

**BÁO CÁO TỔNG KẾT**

Classify Yoga Pose base on Keypoints

**GVHD:** TS. Mai Tiến Dũng.

**Thông tin thành viên:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Họ và tên** | **MSSV** | **Công việc thực hiện** | **Mức độ hoàn thành** |
| Trần Kim Ngọc Ngân | 22520002 | * Chuẩn bị nội dung. * Chuẩn bị slide thuyết trình. * Thực nghiệm với KNN. | 100% |
| Nguyễn Duy Thắng | 22521333 | * Chuẩn bị nội dung. * Thực hiện rút trích đặc trưng Keypoints. * Thực nghiệm với SVM. | 100% |

# **1, Lý do của bài toán**

Trong môi trường hiện đại, con người có xu hướng ngồi làm việc hàng giờ liền. Điều đó dẫn đến hệ quả là nhiều người sẽ dễ cảm thấy mệt mỏi, yếu đuối. Chính vì lý do đó, chúng ta thường được khuyên bảo rằng phải tập luyện thể dục thể thao nhiều hơn. Việc này nghe có vẻ dễ nhưng cũng không hẳn, vì chúng ta rất dễ dàng chán nản và cảm thấy mệt mỏi hơn sau khi tập luyện, nhất là đối với những người mới bắt đầu, những người không có động lực tập luyện mạnh mẽ.

Thường thì khi mới bắt đầu, để không cảm thấy nhanh chán nản, chúng ta cần ưu tiên tập bộ môn mà mình cảm thấy yêu thích. Mỗi một bộ môn thể dục đều có các ưu nhược điểm riêng, nhưng phù hợp với nhiều người nhất thì có lẽ sẽ là Yoga - một bộ môn không quá khó, không yêu cầu quá nhiều đối với người mới bắt đầu mà vẫn mang lại hiệu quả cao. Yoga mang lại rất nhiều lợi ích cho sức khoẻ như: Cải thiện sức khỏe; giảm căng thẳng, stress; tăng cường sự tập trung; Nâng cao sự linh hoạt của cơ thể;…

Tuy nhiên, việc tập yoga cũng có những khó khăn nhất định cho một số người mới bắt đầu. Một trong số chúng là chúng ta thường không biết chính xác các tư thế nếu như không có sự chỉ dẫn và có thể nhầm lẫn giữa các động tác với nhau - nhất là đối với các động tác hơi phức tạp. Và trên thực tế, sẽ có một số người không muốn tốn quá nhiều thời gian mà chỉ cần một số lợi ích của việc tập yoga nên sẽ lựa chọn tập luyện tại nhà. Vì vậy, một mô hình dùng để phân loại các động tác yoga mà người dùng đang thực hiện sẽ khá cần thiết để họ biết được rằng mình đã tập luyện đúng động tác mong muốn hay không.

# **2, Phát biểu bài toán**

**Mô tả bài toán:** Yoga là một môn thể dục có rất nhiều lợi ích và phù hợp với nhiều người. Tuy nhiên, đối với một số người chưa quen, việc nhầm lẫn giữa các động tác là một việc thường xuyên xảy ra nên cần có một thứ gì đó giúp cho họ phân loại được động tác họ đang thực hiện. Đặc biệt, đây là việc phân loại tư thế nên việc rút trích đặc trưng của input sẽ khác so với việc rút trích đặc trưng trong bài toán phân loại khác. Chúng ta sẽ lấy vị trí của các khớp trên cơ thể người để làm đặc trưng phân loại. Sau đây sẽ là một số đặc điểm chính của bài toán:

## **2.1, Input**

Ảnh số một người đang thực hiện một động tác Yoga và Tập dữ liệu đã được gán nhãn (tên của các động tác yoga).

## **2.2, Output**

Nhãn của động tác yoga trong ảnh số đầu vào.

# **3, Cách thức thực hiện**

## **Preview, MediaPipe - BlazePose**

MediaPipe Pose Landmarker cho phép phát hiện các điểm mốc (pose landmarks) của cơ thể con người trong hình ảnh hoặc video. Có thể sử dụng tác vụ này để xác định các vị trí chính của cơ thể, phân tích tư thế và phân loại chuyển động.

**Nguồn gốc:** Về tổng quan, [MediaPipe](https://github.com/google/mediapipe) là tập hợp của một loạt các giải pháp Machine Learning đa nền tảng của Google, có thể can thiệp được và cực kỳ lightweight.

Câu hỏi là làm sao để detect ra được các keypoints trên cơ thể người? Giải pháp của họ dựa trên một paper mà họ đề xuất, keyword: BlazePose. Dựa trên giải pháp này, họ sẽ bóc tách ra 33 điểm tương ứng 33 bộ phận của cơ thể người.

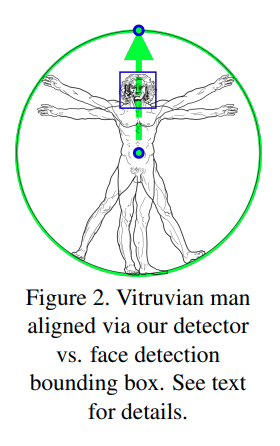
BlazePose bao gồm hai thành phần chính là:

* Pose Detector: Để detect ra vùng chứa person trên bức ảnh.
* Pose Tracker: Để trích xuất ra các keypoints trên vùng chứa vị trí của person đã được crop ra từ bức ảnh đồng thời dự đoán vị trí của person trong next frame.

**Về Pose Detector:**

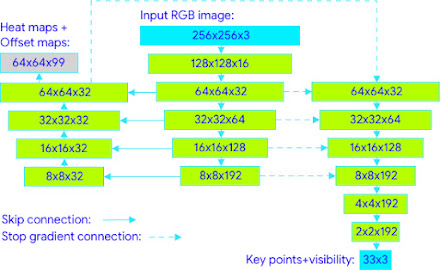
Trong bài báo thay vì detect toàn thân thì tác giả sử dụng bộ detect khuôn mặt cùng với điểm giữa hông của người hiện tại, kích thước của vòng tròn bao quanh toàn bộ người, góc nghiêng (góc được đo bởi đường thẳng nối bởi điểm giữa hông - điểm giữa vai với phương thẳng đứng). Đây là một bộ detector đơn giản và light-weight.

