ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN CUỐI KỲ

MÔN: LẬP TRÌNH PYTHON CHO MÁY HỌC

KERNEL SVM

Sinh viên thực hiện:

Hoàng Tiến Dũng - 19521388

Nguyễn Thành Trọng - 19522410

Giảng viên hướng dẫn: **TS. Nguyễn Vinh Tiệp**

Lớp: **CS116.M11**

NĂM 2021

LỜI CẢM ƠN

Sau quá trình học tập và rèn luyện tại trường Đại học Công nghệ Thông tin chúng em đã được trang bị nhiều kiến thức bổ ích và các kỹ năng cần có để hoàn thành đồ án của môn học của mình.

Chúng em xin chân thành gửi lời cảm ơn tới thầy Nguyễn Vinh Tiệp đã hướng dẫn, truyền đạt kiến thức và kinh nghiệm cho chúng em trong suốt thời gian học tập môn Lập trình Python cho Máy học.

Trong quá trình là đồ án môn không tránh khỏi được những sai sót, chúng em mong nhận được sự góp ý của quý thầy và các bạn để được hoàn thiện hơn.

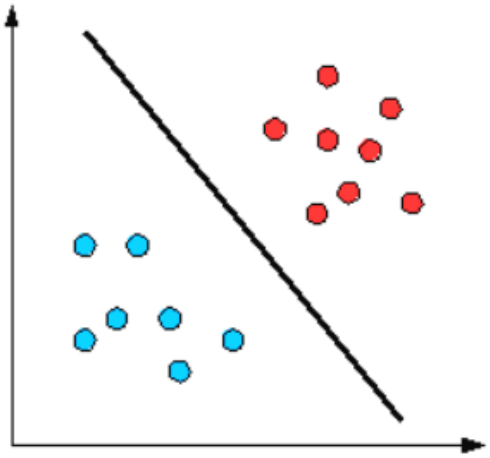
TP.Hồ Chí Minh, tháng 12 năm 2021

MỤC LỤC

I. Giới thiệu

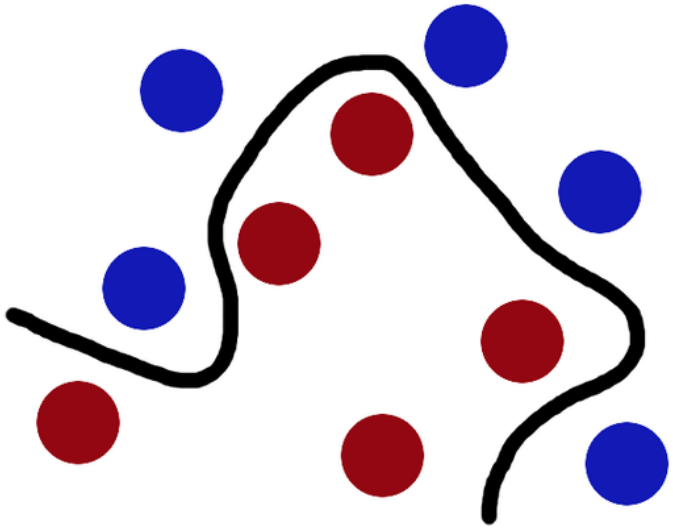
Support Vector Machine (SVM) là một thuật toán thuộc nhóm Supervised Learning (Học có giám sát) dùng để phân chia dữ liệu (Classification) thành các nhóm riêng biệt.

Giả sử ta có bộ data gồm các điểm xanh và đỏ đặt trên cùng mặt phẳng. Ta có thể tìm được đường thẳng để phân chia riêng biệt các bộ điểm xanh và đỏ như hình bên dưới. Vậy với những bộ dữ liệu phức tạp hơn mà không thể tìm được đường thẳng để phân chia thì giải quyết như thế nào?



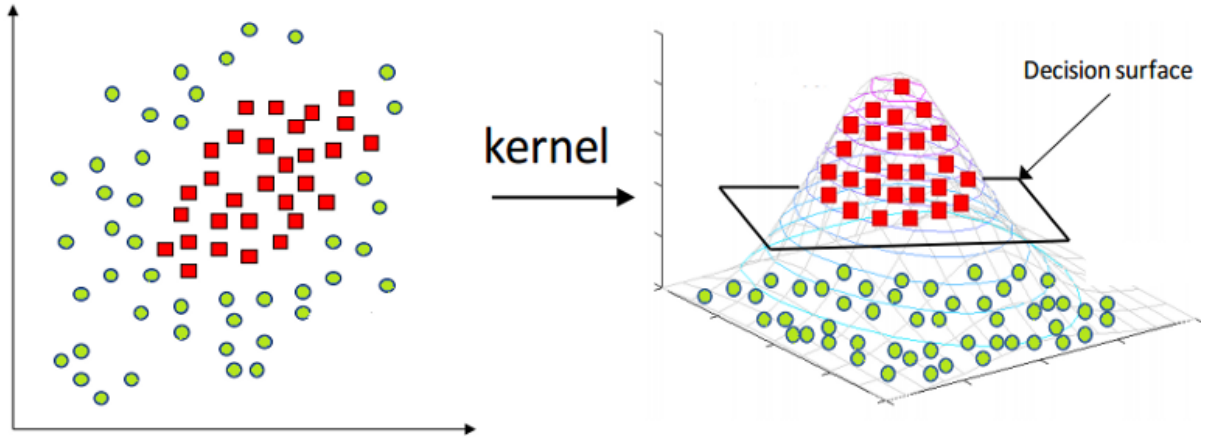
***Hình 1**: Phân chia dữ liệu với SVM*

Như đã biết, bộ dữ liệu từ thực tế hầu hết các trường hợp đều không ở dạng tuyến tính, nếu áp dụng SVM với những bộ dữ liệu như vậy thông thường sẽ vô nghiệm. Với những trường hợp này, ta sẽ cần đến những phép biến đổi dữ liệu đầu vào để có thể dùng SVM. Phương pháp thường được dùng trong SVM với dữ liệu không tuyến tính là Kernel SVM.



