's performance on the given task.
4. **Evaluation**: After training, it's crucial to assess the model's performance to ensure it can generalize well to unseen data. Common evaluation metrics depend on the specific task but may include accuracy, precision, recall, F1-score, mean squared error (MSE), and many others.
5. **Deployment**: Once a machine learning model has been trained and evaluated, it can be deployed in real-world applications to make predictions or automate decision-making. Deployment often involves integrating the model into software or systems.

## Background

Classification is one of the fundamental tasks in machine learning, and it refers to the process of categorizing or labelling data into distinct classes or categories based on certain features or attributes. Common algorithms for classification in Python include:

* Logistic Regression
* Decision Trees
* Random Forest
* Support Vector Machines (SVM)
* k-Nearest Neighbors (k-NN)
* Naive Bayes
* Neural Networks (Deep Learning)

Python provides various libraries and frameworks, such as scikit-learn, TensorFlow, and PyTorch, to implement classification algorithms effectively.

Data classification is a fundamental task in machine learning that involves training a model to classify data into predefined classes or categories. The motion pose dataset includes collection of data containing recorded information about human or object poses over time.

The primary goal of this data classification task is to construct a machine learning model capable of accurately classifying Iris flowers into their respective species based on these four attributes. To accomplish this, we use Python, a popular programming language for data analysis and machine learning, and a variety of libraries and techniques to handle the dataset, build a classification model, and evaluate its performance.

The following are the general steps involved in this data classification task:

* Data Collection: To begin, we obtain the motion pose dataset, which is easily accessible in Python via the scikit-learn library.
* Data Preparation: To facilitate data manipulation, the dataset is organized into a structured format, typically a Data Frame
* Algorithm Selection: We select an appropriate machine learning algorithm for this classification task. In this example, we use the decision tree algorithm and logistic regression algorithm, which is a straightforward yet effective method for classification tasks. The algorithm chosen is determined by the nature of the data and the requirements of the problem.
* Model Training: Using the training dataset, the selected algorithm is trained. The model learns to recognize patterns and relationships in the data during this phase, allowing it to make predictions.
* Model Evaluation: The model is evaluated using the testing dataset after it has been trained. To evaluate the model's performance, metrics such as accuracy, precision, recall, and the F1-score are calculated. A confusion matrix is also generated to visualize the model's classification results.
* Interpretation and Reporting: Model evaluation results are interpreted and reported on to determine the model's accuracy and effectiveness in classifying motion poses. These findings are then documented, most commonly in the form of a classification report and a confusion matrix.

This basic background gives an overview of the steps required to perform data classification on the motion pose dataset using Python. The process is invaluable for understanding the principles of data classification and model evaluation and serves as a foundational example for machine learning practitioners.

# Objectives

Here are some of the objectives of making this report: -

* To use Python to perform data classification on the motion pose dataset.
* To analyse and classify motion pose using machine learning algorithms.
* To identify the correlations between the parameters.
* To document the entire process of motion pose classification, from data collection and pre-processing to model development and evaluation.
* To share knowledge and insights gained during the project with a wider audience, including colleagues, researchers etc.
* To improve result visualization, visual aids such as a confusion matrix.
* To objectively evaluate the performance of the classification model and provide a clear assessment of its accuracy, precision, and other relevant metrics.
* To go over the reasoning behind data cleaning, dealing with missing values, and standardizing features.

# Implementation

To carry out the classification task, we use Python, a versatile programming language for data analysis and machine learning. The following steps outline the key implementation elements:

1. **Load Data:**

The motion pose dataset is loaded to begin the implementation. This dataset is easily accessible in Python via the scikit-learn library, which is a powerful machine learning tool. The four CSV files, x\_test , x\_train, y\_test and y\_train are loaded into Python environment.

1. **Prepare data**:

Preprocess the data as needed. This may include handling missing values, encoding categorical variables, and scaling or normalizing features.

1. **Merge or Concatenate Datasets**:

If the four CSV files represent different aspects of the same dataset or are meant to be combined, merge or concatenate them into a single data frame. For this purpose, x\_train and y\_train data have been concatenated and placed in one variable i.e. train\_data.

1. **Data Exploration and Analysis**:

The combined dataset is explored to understand its characteristics, distributions, and relationships between variables. Pandas are used for data exploration and libraries like Seaborn for data visualization.

1. **Split the Data**:

We divide the dataset into two subsets to facilitate model training and evaluation: a training set and a testing set. The training set is used to train your classification model, while the testing set is used to evaluate its performance. In this implementation, 80% of the data is allocated to the training set, with the remaining 20% used to test the model's performance. This division ensures that the model learns from a subset of the data and is tested on an independent dataset to determine its ability to generalize. **train\_test\_split()** function from scikit-learn is used for this purpose.

1. **Choose a Classification Model and Model Training:**

We select an appropriate machine learning classification model based on the nature of our data. Common choices include logistic regression, decision trees, random forests, and deep learning models like neural networks. Here, we have used logistic regression and decision tree.