Từ các thông tin detect được thì detector tiến hành alignment để xoay person theo phương thẳng đứng.



**Về Pose Tracker:**

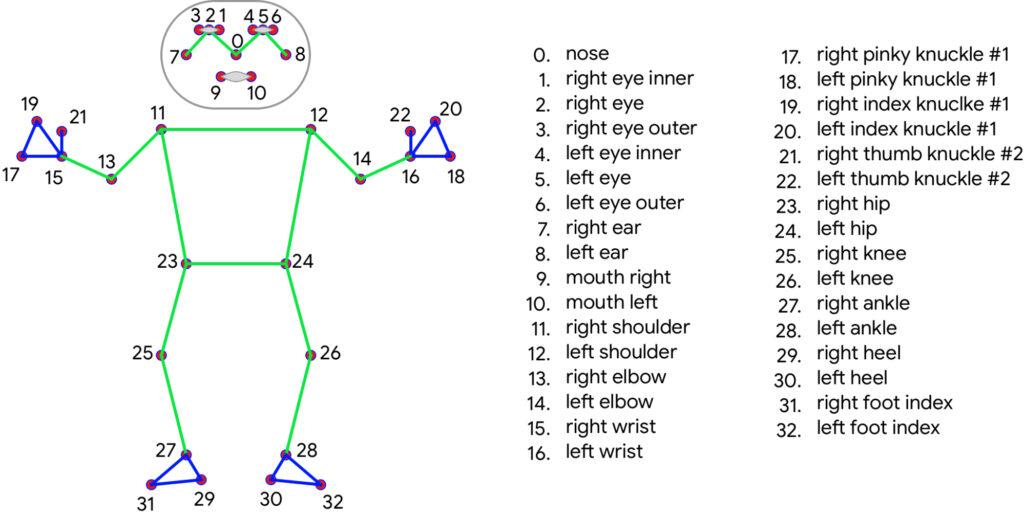
Mạng Pose Tracking được chia làm hai phần đó là Keypoints Detection Part và Keypoints Regression Part như hình bên dưới:



Đầu tiên tác giả training mạng bên trái và ở giữa trước. Sau đó phần regression bên phải sẽ được training bằng việc share feature với mạng. Khi testing, tác giả sẽ bỏ đi hoàn toàn detection part và chỉ giữ lại phần regression. Đầu ra của mạng này sẽ bao gồm 33 keypoints và 2 điểm sử dụng để alignment được mô tả phía trên phần Pose Detector.

**Cách áp dụng:**

* Sử dụng thư viện mediapipe của Python để detect ra 33 điểm keypoints đặc trưng.
* Lấy các đặc trưng toạ độ x, toạ độ y, toạ độ z, vị trí keypoints trên ảnh làm các đặc trưng để phân loại.
* Làm tương tự với 17 điểm keypoints là các điểm của các đoạn thẳng màu xanh lá theo hình bên dưới.
* Sử dụng các toạ độ này như vector đặc trưng dùng cho việc phân loại.

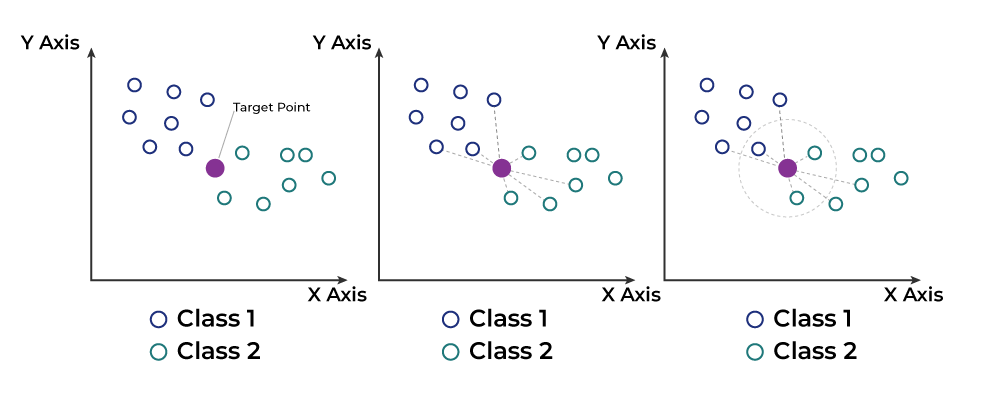


**3.1, KNN - k-nearest neighbor**

**Nguồn gốc:** Thuật toán K-Nearest Neighbors (KNN) được phát triển trong bối cảnh của những nỗ lực đầu tiên trong lĩnh vực học máy và trí tuệ nhân tạo. Nó xuất hiện lần đầu vào những năm 1950 và 1960 như một phần của nghiên cứu về mô hình hóa mẫu và nhận dạng. Mặc dù không rõ ai là người chính thức tạo ra KNN, thuật toán này đã nhanh chóng trở nên phổ biến do tính đơn giản và dễ hiểu của nó.

K-nearest neighbor là một trong những thuật toán supervised-learning đơn giản nhất (mà hiệu quả trong một vài trường hợp) trong Machine Learning. Khi training, thuật toán này không học một điều gì từ dữ liệu training (đây cũng là lý do thuật toán này được xếp vào loại [lazy learning](https://en.wikipedia.org/wiki/Lazy_learning)), mọi tính toán được thực hiện khi nó cần dự đoán kết quả của dữ liệu mới. Lớp (nhãn) của một đối tượng dữ liệu mới có thể dự đoán từ các lớp (nhãn) của k hàng xóm gần nó nhất.

**Cách hoạt động:** Cơ chế hoạt động của KNN dựa trên nguyên tắc rằng các điểm dữ liệu tương tự nhau thường nằm gần nhau. Để dự đoán nhãn cho một điểm dữ liệu mới, KNN xem xét ‘K’ điểm dữ liệu gần nhất (hay là hàng xóm gần nhất) và dựa trên đa số nhãn của chúng để quyết định nhãn cho điểm dữ liệu mới đó.