***Hình 2**: Mô tả dữ liệu từ thực tế*

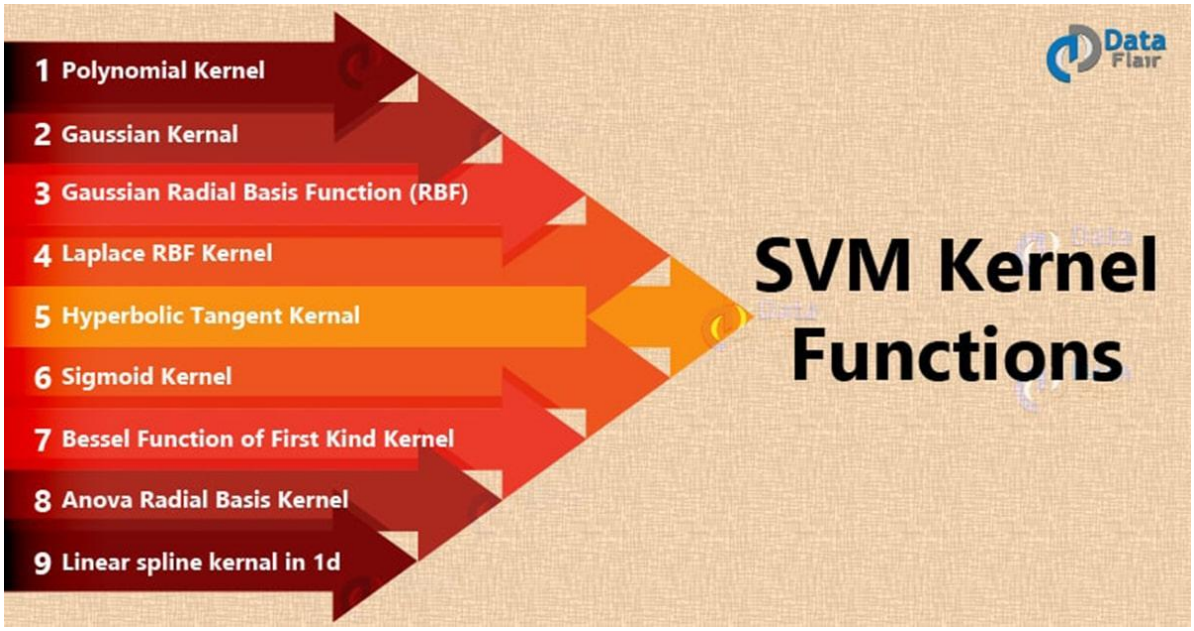
Nói ngắn gọn, Kernel SVM là việc đi tìm một hàm số biến đổi dữ liệu X từ không gian feature ban đầu thành dữ liệu trong một chiều không gian mới bằng hàm số $\phi(x)$. Mục đích là trong chiều không gian mới này, dữ liệu của các phân lớp trong không gian mới sẽ phân biệt tuyến tính hoặc gần như phân biệt tuyến tính. Khi đó, ta có thể áp dụng SVM cho bài toán ban đầu trên không gian dữ liệu mới để tìm ra siêu mặt phẳng (hyperplane/decision surface) phân tách dữ liệu vào các phân lớp.



***Hình 3**: Biến đổi dữ liệu không tuyến tính sang không gian với số chiều lớn hơn*

Tuy nhiên, hàm $\phi(x)$ thường tạo ra số chiều cao hơn số chiều của dữ liệu ban đầu. Nếu tính toán các hàm này trực tiếp sẽ gặp phải các vấn đề về bộ nhớ và hiệu năng tính toán. Do đó, có một cách tiếp cần là sử dụng các hàm kernel để mô tả quan hệ giữa hai điểm dữ liệu trong không gian mới, thay vì tính toán trực tiếp từng điểm.

Có thể kể tới một số hàm kernel phổ biến như Linear Kernel, Polynomial Kernel, Gaussian radial basis function kernel (RBF), Gaussian Kernel, Laplace RBF Kernel, Sigmoid Kernel,....



***Hình 4**: Các hàm kernel phổ biến*

II. Cơ sở toán học

1. Tiếp cận cách Kernel hoạt động

Để tiếp cận cách Kernel hoạt động một cách dễ hiểu, ta hãy xét bài toán sau:

Cho $K(x, y) = < f(x), f(y) >$. K là một hàm kernel. x, y là vectơ n chiều, f là một hàm ánh xạ vectơ n chiều tới không gian m chiều. $< x, y >$ là tích vô hướng của x và y $(m > n)$. Thông thường, để tính $< f(x), f(y) >$ yêu cầu chúng ta phải tính $f(x), f(y)$ trước, sau đó sẽ tính tích vô hướng. Có thể thấy cách tính toán này có độ phức tạp lớn vì m có thể là một số rất lớn. Tuy nhiên, ta nhận thấy sau khi tính toán ở không gian m chiều, việc nhân tích vô hướng hai vectơ sẽ đưa chúng ta quay lại không gian một chiều. Vậy câu hỏi được đặt ra ở đây là: liệu chúng ta có cần phải đi tới không gian m chiều không? Câu trả lời là KHÔNG nếu chúng ta tìm được một kernel phù hợp.

Giả sử: $x = (x_1, x_2, x_3)$; $y = (y_1, y_2, y_3)$. Cho hàm $f(x) = (x_1x_1, x_1x_2, x_1x_3$, $x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3)$ thì hàm kernel phù hợp sẽ là $K(x, y) = (< x, y >)^2$.

Để trực quan, xét các vectơ cụ thể:

Với $x = (1, 2, 3), y = (4, 5, 6)$. *Lúc này*

$$f(x) = (1, 2, 3, 2, 4, 6, 3, 6, 9)$$

$$f(y) = (16, 20, 24, 20, 25, 30, 24, 30, 36)$$

$$K(x, y) = < f(x), f(y) > = 16 + 40 + 72 + 40 + 100 + 180 + 72 + 180 + 324 = 1024$$

→Rất phức tạp! Lý do là f là hàm ánh xạ từ không gian 3 chiều sang không gian 9 chiều.

Với cách giải dùng hàm kernel phù hợp:

$$K(x, y) = (< x, y >)^2 = (4 + 10 + 18)^2 = 32^2 = 1024$$

→ Cách tính dễ dàng và nhỏ gọn.

2. Cơ sở toán học

Ta có bài toán đối ngẫu trong Soft Margin SVM cho dữ liệu gần phân biệt tuyến tính:

$$\lambda = argmax \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1, m=1}^{N} \lambda_n \lambda_m y_n y_m x_n^T x_m$$

thỏa mãn: $\sum_{n=1}^{N} \lambda_n y_n = 0$ *và* $0 \le \lambda_n \le C, \forall n = 1, 2, ..., N$

Trong đó:

N là số lần huấn luyện

x_n là vectơ đặc trưng thứ n

$y_n = \pm 1$ là nhãn dữ liệu thứ n

λ_n là nhân từ Largrange ứng với điểm dữ liệu thứ n

C là một hằng số dương giúp cân đối độ lớn giữa độ rộng lề và sự hi sinh của các điểm nằm trong vùng không an toàn.

Sau khi tìm được λ cho bài toán, nhãn mới của dữ liệu sẽ được xác định bởi:

$$class(x) = sgn\left\{\sum_{m\in S} \lambda_m y_m x_m^T + \frac{1}{N_M}\sum_{n\in S}\left(y_n - \sum_{m\in S}\lambda_m y_m x_m^T x_n\right)\right\}$$

Trong không gian mới:

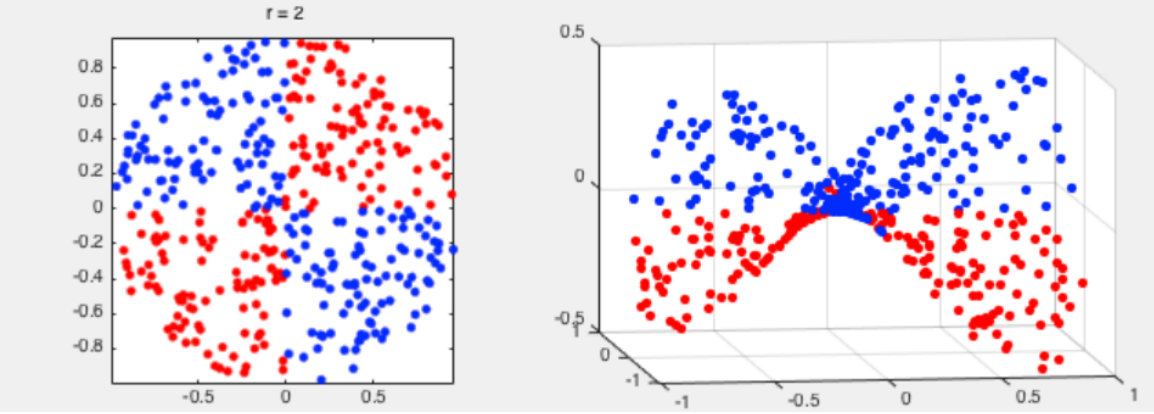
$$\lambda = argmax \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1,m=1}^{N} \lambda_n \lambda_m y_n y_m \varphi(x_n)\ \varphi(x_m) \quad (2.2.1)$$

thỏa mãn: $\sum_{n=1}^{N} \lambda_n y_n = 0$ và $0 \le \lambda_n \le C,\ \forall n = 1, 2, \ldots, N$