1. **Model Evaluation**:

Here, we evaluate the performance of your classification model using appropriate evaluation metrics such as accuracy and confusion matrices. This helps you assess how well your model classifies data. We include visual aids such as a confusion matrix to improve the presentation of results and use heatmap to present the correlation in figure.

The codes used for this project are given below:

* **Necessary imports:**

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import accuracy\_score

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

* **Read the CSV file:**

x\_test = pd.read\_csv("./csv\_files/x\_test.csv")

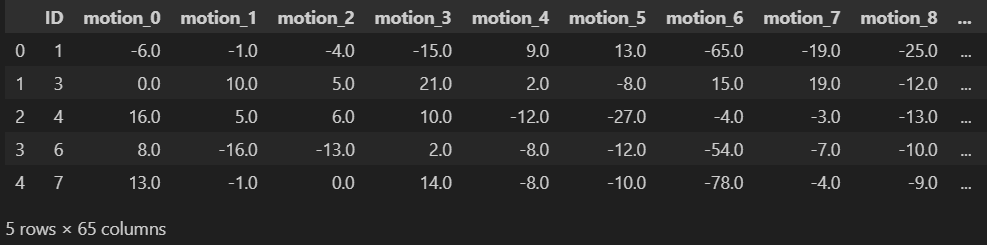
x\_test.head()

**A screen shot of a black screen

Description automatically generated**

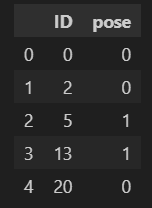
x\_train = pd.read\_csv("./csv\_files/x\_train.csv")

x\_train.head()



y\_test = pd.read\_csv("./csv\_files/y\_test.csv")

y\_test.head()



y\_train = pd.read\_csv("./csv\_files/y\_train.csv")

y\_train.head()

A screenshot of a black and white screen

Description automatically generated

* **Combining the data**

train\_data = x\_train.copy()

train\_data['pose'] = y\_train['pose']

train\_data

A screen shot of a black screen

Description automatically generated

* **Describing the data**

train\_data.describe()

A screen shot of a black screen

Description automatically generated

train\_data.info()

A screenshot of a computer

Description automatically generated

* **Check for duplicated and null values:**

train\_data.isnull().sum()

A screen shot of a computer

Description automatically generated

train\_data.duplicated().sum()

A black screen with a white number

Description automatically generated

* **Correlation matrix**

train\_data.corr()

A screen shot of a computer

Description automatically generated

import seaborn as sns

sns.heatmap(train\_data.corr(), cmap='Blues')

A graph of a line

Description automatically generated with medium confidence

* **Model Training**

A black background with white text

Description automatically generated

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, train\_size= 0.20, random\_state=0)

* **Logistic Regression**

from sklearn.linear\_model import LogisticRegression

classify\_model = LogisticRegression(max\_iter=1000)

classify\_model.fit(X\_train, y\_train)

classify\_model

A close-up of a computer screen

Description automatically generated

preds = classify\_model.predict(X\_test)

preds



* **Confusion matrix**

from sklearn.metrics import confusion\_matrix

confusion\_matrix(y\_test,preds)

A black background with white numbers

Description automatically generated

accuracy\_score(y\_test,preds)



* **Decision Tree**

from sklearn.tree import DecisionTreeClassifier

dt\_classifier = DecisionTreeClassifier()

dt\_classifier.fit(X\_train, y\_train)

dt\_classifier.predict(X\_test)



preds\_dt = dt\_classifier.predict(X\_test)

print(preds\_dt)

print(confusion\_matrix(y\_test, preds\_dt))

A number on a black background

Description automatically generated

print(accuracy\_score(y\_test, preds\_dt))



# Conclusion

In conclusion, I embarked on a journey of data classification using Python in this report, focusing on the well-known motion pose dataset. This report has explored the exciting and versatile realm of motion pose estimation using Python. We have delved into various aspects of this field, including the importance of motion pose analysis, the techniques and algorithms commonly employed.

Motion pose estimation plays a crucial role in a wide range of applications, from sports analytics to healthcare and robotics. It enables us to extract valuable insights from motion data, enhance human-computer interactions, and improve the efficiency and safety of various processes.

Python, with its extensive libraries and frameworks, offers a powerful platform for developing motion pose estimation solutions. Throughout this report, we have discussed key concepts such as data preprocessing, model training etc. Additionally, we have highlighted the importance of data quality and model selection in achieving accurate results. The proposed deep learning model was implemented using the Python programming language with packages and libraries. The motion pose dataset was used to perform basic data analysis, then data standardization and visualization were performed. Finally, the model was trained for the accuracy and performance of the model were also evaluated.

As technology continues to advance, motion pose estimation will only become more relevant and impactful. In conclusion, I gained valuable insights from this project and came to know that motion pose estimation using Python is a dynamic and evolving field with a broad range of applications, and it holds the promise of transforming how we interact with and understand the world around us.