Giá trị ‘K’ là một siêu tham số quan trọng – nó quy định số lượng hàng xóm gần nhất mà thuật toán sẽ xem xét. Lựa chọn K phù hợp là cực kỳ quan trọng: một giá trị K quá nhỏ có thể dẫn đến mô hình bị ảnh hưởng bởi nhiễu dữ liệu, trong khi một giá trị K quá lớn có thể làm mô hình không nhạy với các đặc điểm cụ thể của dữ liệu.

**Cách áp dụng:**

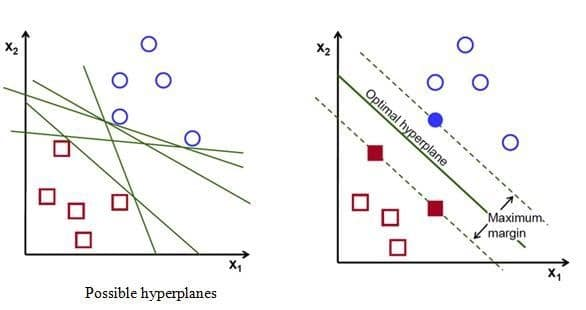
* Sử dụng KNN với các số k khác nhau từ 1 đến 48 (số lượng class + 1).
* Sử dụng tập train để huấn luyện và tập validation để đánh giá.
* Chọn ra model có accuracy cao nhất đem đi đánh giá trên tập test và đánh giá thực tế.

## **3.2, SVM - support vector machine**

**Nguồn gốc:**

Support Vector Machines là một trong số những thuật toán phổ biến và được sử dụng nhiều nhất trong học máy trước khi mạng [nơ ron](https://trituenhantao.io/tu-dien-thuat-ngu/no-ron/) nhân tạo trở lại với các mô hình [deep learning](https://trituenhantao.io/tu-dien-thuat-ngu/deep-learning/). Nó được biết đến rộng rãi ngay từ khi mới được phát triển vào những năm 1990. Thuật toán SVM ban đầu được tìm ra bởi [Vladimir N. Vapnik](https://vi.wikipedia.org/w/index.php?title=Vladimir_N._Vapnik&action=edit&redlink=1) và dạng chuẩn hiện nay sử dụng Biên mềm được tìm ra bởi Vapnik và [Corinna Cortes](https://vi.wikipedia.org/w/index.php?title=Corinna_Cortes&action=edit&redlink=1) năm 1995.

Mục tiêu của [SVM](https://trituenhantao.io/tu-dien-thuat-ngu/svm/) là tìm ra một [siêu phẳng](https://vi.wikipedia.org/wiki/Si%C3%AAu_ph%E1%BA%B3ng) trong không gian N chiều (ứng với N [đặc trưng](https://trituenhantao.io/tu-dien-thuat-ngu/dac-trung/)) chia dữ liệu thành hai phần tương ứng với lớp của chúng. Để phân chia hai lớp dữ liệu, rõ ràng là có rất nhiều siêu phẳng có thể làm được điều này. Mặc dù vậy, mục tiêu của chúng ta là tìm ra siêu phẳng có lề rộng nhất tức là có khoảng cách tới các điểm của hai lớp là lớn nhất.