Nhãn của một điểm dữ liệu mới được xác định bởi dấu của biểu thức:

$$\omega^T \varphi(x) + b = \sum_{m\in S} \lambda_m y_m \varphi(x_m)^T \varphi(x) + \frac{1}{N_M}\sum_{n\in S}\left(y_n - \sum_{m\in S}\lambda_m y_m x_m^T x_n\right) \quad (2.2.2)$$

Trong bài toán 2.2.1 và biểu thức 2.2.2 chúng ta không cần tính trực tiếp $\varphi(x)$ cho mọi điểm dữ liệu. Chúng ta chỉ cần tính được $\varphi(x)^T \varphi(z)$ dựa trên hai điểm dữ liệu $x,\ z$ bất kỳ!



***Hình 5**: Kernel trick trong SVM*

Kỹ thuật này còn được gọi là "Kernel trick". Những phương pháp dựa trên kỹ thuật này, tức thay vì tính trực tiếp tọa độ của một điểm trong không gian mới, ta đi tính tích vô hướng giữa hai điểm trong không gian mới, được gọi chung là "Kernel method". Ta định nghĩa hàm $k(x,\ z) = \varphi(x)^T \varphi(z)$, biểu thức 2.2.1 và 2.2.2 có thể viết lại công thức:

$$\lambda = argmax \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1, m=1}^{N} \lambda_n \lambda_m y_n y_m k(x_n, x_m)$$

thỏa mãn: $\sum_{n=1}^{N} \lambda_n y_n = 0$ và $0 \leq \lambda_n \leq C, \forall n = 1, 2, \ldots, N$

và $\sum_{m \in S} \lambda_m y_m k(x_m, x) + \frac{1}{N_M} \sum_{n \in S} (y_n - \sum_{m \in S} \lambda_m y_m k(x_m, x))$

III.Hàm số hạt nhân

1. Tính chất của hàm số hạt nhân

a. Đối xứng:

$$k(x, z) = k(z, x)$$

Điều này để nhận ra vì tích vô hướng của hai vectơ có tính đối xứng.

b. Thỏa mãn điều kiện Mercer

$$\sum_{n=1}^{k} \sum_{m=1}^{k} k(x_m, x_n) c_m c_n \geq 0, \forall c_i \in R, i = 1, 2, \ldots, N \textbf{ (1)}$$

Nếu 1 hàm kernel thỏa mãn điều kiện (1), xét $c_n = y_n \lambda_n$, ta sẽ có:

$$\lambda^T K \lambda = \sum_{n=1}^{k} \sum_{m=1}^{k} k(x_m, x_n) y_n y_m \lambda_n \lambda_m \geq 0, \forall \lambda_n \textbf{ (2)}$$

Với *K* là một ma trận đối xứng với phần tử hàng thứ n, cột thứ m được định nghĩa bởi:

$$K_{nm} = k(x_m, x_n) y_n y_m$$

Từ **(2)** suy ra *K* là ma trận bán xác định dương

2. Một số hàm hạt nhân thông dụng

a. Linear kernel (Linear)

Đây là loại kernel cơ bản nhất, thường là một chiều. Nó được chứng minh là chức năng tốt nhất khi có rất nhiều tính năng. Linear kernel chủ yếu được ưu tiên cho các bài toán phân loại văn bản. Vì hầu hết các bài toán phân loại này có thể được phân tách một cách tuyến tính.

$$\phi(x_1, x_2) = x_1^T x_2$$

Trong đó:

x_1, x_2: là đại diện cho dữ liệu đang phân loại

b. Polynomial kernel (Poly)

Polynomial kernel là một biểu diễn tổng quát hơn của hạt nhân tuyến tính(linear kernel). Nó không được ưa thích như các hàm nhân khác vì nó kém hiệu quả và chính xác hơn.

$$\phi(x_1, x_2) = (\gamma {x_1}^T x_2 + r)^d$$

Trong đó:

d: số bậc của đa thức

r: hệ số tự do

γ: hệ số tỷ lệ của dữ liệu đầu vào

c. Radial basis function kernel (RBF)

Là một trong những hàm kernel được ưu tiên và sử dụng nhiều nhất trong SVM. Nó thường được chọn cho dữ liệu phi tuyến tính. Nó giúp thực hiện phân tách thích hợp khi ta không biết gì về dữ liệu.

$$\phi(x_1, x_2) = exp(-\gamma ||x_1 - x_2||_2^2)$$

Trong đó:

γ: hệ số tỷ lệ của dữ liệu đầu vào

d. Sigmoid kernel (Sigmoid)

Các kernel phải được thỏa mãn định lý Mercer, và điều đó yêu cầu kernel xác định dương. Tuy nhiên, Sigmoid kernel, mặc dù được sử dụng rộng rãi nhưng nó không phải là bán xác định dương đối với một số lựa chọn giá trị tham số của nó. Do đó, khi áp dụng Sigmoid kernel, các tham số σ, r phải được chọn phù hợp. Nếu không có thể SVM hoạt động kém hơn ngẫu nhiên.

$$\phi(x_1, x_2) = tanh(\gamma {x_1}^T x_2 + r)$$

Trong đó:

γ : hệ số tỷ lệ của dữ liệu đầu vào

r : hệ số tự do

e. Custom kernel

Ngoài các kernels phổ biến được nêu trên, chúng ta còn có thể tự định nghĩa kernel cho mình và thử nghiệm trên các bộ dữ liệu cá nhân để so sánh kết quả với các mô hình sử dụng các kernel trên.

