MỤC LỤC

LỜI CẢM ƠN 3
GIỚI THIỆU... 4
CHƯƠNG I: KHÁI QUÁT VỀ NGÔN NGỮ C# 6
1.1 Khái niệm .NET 6
1.2 Hoạt động của .NET 6
1.2.1 Hỗ trợ hướng đối tượng và sử dụng giao diện. 7
1.2.3 Định kiểu mạnh. 8
1.2.4 Bắt lỗi xử dụng các ngoại lệ... 9
1.2.5 Sử dụng các thuộc tính 9
1.2 Khái quát về C# 9
1.2.1 Đặc điểm ngôn ngữ 10
1.2.2 Các kiểu nguyên (primitive)... 11
1.2.3 Khai báo (declarations) 11
1.2.4. Cấu trúc điều kiện (Conditionals structure) 12
1.2.5. Các vòng lặp (Loops) 12
1.2.6. Các phát biểu nhảy (Jumps) 12
1.2.7. Các phương thức (methods) 13
1.2.8. Các thuộc tính (properties)... 13
1.2.9. Từ chỉ định truy cập (Accessbility Modifiers).................................. 14
1.2.10. Các đối tượng, các lớp và các cấu trúc.. 14
CHƯƠNG II NHẬN DẠNG CHỮ VIẾT TAY SỬ DỤNG HẠT NHÂN PHÂN TÍCH BIỆT THỨC.. 16
2.1. Phân tích biệt thức tuyến tính... 16
2.2. Nhân Phân tích biệt thức 17
2.3. Chức năng nhân trick và nhân chuẩn. 17
2.4. Mô hình các lớp được sử dụng trong KDA.. 18
2.5. Nhận dạng chữ số 19
2.5.1. Dữ liệu số của UCI 19

2.5.2 Phân lớp các chữ số số bằng KDA. 21
CHƯƠNG III CHƯƠNG TRÌNH THỬ NGHIỆM 23
3.1. Kiểm tra ứng dụng 23
3.1.1 Phân tích 23
3.1.2 Kết quả 25
3.2 Mã lệnh trong chương trình viết cho một số các class 28
3.2.1 Class Linear Discrimnant Analysis 28
3.2.2 Kernel Discrimnant Analysis 43
KẾT LUẬN 49
TÀI LIỆU THAM KHẢO 50

LỜI CẢM ƠN

Lời đầu tiên em xin được bày tỏ lòng biết ơn sâu sắc tới thầy giáo ThS. Đỗ Văn Chiểu, đã dành rất nhiều thời gian quý báu để tận tình giúp đỡ, chỉ bảo và hướng dẫn em hoàn thành tốt đồ án tốt nghiệp này.

Em xin gửi lời cảm ơn đến ban giám hiệu nhà trường và các thầy, cô giáo của trường dân lập Hải Phòng đặc biệt là các thầy, cô khoa công nghệ thông tin đã giảng dạy chúng em trong suốt quãng thời gian qua, cung cấp cho chúng em những chuyên môn cần thiết và quý báu giúp chúng em hiểu rõ hơn các lĩnh vực đã nghiên cứu để hoàn thành đề tài đươc giao.

Cuối cùng em xin cảm ơn sự quan tâm, chăm sóc của gia đình, sự động viên, giúp đỡ của bạn bè đã tạo điều kiện giúp đỡ em hoàn thành tốt quá trình nghiên cứu thực tập và thực hiện đề tài này.

Hải Phòng, ngày 18 tháng 10 năm 2010

GIỚI THIỆU

Trong thời đại ngày nay Cong nghệ thông tin hầu như đã thâm nhập vào toàn bộ các lĩnh vực đời sống xã hội. Xã hội càng phát triển thì nhu cầu về công nghệ thông tin ngày càng cao, do vậy dữ liệu số hầu như không còn xa lạ đối với mỗi người chúng ta. Trong mọi lĩnh vực các ứng dụng công nghệ thông tin đã trợ giúp con người rất nhiều. Công nghệ thông tin được ứng dụng một cách mạnh mẽ mọi nơi, mọi lúc. Ta dường như có thể thấy máy tính xuất hiện ở mọi nơi trong các doanh nghiệp, cơ quan, trường học…Mọi công việc đều đòi hỏi liên quan tới Công nghệ thông tin như: quản lý học sinh, sinh viên, quản lý nhân sự và lương, thiết kế cố vấn kiến trúc xây dựng cho đến các ngành nghề đòi hỏi chất lượng và chuyên môn sâu. Ngành nào nghề nào cũng có sự góp mặt của Công nghệ thông tin, từ giáo dục, y tế, giao thông, xây dựng, viễn thông….