**Cách hoạt động:**

* **Đối với dữ liệu có thể phân chia tuyến tính:**

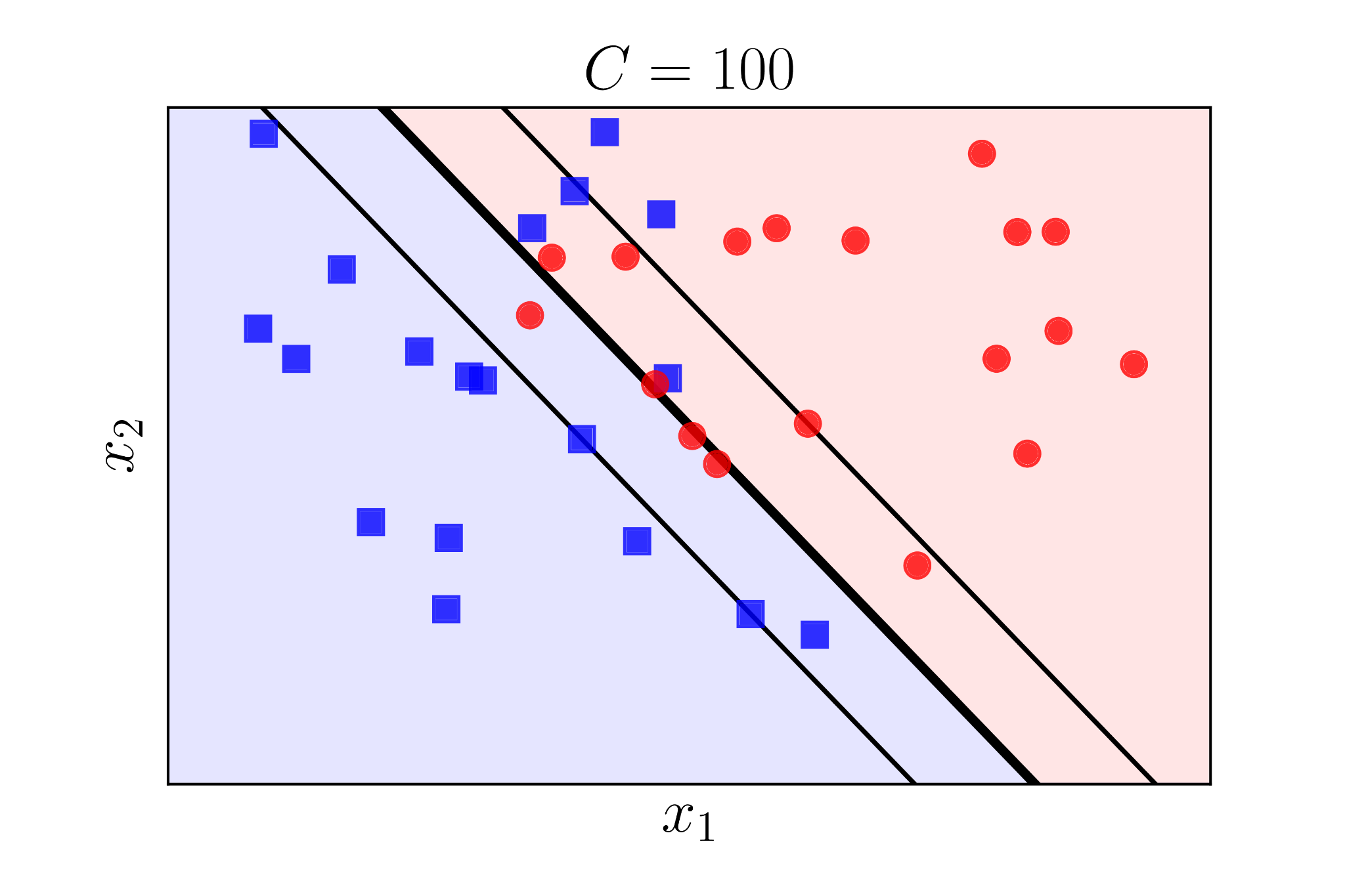
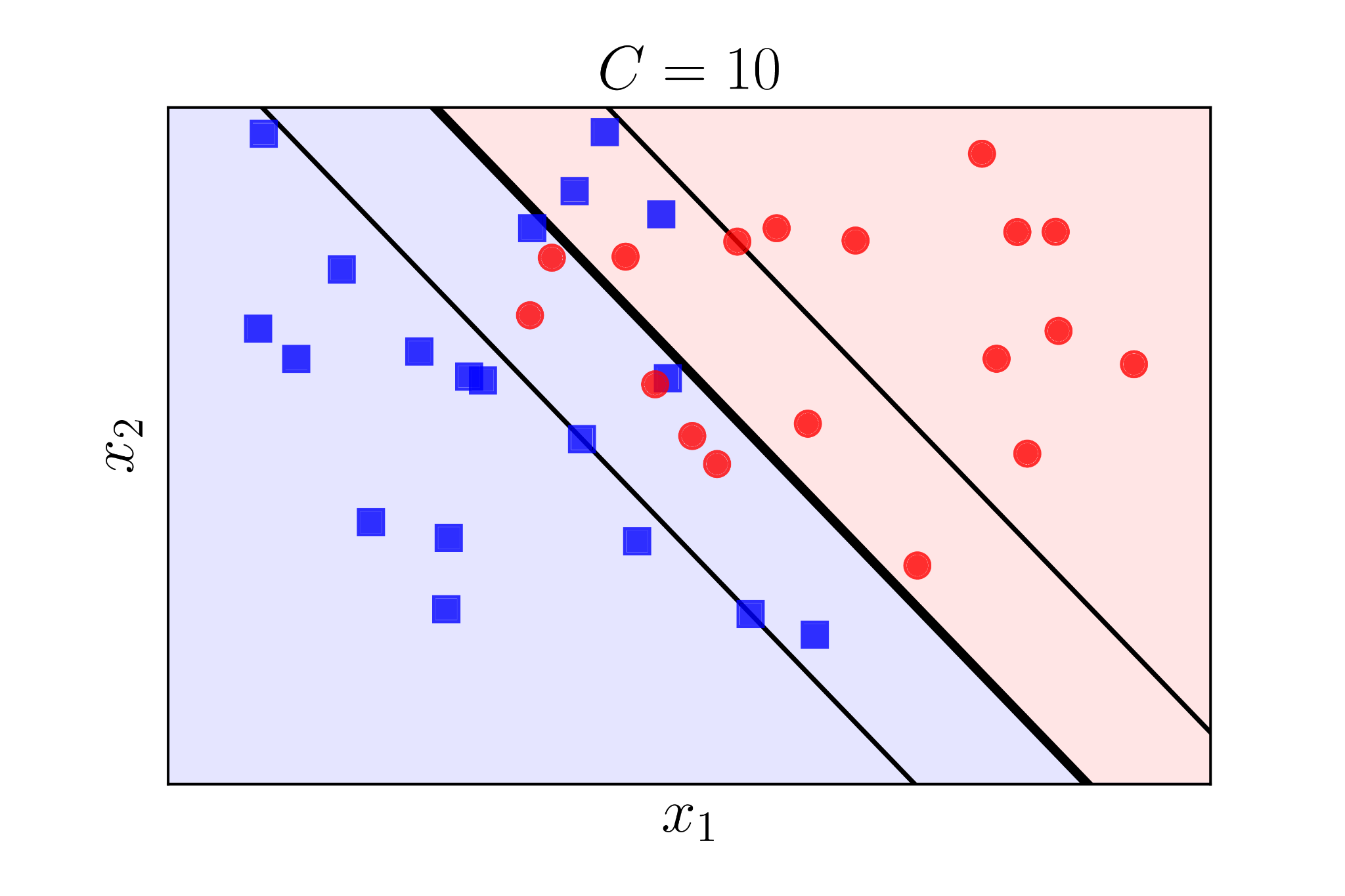
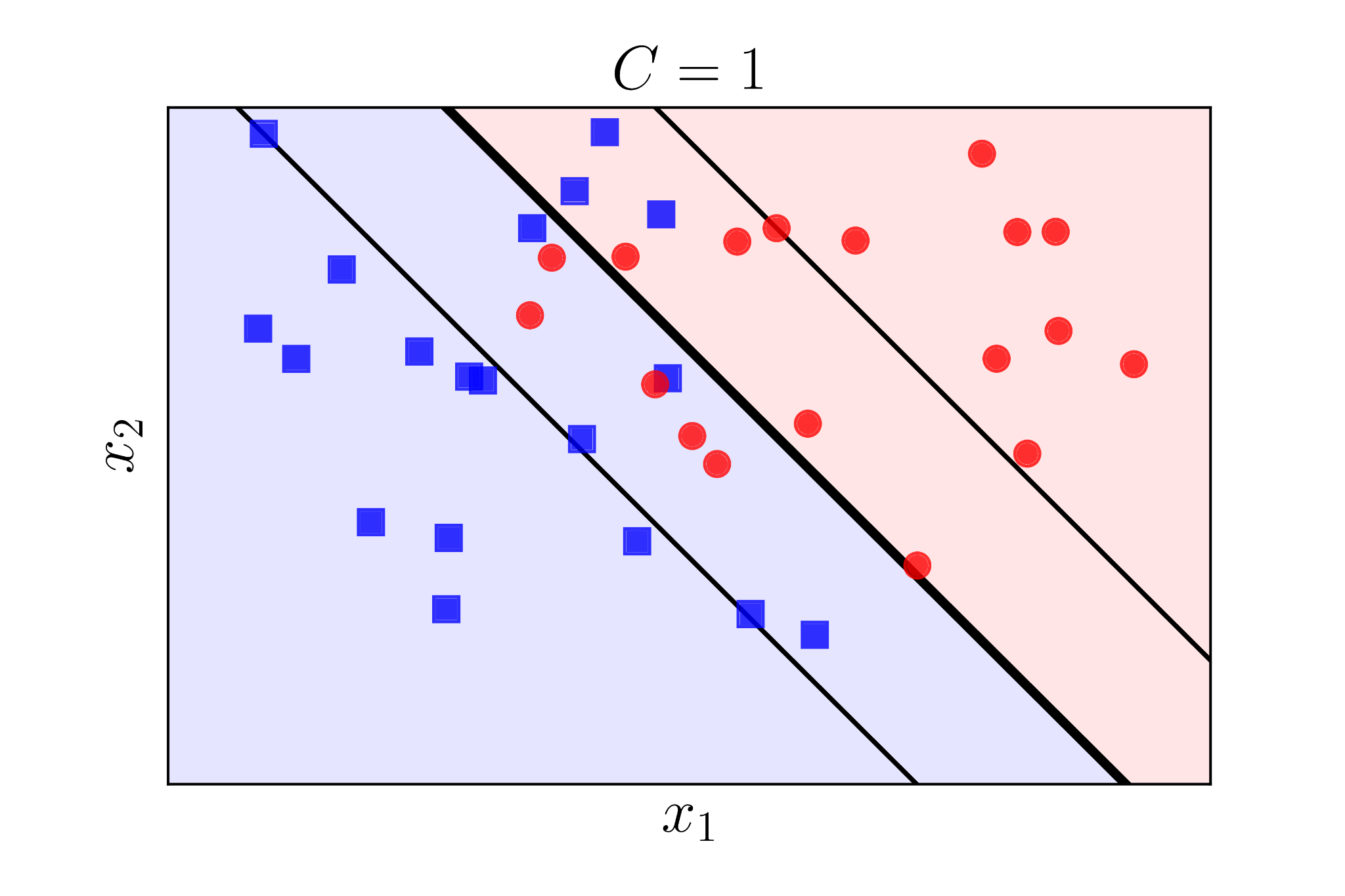
Bài toán tối ưu trong Support Vector Machine (SVM) chính là bài toán đi tìm đường phân chia sao cho margin là lớn nhất. Việc giải bài toán trên liên quan đến một số lý thuyết toán học tương đối phức tạp như điều kiện Karush-Kuhn-Tucker, hàm đối ngẫu Lagrange, Convex optimization.