3. So sánh các hàm kernel

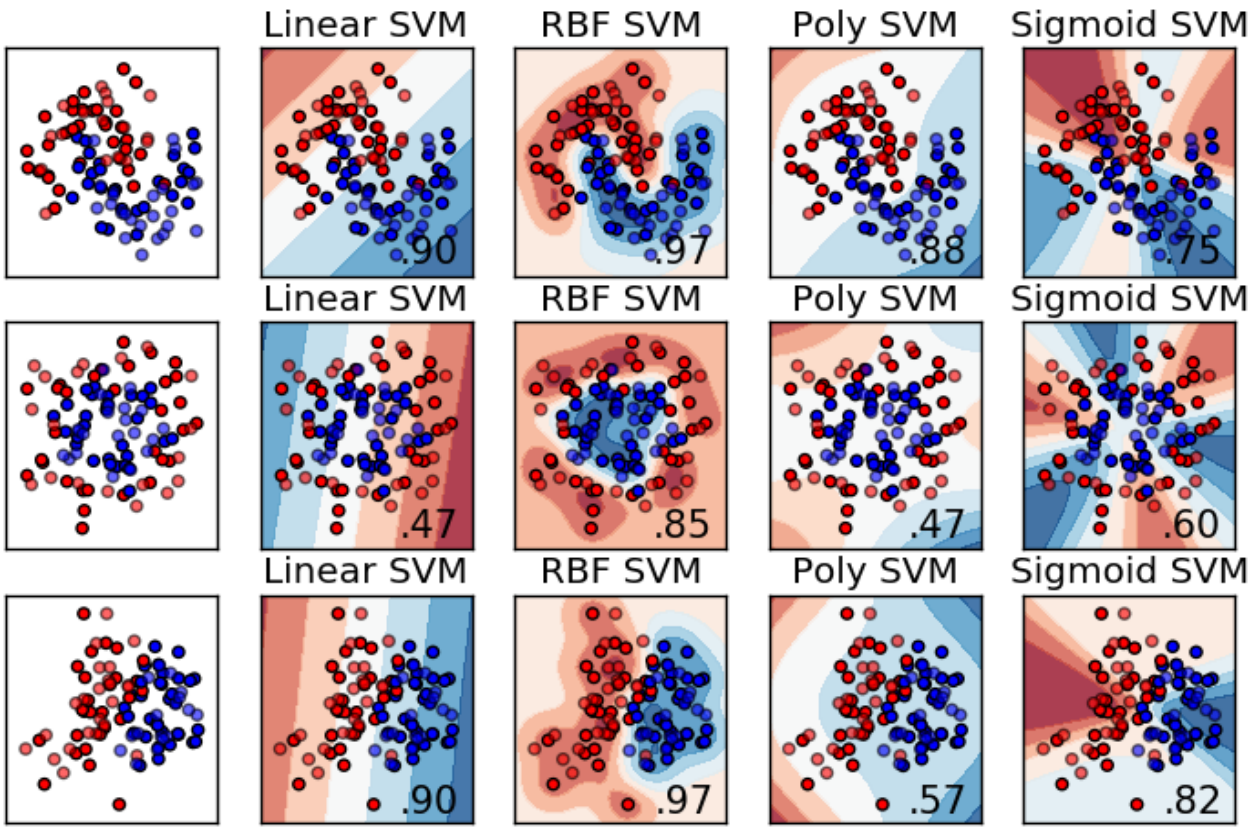
	Linear	Poly	RBF	Sigmoid
Cơ sở	Đây là tích vô hướng của hai véc tơ.	Là một biểu diễn tổng quát hơn của linear kernel. Tạo ra một đa thức bậc cao kết hợp giữa hai véc tơ.	Dựa trên hàm Gaussian RBF. Hàm biến đổi phi tuyến của kernel này là hàm ẩn và tương đương với một đa thức với	Dựa trên kernel về đa thức, đưa chuyển tiếp qua hàm tanh. Hàm tanh có thể biểu diễn theo hàm sigmoid.

			bậc vô hạn.	
Độ phổ biến	Linear kernel thường được sử dụng cho phân loại văn bản hoạt động tốt trên tập dữ liệu lớn.	Polynomial kernel không được ưa thích như các kernel khác vì nó kém hiệu quả và chính xác hơn so với các kernel khác.	Đây là một hàm được sử dụng nhiều nhất trong SVM. RBF kernel thực hiện phân loại thích hợp khi chưa có hiểu biết trước về dữ liệu.	Thường được sử dụng cho các mạng neural. Sigmoid kernel hoạt động như một hàm activation trong neural network.
Tham số tinh chỉnh	C	C, γ, r, d	C, γ	C, γ, r
Hiệu quả	Hoạt động tốt trên các tập dữ liệu đơn giản (ví dụ như tập dữ liệu Iris)	Kết quả tốt khi dữ liệu đã được chuẩn hóa. Với d càng lớn, thì kết quả thu được có thể dẫn tới overfitting.	Hiệu quả và được sử dụng phổ biến nhất. Tuy nhiên, việc tính toán sẽ tốn kém và phức tạp khi làm việc trên tập dữ liệu lớn.	Sigmoid kernel được được cho là có hiệu suất tương đương với RBF đối với nhiều bài toán. Tuy nhiên, Sigmoid kernel lại có hiệu suất kém trên các tập dữ liệu như Iris.

***Bảng 1**: So sánh các kernel SVM thông dụng*

Giải thích:

- C: Tham số regularization
- γ: Gamma
- r: Coefficient
- d: Degree



***Hình 6**: Trực quan hóa việc phân loại dữ liệu bằng SVM qua các kernel khác nhau*

4. Cài đặt mô hình SVM với các kernel

Scikit-learn (Sklearn) là thư viện mạnh mẽ dành cho các thuật toán học máy được viết trên ngôn ngữ Python. Thư viện cung cấp một tập các công cụ xử lý các bài toán machine learning và statistical modeling gồm: classification , regression , clustering , và dimensionality reduction. Vì thế, nhóm đã sử dụng thư viện Sklearn để cài đặt mô hình SVM. Cụ thể là hàm SVC trong sklearn.svm. Phần cài đặt được nhóm tiến hành trên Google Colab. Dưới đây là cách nhóm cài đặt và tinh chỉnh tham số.

a. Linear kernel

```
from sklearn.svm import SVC
model = SVC(C=c, kernel='linear')
model.fit(X_train, y_train)
```

Ở Linear kernel nói riêng và các Kernel khác nói chung, tham số C cho biết mức tối ưu hóa SVM để tránh phân loại sai từng ví dụ huấn luyện. Đối với C càng lớn, mô hình sẽ phân loại càng chính xác. Ngược lại, một giá trị rất nhỏ của C sẽ làm mô hình phân loại sai tương đối ngay cả khi dữ liệu đã được phân tách tuyến tính. Với C càng lớn, mô hình sẽ tính toán phức tạp và tốn thời gian hơn.

b. Polynomial kernel

```
from sklearn.svm import SVC
model = SVC(C=c, kernel='poly', degree=degree, gamma=gamma, coef0=coef0)
model.fit(X_train, y_train)
```