Có thể coi chi tiết tại: <https://phamdinhkhanh.github.io/deepai-book/ch_ml/SVM.html>

* **Đối với dữ liệu không thể phân chia tuyến tính:** Có 2 cách để xử lý.
  + **Cách 1: Soft Margin:**

Thuật toán SVM có các đặc điểm để bỏ qua phần ngoại lệ và tìm ra siêu phẳng tốt nhất giúp tối đa hóa lề. Thuật toán này cho phép SVM mắc một số lỗi nhất định và giữ cho lề càng rộng càng tốt để các điểm khác vẫn có thể được phân loại chính xác. Nói một cách khác, nó cân bằng giữa việc phân loại sai và tối đa hóa lề.

Mức độ chấp nhận lỗi là một siêu tham số quan trọng trong SVM. Khi lập trình với sklearn, mức độ chấp nhận lỗi được coi như một tham số phạt (C). C càng lớn có nghĩa là SVM càng bị phạt nặng khi thực hiện phân loại sai. Do đó, lề càng hẹp.



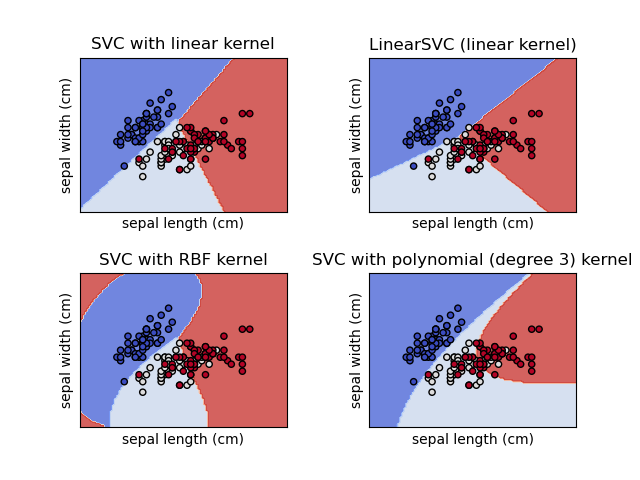
* **Cách 2: Kernel**

Một kernel là một hàm ánh xạ dữ liệu từ không gian ít nhiều hơn sang không gian nhiều chiều hơn, từ đó ta tìm được siêu phẳng phân tách dữ liệu. Một cách trực quan, kỹ thuật này giống như việc bạn gập tờ giấy lại để có thể dùng kéo cắt một lỗ tròn trên nó. Ví dụ:



Các kiểu Kernel:

* Tuyến tính (Linear).
* Đa thức (Polynomial).
* RBF.

****

**Cách áp dụng:**

* Sử dụng tập train, val để tìm hiểu các loại bậc trong kernel polydegree để tìm ra các bậc tốt nhất.
* Tương tự, nhưng áp dụng để tìm tham số C thích hợp cho từng kernel với từng loại đặc trưng.
* Huấn luyện mô hình trên tập train và đánh giá trên tập validation.
* Chọn ra model có accuracy cao nhất trong 3 để đem đi đánh giá trên tập test và đánh giá thực tế.

# **4, Thực nghiệm**

## **4.1, Độ đo / Cách đánh giá**

**Accuracy:** Khi xây dựng một mô hình Machine Learning, chúng ta cần một phép đánh giá để xem mô hình sử dụng có hiệu quả không và để so sánh khả năng của các mô hình.