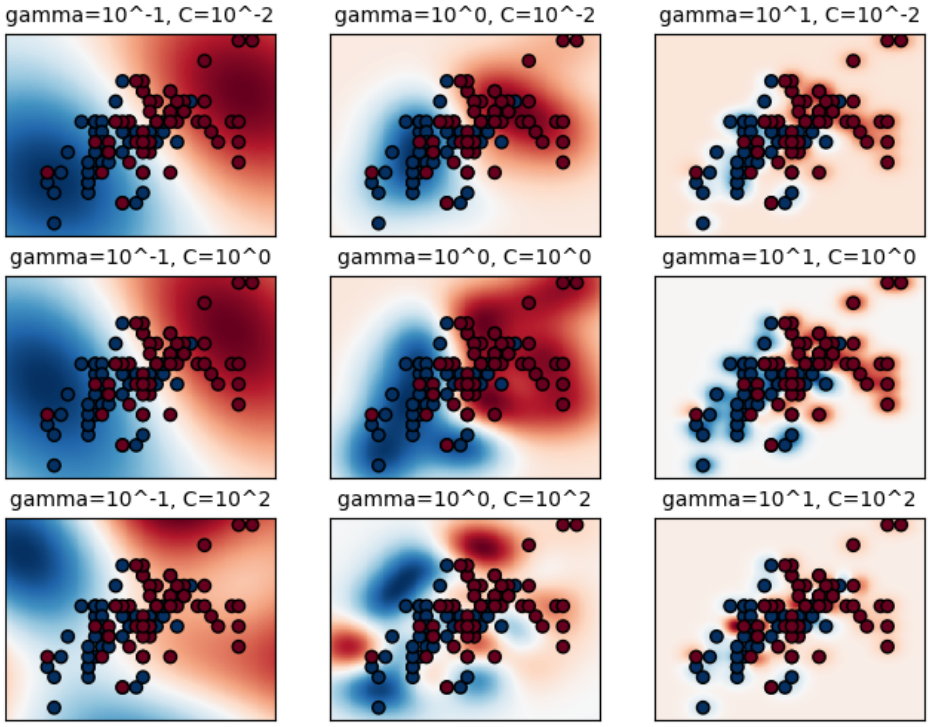
Các tham số cần tinh chỉnh của Polynomial kernel là C (tương tự Linear kernel), degree, gamma, coef0 trong đó gamma tương đương với γ, coef0 tương đương với r và degree tương đương với d trong hàm Poly kernel được nêu ở trên. Với d lớn sẽ làm mô hình bị "quá khớp" dẫn tới việc đánh giá độ chính xác trên tập dữ liệu Test thấp.

c. Radial basis function kernel

```
from sklearn.svm import SVC
model = SVC(C=c, kernel='rbf', gamma=gamma)
model.fit(X_train, y_train)
```

Ở RBF kernel, các tham số được tinh chỉnh là C (tương tự Linear) và gamma.

Tham số gamma xác định mức độ ảnh hưởng của một ví dụ đào tạo duy nhất đạt đến bao xa, với các giá trị thấp có nghĩa là "xa" và giá trị cao có nghĩa là "gần". Các giá trị gamma thấp hơn dẫn đến các mô hình có độ chính xác thấp hơn. Với gamma đủ nhỏ (0.008 đến 0.01), mô hình sẽ hoạt động như một mô hình tuyến tính. Gamma phải là một số dương.



***Hình** 7: Mô tả cách phân loại với mỗi bộ tham số của RBF kernel*

d. Sigmoid kernel

Các tham số có thể tinh chỉnh ở Sigmoid kernel là C (tương tự Linear kernel), gamma tương ứng với γ, coef0 tương ứng với r trong công thức của Sigmoid kernel.

```
from sklearn.svm import SVC
model = SVC(C=c, kernel='sigmoid', gamma=gamma, coef0=coef0)
model.fit(X_train, y_train)
```

Tùy vào bộ dữ liệu huấn luyện mà mỗi bộ tham số của mỗi kernel cho ra kết quả khác nhau. Để chọn bộ tham số tốt nhất, chúng ta phải thực nghiệm với nhiều bộ tham số khác nhau và so sánh kết quả đánh giá trên tập dữ liệu Validation.

IV. Thực nghiệm

1. Dữ liệu huấn luyện

a. Giới thiệu

Ở phần thực nghiệm, nhóm sẽ làm về bài toán "Phân loại giới tính".

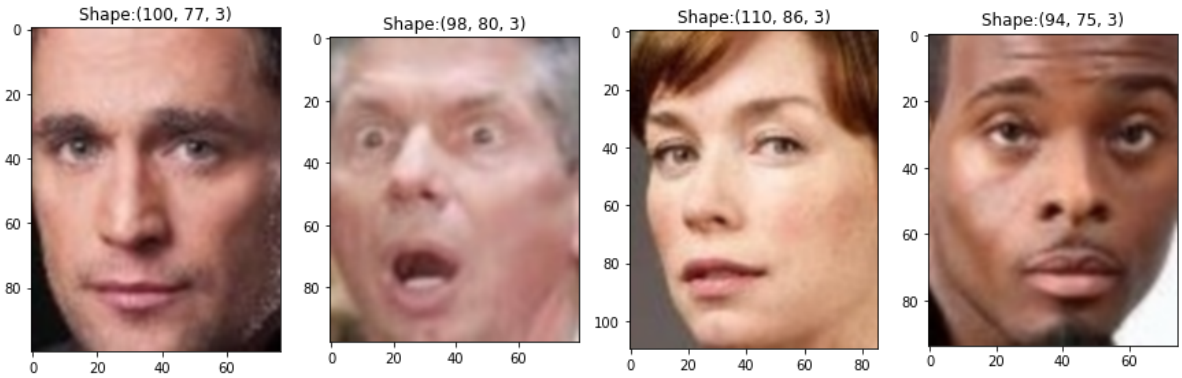
- Đầu vào: một ảnh chụp một khuôn mặt người
- Đầu ra: một nhãn giới tính tương ứng với khuôn mặt đó (male/female)

Nhóm sử dụng bộ dữ liệu "Gender Classification Dataset" được lấy từ Kaggle. Mô tả bộ dữ liệu thô lấy từ Kaggle:

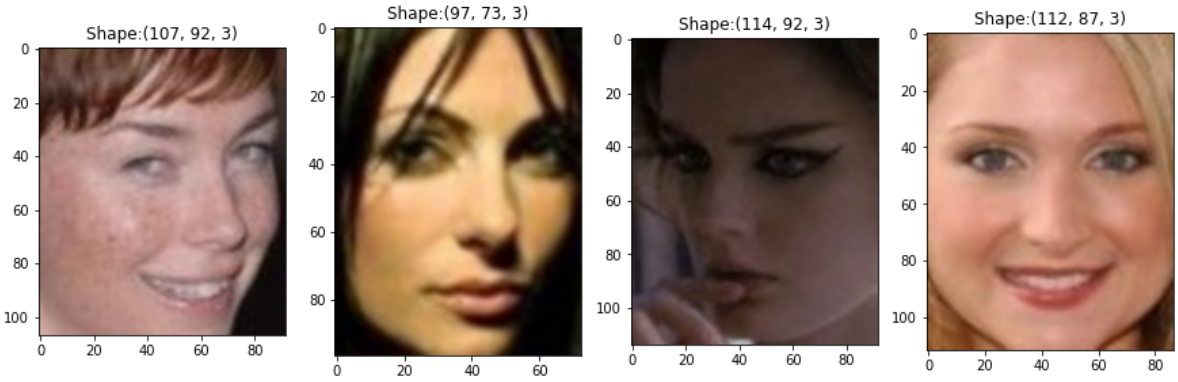
Training	female	23243 ảnh
	male	23766 ảnh
Validation	female	5841 ảnh
	male	4074 ảnh

***Bảng 2**: Mô tả dữ liệu ban đầu*

Một số ảnh từ bộ dữ liệu trên:



***Hình 8**: Một số ảnh có nhãn "male" lấy từ tập dữ liệu*



***Hình 9**: Một số ảnh nhãn "female" lấy từ tập dữ liệu*

b. Xử lý dữ liệu

Từ một số ảnh lấy từ bộ dữ liệu trên, nhận thấy bộ dữ liệu gồm các ảnh có ba kênh màu và chiều của các ảnh không giống nhau, nhóm đã thực hiện xử lý đầu vào như sau:

- Đọc các ảnh với thư viện OpenCV và chuyển các ảnh đã đọc thành ảnh xám.

```
img = cv.imread(img_dir, 0)
```

- Resize ma trận ảnh sau khi đọc về ma trận có kích thước rộng 60, cao 80

```
img = cv.resize(img, (60,80))
```

- Vì các điểm trong ma trận ảnh có giá trị nằm trong khoảng 0 đến 255 nên ta tiến hành chia mỗi giá trị trong ma trận đó cho 255

```
img = img/255
```

Sau các bước xử lý ảnh trên, nhóm đã lưu các ma trận ảnh sau khi được làm phẳng và các nhãn tương ứng với ảnh:

```
img = img.flatten()
images.append(img)
genders.append(gender)
```

Từ đó, ta nhận được bộ đầu vào cho mô hình:

```
[ ]  def GetFeature(path_trainval):
       trainval_dir = os.listdir(path_trainval)
       images = []
       genders = []
       for gender in trainval_dir:
         imgs_dir = os.path.join(path_trainval, gender)
         img_files = os.listdir(imgs_dir)
         for img_file in img_files:
           t = os.path.join(imgs_dir, img_file)
           print(t)
           try:
             img = cv.imread(t, 0) #Đọc ảnh và chuyển thành ảnh mức xám
             img = cv.resize(img, (60,80)) #Resize ảnh về rộng 60, cao 80
             img = img/255 #Vì màu nằm trong khoảng 0 đến 255 nên ta chia cho 255
             img = img.flatten() #Làm phẳng ma trận ảnh
             images.append(img)
             genders.append(gender)
           except:
             print("************************" + t)
       images = np.array(images)
       genders = np.array(genders)
       return images, genders
```

c. Vấn đề dữ liệu lớn

Sau khi được có được bộ dữ liệu đầu vào, nhóm nhận được các ma trận có kích thước sau:

- Training:
  X_train: (47009, 4800)
  y_train: (47009,)
- Validation:
  X_test: (9915, 4800)
  y_test: (9915,)

Nhận thấy dữ liệu đầu vào quá lớn, dẫn tới việc mất nhiều thời gian để huấn luyện, chi phí tính toán cao nên nhóm đã chỉ giữ lại 20% tập Training để làm dữ liệu huấn luyện. Và giữ 20% tập Validation làm tập dữ liệu Test. Lúc này:

- Training:
  X_train: (9401, 4800)
  y_train: (9401,)
- Test:
  X_test: (1983, 4800)
  y_test: (1983,)

Bộ dữ liệu sau khi đã phân tách dữ liệu lưu giữ thông tin như sau:

Training	female	4681 ảnh
	male	4720 ảnh
Test	female	1179 ảnh
	male	804 ảnh

***Bảng 3**: Mô tả dữ liệu dùng để huấn luyện*

Tiếp theo, nhóm đã dùng mô hình PCA với n_components=16 nhằm giảm bớt chi phí, giảm thời gian tính toán và chỉ lưu giữ lại những thông tin quan trọng của dữ liệu.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=16)
```

```
pca.fit(X_train)
```

```
PCA(n_components=16)
```

```
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
```

2. Huấn luyện và đánh giá

a. Linear kernel

Hàm huấn luyện và đánh giá của Linear kernel:

```
def linearSVM(c):
  linear_model = SVC(C=c, kernel="linear")
  t0 = time()
  linear_model.fit(X_train_pca, y_train)
  tt = time() - t0
  print("Linear, c =", c)
  print("    + Time training: ", tt)
  y_test_pred = linear_model.predict(X_test_pca)
  acc_test = accuracy_score(y_test, y_test_pred)
  print("    + Accuracy of Test:", acc_test)
  return tt, acc_test
```

Sau khi huấn luyện và đánh giá mô hình với các giá trị C khác nhau, nhóm thu được kết quả:

C	Độ chính xác trên tập Test	Thời gian huấn luyện (s)
10^{-9}	0.4054	4.8
0.001	0.7327	3.4
0.01	0.7378	3.9

0.1	0.7403	6.4
1	0.7398	18.5
10	0.7403	100.5
50	0.7403	404.1

***Bảng 4**: Kết quả khi huấn luyện SVM với Linear kernel*

Nhận xét:

- Độ chính xác của mô hình hầu như tăng khi C tăng, với C quá nhỏ sẽ cho ra độ chính xác thấp hơn hẳn.
- Thời gian huấn luyện tăng theo tham số C trong khoảng 0.001 đến 50.

b. Polynomial kernel

Hàm cài đặt và tính thời gian huấn luyện, độ chính xác trên tập Test:

```
from sklearn.metrics import accuracy_score
from time import time
from sklearn.svm import SVC

def polySVM(c, degree, gamma, coef):
  print("Poly, c ="+ str(c) + ", degree=" + str(degree) + ", gamma=" + str(gamma) + ", coef0=" + str(coef))
  poly_model = SVC(C=c, kernel="poly", degree=degree, gamma=gamma, coef0=coef)
  t0 = time()
  poly_model.fit(X_train_pca, y_train)
  tt = time() - t0
  print("    + Time training: ", tt)
  y_test_pred = poly_model.predict(X_test_pca)
  acc_test = accuracy_score(y_test, y_test_pred)
  print("    + Accuracy of Test:", acc_test)
  return tt, acc_test
```

- Đầu tiên, cố định C=5, degree=1, gamma=1 và thay đổi coef0 ta có kết quả sau:

coef0	**Độ chính xác trên tập Test**	**Thời gian huấn luyện (s)**
0.001	0.7403	56.7
0.03	0.7403	55.5
1	0.7403	55.7
31.6	0.7403	55.6
1000	0.7388	56.6

***Bảng 5**: Kết quả của SVM với Poly kernel khi thay đổi coef0*

- Cố định degree=1, gamma=2, coef0=1000 và thay đổi C được kết quả:

C	**Độ chính xác trên tập Test**	**Thời gian huấn luyện (s)**

0.01	0.7398	3.9
0.1	0.7403	7.6
1	0.7408	28.5
5	0.7403	99.2

***Bảng 6**: Kết quả của SVM với Poly kernel khi thay đổi C*

- Cố định C=0.1, degree=1, coef0=1.0, thay đổi gamma:

Gamma	Độ chính xác trên tập Test	Thời gian huấn luyện (s)
auto	0.7368	3.1
scale	0.7332	2.9
0.01	0.7327	3.0
0.1	0.7378	3.2
1	0.7403	5.6
2	0.7403	7.7

***Bảng 7**: Kết quả của SVM với Poly kernel khi thay đổi gamma*

- Cố định C=1.0, gamma= ‘scale’, coef0=0.0 và thay đổi degree:

Degree	Độ chính xác trên tập Test	Thời gian huấn luyện (s)
0	0.4054	2.8
1	0.7378	3.2
2	0.7009	3.8
3	0.7605	4.6
10	0.6586	9.6
15	0.647	11.3

***Bảng 8**: Kết quả của SVM với Poly kernel khi thay đổi degree*

Nhận xét:

- Sự thay đổi của coef0 có sự ảnh hưởng rất nhỏ đối với mô hình, hầu như là không đáng kể.
- Tương tự như Linear kernel, thời gian huấn luyện mô hình tăng khi C tăng và độ chính xác trên tập Test tăng không đáng kể và có biến động khi C tăng.

- Ta chưa thể đánh giá gì với gamma về độ chính xác vì sự biến động của nó trên tập Test, gamma tăng thì thời gian huấn luyện tăng
- Về degree, một degree phù hợp sẽ làm tăng độ chính xác trên tập Test, cụ thể với degree=3 thì đạt được độ chính xác trên tập Test lớn nhất, degree tăng dẫn tới tính toán phức tạp hơn nên thời gian huấn luyện tăng.

c. Radial basis function kernel

Hàm cài đặt đánh giá RBF kernel:

```
def rbfSVM(c, gamma):
  rbf_model = SVC(C=c, kernel='rbf', gamma=gamma)
  t0 = time()
  rbf_model.fit(X_train_pca, y_train)
  tt = time() - t0
  print("RBF, c =" +str(c) + ", gamma=" + str(gamma))
  print("    + Time training: ", tt)
  y_test_pred = rbf_model.predict(X_test_pca)
  acc_test = accuracy_score(y_test, y_test_pred)
  print("    + Accuracy of Test:", acc_test)
  return tt, acc_test
```

- Cố định gamma= ‘auto’, thay đổi C ta có kết quả:

C	Độ chính xác trên tập Test	Thời gian huấn luyện (s)
0.01	0.4059	5.9
0.1	0.7126	5.5
1	0.7902	7.4
5	0.7801	11

***Bảng 9**: Kết quả của SVM với Rbf kernel khi thay đổi C*

- Cố định C=5 và thay đổi gamma ta có:

Gamma	Độ chính xác trên tập Test	Thời gian huấn luyện (s)
auto	0.7801	11
scale	0.8119	4.2
0.01	0.8139	4.4
0.1	0.7731	12.2
1	0.4059	11.7
2	0.4054	12.8

***Bảng 10**: Kết quả của SVM với Rbf kernel khi thay đổi gamma*

Nhận xét:

- Với RBF kernel, ta thấy kết quả đánh giá trên tập Test cao hơn so với Linear và Poly kernel. Cụ thể với C=5 và gamma=0.01 ta có độ chính xác trên tập Test là 0.8139
- Với C tăng thì thời gian huấn luyện tăng, kết quả đánh giá trên tập Test biến động không rõ ràng.
- Với gamma tăng, thời gian huấn luyện hầu như tăng và độ chính xác trên tập Test của mô hình tăng đến một ngưỡng nào đó rồi bị hạ xuống.

d. Sigmoid kernel

Hàm cài đặt và đánh giá Sigmoid kernel:

```
from sklearn.metrics import accuracy_score
from time import time
from sklearn.svm import SVC

def sigmoidSVM(c, gamma, coef):
  sigmoid_model = SVC(C=c, kernel="sigmoid", gamma=gamma, coef0=coef)
  t0 = time()
  sigmoid_model.fit(X_train_pca, y_train)
  tt = time() - t0
  print("Sigmoid, c ="+ str(c)+ ", gamma=" + str(gamma) + ", coef0=" + str(coef))
  print("    + Time training: ", tt)
  y_test_pred = sigmoid_model.predict(X_test_pca)
  acc_test = accuracy_score(y_test, y_test_pred)
  print("    + Accuracy of Test:", acc_test)
  return tt, acc_test
```

- Cố định gamma=5, coef0=1, thay đổi C ta có:

C	Độ chính xác trên tập Test	Thời gian huấn luyện (s)
0.0001	0.4054	5.7
0.001	0.6546	5.9
0.01	0.5754	3
1	0.5673	2.6
5	0.5673	2.6
10	0.5673	2.6

***Bảng 11**: Kết quả của SVM với Sigmoid kernel khi thay đổi C*

- Cố định coef0=1, C=0.001, thay đổi gamma ta có:

Gamma	Độ chính xác trên tập Test	Thời gian huấn luyện (s)
auto	0.6677	7.6
scale	0.5239	8.1

0.01	0.6152	8.1
0.1	0.6601	7.9
1	0.6531	7.3
2	0.6541	6.4

***Bảng 12**: Kết quả của SVM với Sigmoid kernel khi thay đổi gamma*

- Cố định C=0.001, gamma=0.1, thay đổi coef0 ta có:

Coef0	Độ chính xác trên tập Test	Thời gian huấn luyện (s)
0.01	0.6646	8.3
0.03	0.6636	8.1
1	0.6601	7.9
32	0.4054	4
1000	0.4054	2.9

***Bảng 13**: Kết quả của SVM với Sigmoid kernel khi thay đổi coef0*

Nhận xét:

- Nhìn chung, độ chính xác trên tập Test của SVM khi dùng Sigmoid kernel không tốt hơn các kernel khác. Qua ba bảng 11, 12, 13, ta chưa thấy không có độ chính xác nào lớn hơn 70%.
- Khi C tăng, thời gian huấn luyện và độ chính xác trên tập Test tăng đến một ngưỡng nào đó rồi hạ xuống.
- Với gamma, thời gian huấn luyện và độ chính xác biến động không rõ ràng khi gamma tăng.
- Khi coef0 tăng, độ chính xác trên tập Test giảm và thời gian huấn luyện giảm.

3. Nhận xét chung

Qua thực nghiệm trên từng kernel, ta thấy kết quả thu được khá chính xác so với nhận xét về các kernel ở mục II.

Nhìn khái quát ta thấy SVM với RBF kernel cho kết quả tốt nhất, tuy nhiên ở phần thực nghiệm trên nhóm chưa thể thực nghiệm được các tham số lớn hơn do thời gian và chi phí tính toán của mô hình quá lớn. Do đó chưa thể khẳng định được kernel nào là mô hình tốt nhất cho bộ dữ liệu.

Vì vậy, nhóm quyết định dùng Randomized Search kết hợp KFold để tìm kernel và bộ tham số **tốt nhất có thể** cho bộ dữ liệu.

4. Tìm siêu tham số với Randomized Search

a. Các bộ siêu tham số sử dụng

```
hyper_parameters = {
    'C': [0.01, 0.1, 1, 2],
    'gamma':  ['auto', 'scale', 0.001, 0.01, 0.1, 1, 2],
    'kernel': ['rbf','linear','sigmoid', 'poly'],
    'coef0': np.logspace(-3, 3, 5).tolist(),
    'degree': [1,2]
}
```

b. Cài đặt

Nhóm đã sử dụng RandomizedSearchCV trong sklearn.model_selection với *estimator=SVC(), param_distributions=hyper_parameters, n_iter=20, cv=KFold(5), n_job=-1, scoring= "accuracy"*

- Giải thích:
  - *estimator=SVC()*: chọn mô hình là SVC()
  - *param_distributions=hyper_parameters:* truyền bộ tham số
  - *n_iter=20*: Số lần tìm kiếm ngẫu nhiên trong grid là 20
  - *cv=KFold(5)*: sử dụng xác thực chéo với bộ dữ liệu được chia thành 5 phần (4 phần huấn luyện, 1 phần đánh giá)
  - *n_job=-1*: sử dụng tất cả số luồng đang có để tính toán
  - *scoring= "accuracy"*: đánh giá mô hình bằng độ đo "accuracy"
- Code:

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import KFold
randsearch = RandomizedSearchCV(SVC(), hyper_parameters, n_iter=20, cv=KFold(5), n_jobs=-1, scoring="accuracy")
```

```
%time randsearch.fit(X_train_pca, y_train)
```

c. Kết quả thu được

```
print("----best_params_----")
print(randsearch.best_params_)
```

```
----best_params_----
{'kernel': 'rbf', 'gamma': 0.01, 'degree': 1, 'coef0': 0.001, 'C': 2}
```

```
print("----best_estimator_----")
print(randsearch.best_estimator_)
```

```
----best_estimator_----
SVC(C=2, coef0=0.001, degree=1, gamma=0.01)
```