Cách đơn giản và hay được sử dụng nhất là *accuracy* (độ chính xác). Cách đánh giá này đơn giản tính tỉ lệ giữa số điểm được dự đoán đúng và tổng số điểm trong tập dữ liệu kiểm thử.

## **4.2, Dataset**

Source: <https://www.kaggle.com/datasets/tr1gg3rtrash/yoga-posture-dataset>

* Dataset sẽ được tạo lại dựa trên Source, các ảnh đều được detect lại cho giống input và lật ngang để augmentation.
* Dataset gồm 47 class chưa được chia train-test với 4374 file ảnh số.
* Các ảnh cùng nhãn sẽ được đặt vào cùng 1 folder mang nhãn đó.
* Dataset được chia train-test theo tỉ lệ 7:3. Và tập validation sẽ được lấy ra từ tập train cũng theo tỉ lệ 7:3. Số lượng chi tiết như sau:

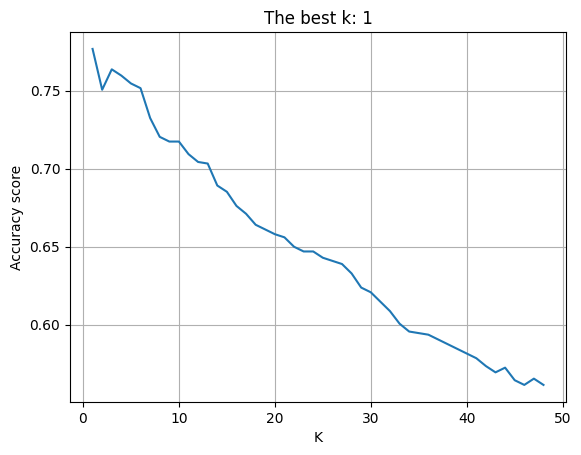
|  |  |  |
| --- | --- | --- |
| Train | Val | Test |
| 2319 | 994 | 1421 |

## **4.3, Kết quả thực nghiệm**

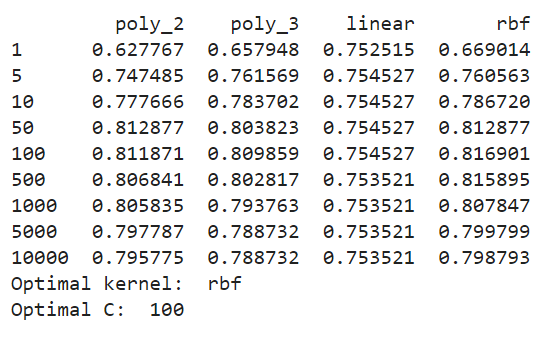
### **4.3.1: feature\_points\_2D.csv**

#### **Train**

KNN:



SVM:



#### **Test**

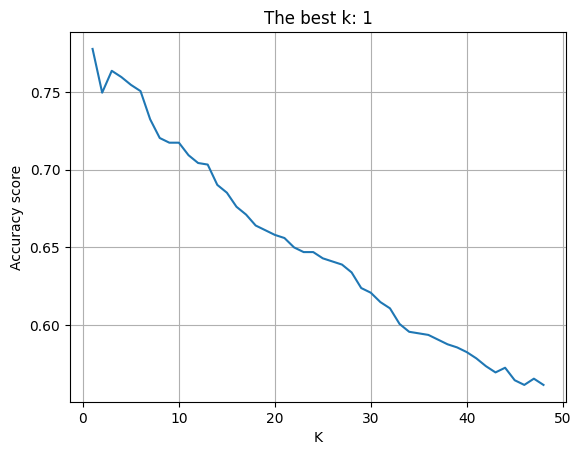
KNN: 0.8191414496833216

SVM: 0.844475721323012

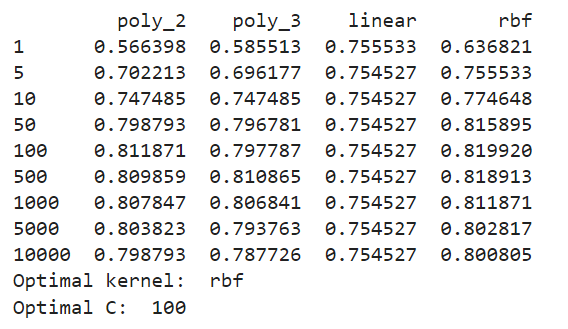
### **4.3.2: feature\_points\_3D.csv**

#### **Train**

KNN:



SVM:



#### **Test**

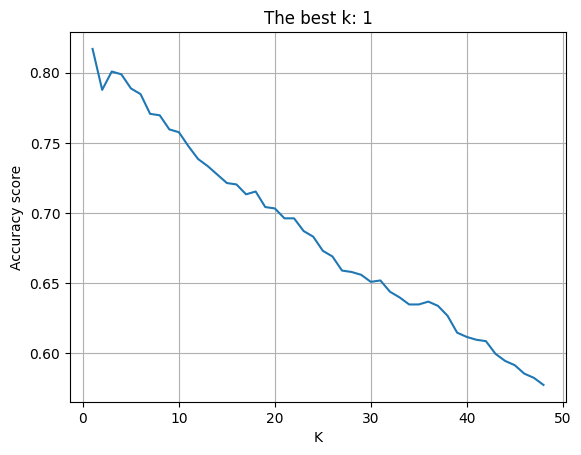
KNN: 0.8198451794510908

SVM: 0.8423645320197044

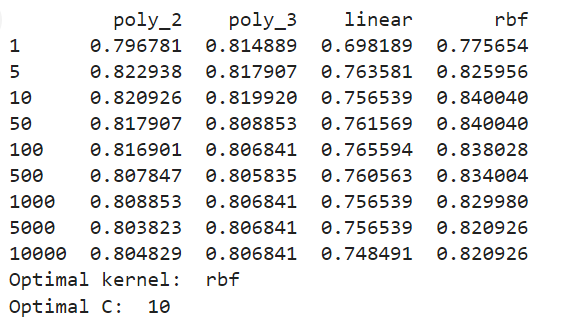
### **4.3.3: feature\_2D.csv**

#### **Train**

KNN:



SVM:



#### **Test**

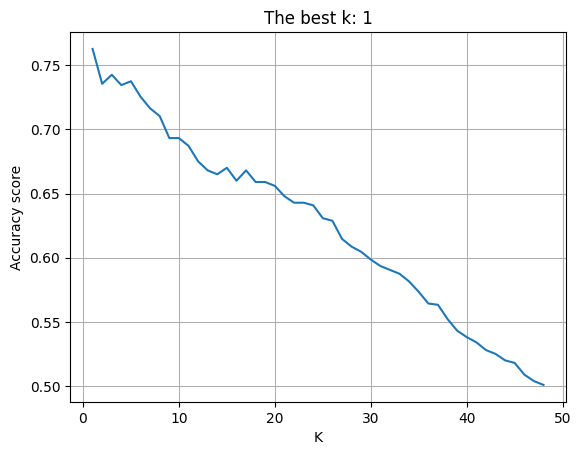
KNN: 0.844475721323012

SVM: 0.8740323715693173

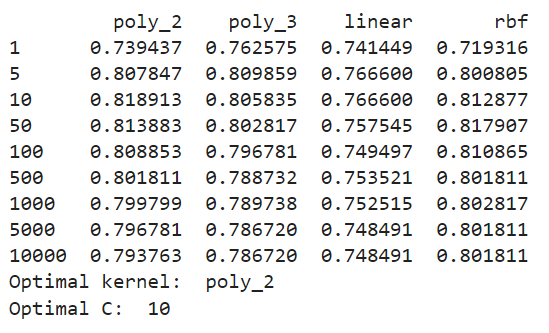
### **4.3.4: feature\_3D.csv**

#### **Train**

KNN:



SVM:



#### **Test**

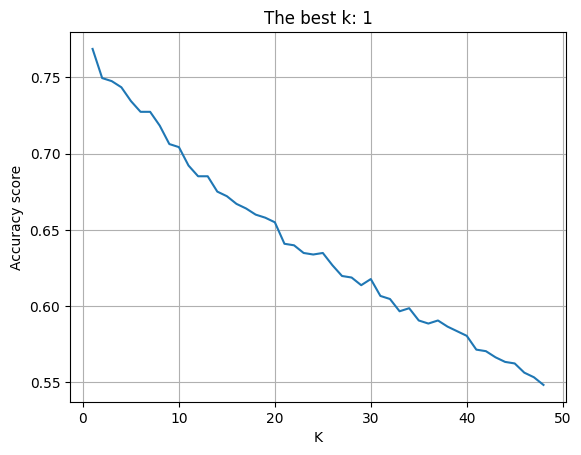
KNN: 0.7994370161857847

SVM: 0.82899366643209

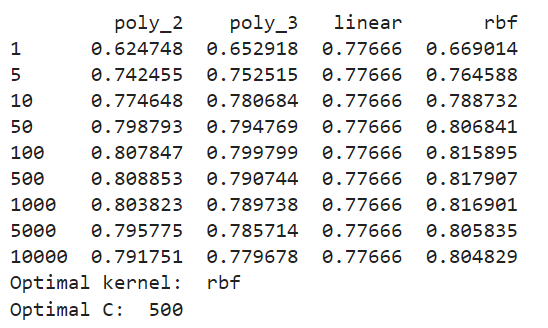
### **4.3.5: feature\_all\_points\_2D.csv**

#### **Train**

KNN:



SVM:



#### **Test**

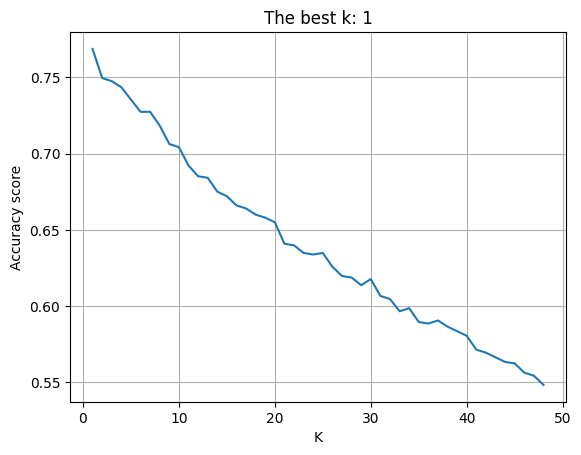
KNN: 0.8071780436312456

SVM: 0.8353272343420126

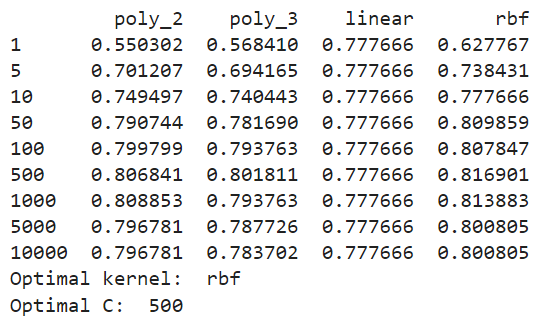
### **4.3.6: feature\_all\_points\_3D.csv**

#### **Train**

KNN:



SVM:



#### **Test**

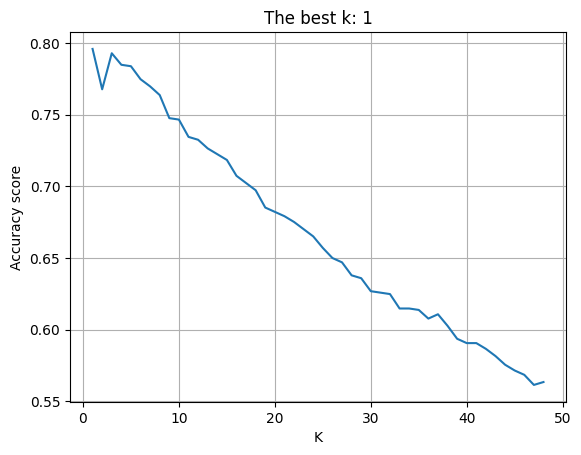
KNN: 0.8064743138634765

SVM: 0.8395496129486277

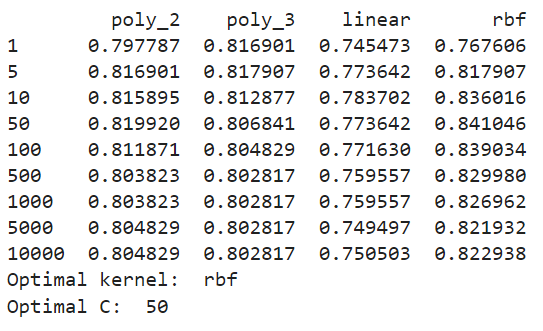
### **4.3.7: feature\_all\_2D.csv**

#### **Train**

KNN:



SVM:



#### **Test**

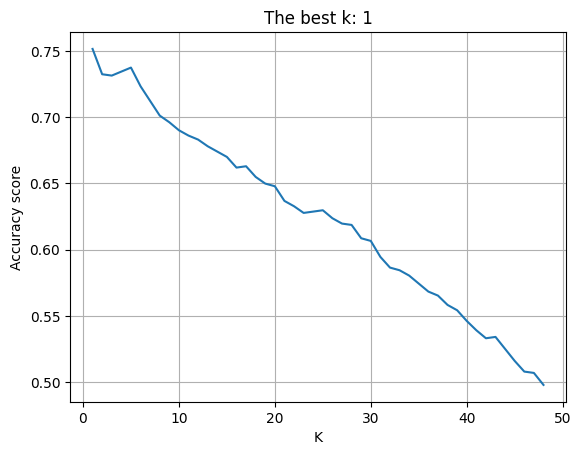
KNN: 0.8184377199155525

SVM: 0.8550316678395496

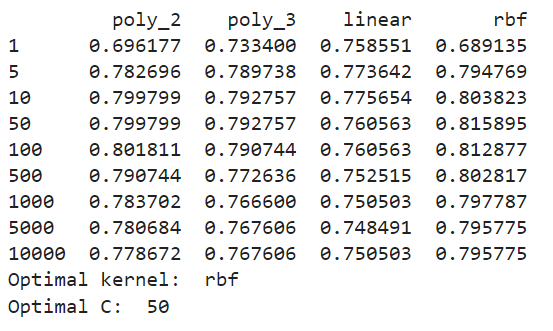
### **4.3.8: feature\_all\_3D.csv**

#### **Train**

KNN:



SVM:



#### **Test**

KNN: 0.7762139338494018

SVM: 0.8325123152709359

# **5, Đánh giá, nhận xét.**

**Ưu điểm:**

* .
* .

**Hạn chế:**

* .
* .

# **6, Tài liệu tham khảo.**

*1.4. Support vector machines*. (n.d.). Scikit-Learn. Retrieved June 6, 2024, from https://scikit-learn.org/stable/modules/svm.html

*7.1. Hàm mất mát của SVM — Deep AI KhanhBlog*. (n.d.). Github.Io. Retrieved June 6, 2024, from https://phamdinhkhanh.github.io/deepai-book/ch\_ml/SVM.html

aswathisasidharan Follow Improve. (2021, January 20). *Support vector machine (SVM) algorithm*. GeeksforGeeks. https://www.geeksforgeeks.org/support-vector-machine-algorithm/

Bazarevsky, V., Grishchenko, I., Raveendran, K., Zhu, T., Zhang, F., & Grundmann, M. (2020). BlazePose: On-device Real-time Body Pose tracking. In *arXiv [cs.CV]*. http://arxiv.org/abs/2006.10204

Hoang, P. M. (2021, April 16). *Tản mạn một chút về Pose Classification*. Viblo. https://viblo.asia/p/tan-man-mot-chut-ve-pose-classification-yMnKM07g57P

Improve, G. (2017, April 14). *K-nearest neighbor(KNN) algorithm*. GeeksforGeeks. https://www.geeksforgeeks.org/k-nearest-neighbours/

Io T. (2020, January 5). *SVM quá khó hiểu! Hãy đọc bài này*. Trí tuệ nhân tạo; trituenhantao.io. https://trituenhantao.io/kien-thuc/svm-qua-kho-hieu-hay-doc-bai-nay/

*KNeighborsClassifier*. (n.d.). Scikit-Learn. Retrieved June 6, 2024, from https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Melo, C. (2023, September 11). Real-time Human Pose Estimation using MediaPipe. *Sigmoidal*. https://sigmoidal.ai/en/real-time-human-pose-estimation-using-mediapipe/

Phạm Văn, T. (2021, March 31). *[B5’] - Tìm hiểu về BlazePose: On-device Real-time Body Pose tracking*. Viblo; Sun\* AI Research Team. https://viblo.asia/p/b5-tim-hieu-ve-blazepose-on-device-real-time-body-pose-tracking-XL6lADpJZek

*Pose landmark detection guide*. (n.d.). Google for Developers. Retrieved June 6, 2024, from https://ai.google.dev/edge/mediapipe/solutions/vision/pose\_landmarker

Renotte, N. (n.d.). *MediaPipePoseEstimation*.

Trung, H. C. (2020, August 20). *Giới thiệu về Support Vector Machine (SVM)*. Viblo. https://viblo.asia/p/gioi-thieu-ve-support-vector-machine-svm-6J3ZgPVElmB

Vu, T. (2017a, January 8). *Bài 6: K-nearest neighbors*. Tiep Vu’s Blog. https://machinelearningcoban.com/2017/01/08/knn/

Vu, T. (2017b, April 9). *Bài 19: Support Vector Machine*. Tiep Vu’s Blog. https://machinelearningcoban.com/2017/04/09/smv/

Vu, T. (2017c, April 13). *Bài 20: Soft margin support vector machine*. Tiep Vu’s Blog. https://machinelearningcoban.com/2017/04/13/softmarginsmv/

Vu, T. (2017d, April 22). *Bài 21: Kernel Support Vector Machine*. Tiep Vu’s Blog. https://machinelearningcoban.com/2017/04/22/kernelsmv/

Vu, T. (2017e, April 28). *Bài 22: Multi-class Support Vector Machine*. Tiep Vu’s Blog. https://machinelearningcoban.com/2017/04/28/multiclasssmv/