Sau khi thực hiện tìm kiếm với RandomizedSearchCV ta tìm được mô hình SVM có *'kernel': 'rbf', 'gamma': 0.01, 'degree': 1, 'coef0': 0.001, 'C': 2*

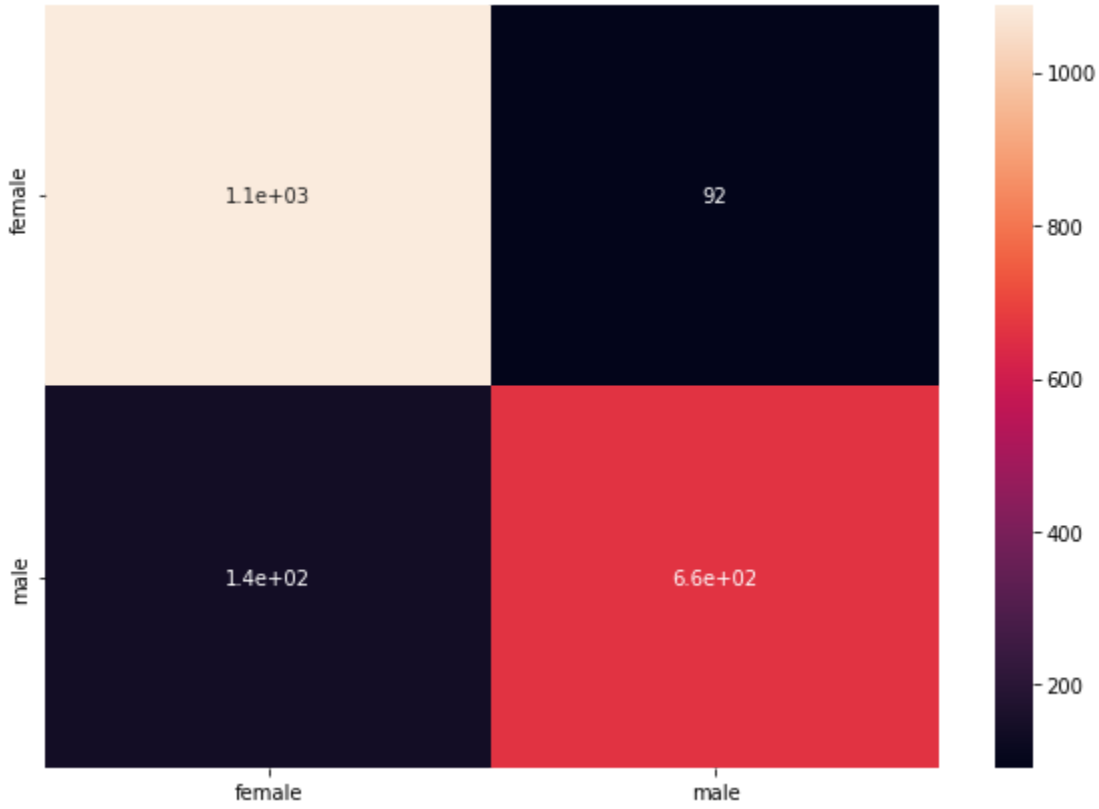
d. Đánh giá trên tập dữ liệu Test

Tiến hành đánh tính độ chính xác trên tập Test, nhóm thu được kết quả khá tốt là khoảng 0.882.

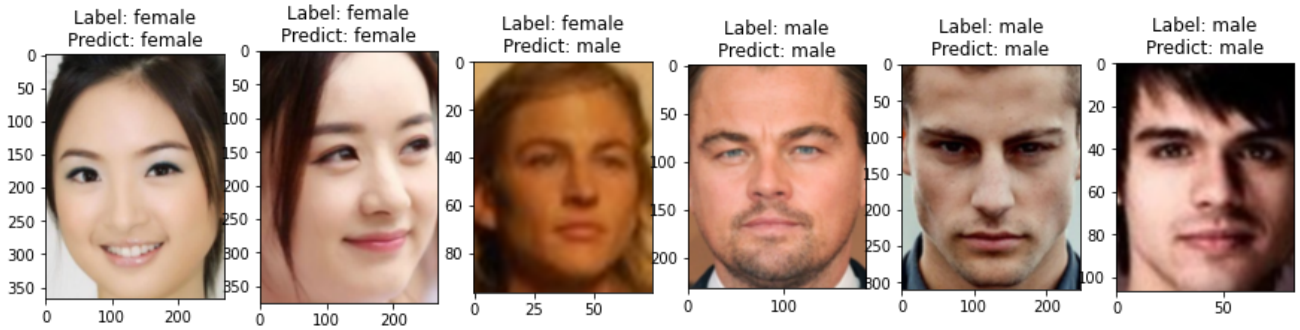
```
from sklearn.metrics import accuracy_score
model = randsearch.best_estimator_
model.fit(X_test_pca, y_test)
pred = model.predict(X_test_pca)
print("Accuracy: ", accuracy_score(y_test, pred))
```

```
Accuracy:  0.8819969742813918
```

Với confusion matrix:



e. Thử predict trên một số ảnh



***Hình 10**: Predict một số ảnh bằng mô hình SVM với bộ siêu tham số tìm được*

V. Kết luận

Kernel SVM là việc đi tìm một hàm số biến đổi dữ liệu từ không gian ban đầu sang một không gian mới, tại không gian mới này dữ liệu phân biệt tuyến tính hoặc gần như phân biệt tuyến tính. Tại đó, ta có thể dùng các bộ phân lớp tuyến tính thông thường để phân loại dữ liệu.

Có bốn loại kernel thông dụng: linear, poly, rbf, sigmoid. Trong đó, rbf được sử dụng nhiều nhất và là lựa chọn mặc định trong các thư viện SVM. Sigmoid được cho là có hiệu suất tương đương với RBF trong một số vấn đề. Với dữ liệu gần phân biệt tuyến tính, Linear và Poly cho kết quả tốt hơn.

Mỗi Kernel đều có những ưu nhược điểm riêng vì vậy để tìm ra được Kernel phù hợp người dùng phải hiểu rõ về bộ dữ liệu của mình. Bên cạnh đó còn cần phải thực nghiệm với nhiều bộ siêu tham số để tìm ra bộ siêu tham số tốt nhất đối với dữ liệu giúp mô hình SVM hoạt động hiệu quả và chính xác hơn.

Code thực hiện thực nghiệm:
https://colab.research.google.com/drive/1XqbzqzgbTzMTNTVSL1r6coj_9MermIEI?usp=sharing

VI. Tài liệu tham khảo

[1] Đinh lý merce về kernel - KhanhBlog
[2] Kernels and support vector machine regularization - Kaggle
[3] SEVEN MOST POPULAR SVM KERNELS - Dataaspirant
[4] Kernels and Feature maps: Theory and intuition - Xavierbourretsicotte
[5] sklearn.svm.SVC - Scikit-learn
[6] RBF SVM parameters - Scikit-learn
[7] https://stackoverflow.com/
[8] https://www.youtube.com/