Trong ngành Công nghệ thông tin, thông tin hình ảnh đóng vai trò rất quan trọng trong trao đổi thông tin, bởi phần lớn các thông tin mà con người thu nhận được đều thông qua thị giác. Một loạt các bài toán được đặt ra và yêu cầu được giải quyết như: nhận dạng các tối tượng chuyển động, xử lý và chọn mẫu màu, nhận dạng chữ viết tay… Bài toán nhận dạng tuy ra đời từ thập niên 60 của thế kỷ trước nhưng vẫn luôn nhận được sự quan tâm, nghiên cứu của nhiều nhà khoa học trên thế giới.Đặc biệt là trong những thập niên gần đây, cùng với quá trình đẩy mạnh tin học hóa trong mọi lĩnh vực đời sống xã hội, nhận dạng không chỉ còn là lĩnh vực nghiên cứu lý thuyết nữa mà đã được ứng dụng rộng rãi trong thực tế cuộc sống. Các bài toán nhận dạng được nghiên cứu nhiều nhất hiện nay bao gồm nhận dạng các mẫu hình học (vân tay, mặt người, hình khối,…), nhận dạng tiếng nói và nhận dạng chữ viết. Chúng được áp dụng vào nhiều lĩnh vực như y học, dự báo thời tiết, dự báo cháy rừng, điều khiển robot,...Trong các bài toán nhận dạng này, nhận dạng chữ viết là bài toán đang được ứng dụng phổ biến nhất hiện nay.

Nhận dạng chữ viết bao gồm hai kiểu chính là nhận dạng chữ in và nhận dạng chữ viết tay. Cho đến nay bài toán nhận dạng chữ in đã được giải quyết khá trọn vẹn với sự ra đời của nhiều hệ thống nhận dạng đạt tới độ chính xác gần như tuyệt đối. Tiêu biểu có hệ nhận dạng chữ in dựa trên mô hình mạng nơron bốn lớp của J. Wang và J.S.N. Jean đạt tới tỷ lệ chính xác 99.75%. Ở Việt Nam

hiện đã có sản phẩm VNDOCR của Viện Công nghệ thông tin nhận dạng chữ in tiếng Việt với độ chính xác tới 99%.

Trong nhận dạng chữ viết tay thì nhận dạng chữ viết tay được chia thành hai lớp bài toán lớn là nhận dạng chữ viết tay trực tuyến (online) và nhận dạng chữ viết tay ngoại tuyến (offline). Nhận dạng chữ viết tay trực tuyến là nhận dạng các chữ trên màn hình ngay khi nó được viết. Trong hệ nhận dạng này máy tính sẽ lưu lại các thông tin về nét chữ như thứ tự nét viết, hướng và tốc độ của nét… Ngược lại, trong nhận dạng chữ viết tay ngoại tuyến, dữ liệu đầu vào được cho dưới dạng các ảnh được quét từ các giấy tờ, văn bản.

Vấn đề nhận dạng chữ viết tay đã được đề cập và nghiên cứu trong nước từ khá lâu và đạt được kết quả với các phương pháp như Support Vecto Machine (SVM). Tuy nhiên, việc ứng dụng các nhân (Kernel) vào việc nhận dạng chưa được nghiên cứu nhiều. Chính vì thế mà em đã chọn đề tài: ***Tìm hiểu phương pháp nhận diện chữ viết tay sử dụng Kernel Discriminant Analysis*.**

Nội dung chính của khóa luận bao gồm các phần sau: phần mở đầu, phần kết luận, ba chương nội dung, cụ thể:

Chương 1: Khái quát về ngôn ngữ C#.

Chương 2: Giới thiệu về phương pháp Kernel Discriminant Analysis.

Chương 3: Chương trình thử nghiệm.

CHƯƠNG I:
KHÁI QUÁT VỀ NGÔN NGỮ C#

C# là một ngôn ngữ lập trình hướng đối tượng mới. Cấu trúc và lập luận của C# có đầy đủ các đặc tính của một ngôn ngữ lập trình hướng đối tượng trước đó (C++, Java). C# được thiết kế dùng cho nền .NET framework, một công nghệ mới và đầy triển vọng trong việc phát triển các ứng dụng hệ thống và mạng internet.

Bên cạnh đó, C# còn là một ngôn ngữ lập trình hoàn toàn độc lập, điều đó có nghĩa là mã của C# được chạy trên .NET nhưng có những đặc tính của C# mà .NET không hỗ trợ (quá tải toán tử) hay là những đặc tính của .NET mà C# không hỗ trợ.

1.1 Khái niệm .NET

.NET Framework là một thư viện class có thể được sử dụng với một ngôn ngữ .NET để thực thi các việc từ thao tác chuỗi cho đến phát sinh ra các trang web động (ASP.NET), phân tích XML và reflection. .NET Framework được tổ chức thành tập hợp các namespace, nhóm các class có cùng chức năng lại với nhau, thí dụ như System.Drawing cho đồ hoạ, System.Collections cho cấu trúc dữ liệu và System.Windows.Forms cho hệ thống Windows Forms.

.NET là một môi trường quản lý, phát triển và thực thi các mã ngôn ngữ biết .NET. .NET cung cấp các khả năng về cấp phát và thu hồi bộ nhớ, quản ly cấp quyền, cung cấp và quản lý các nguồn tài nguyên. Trọng tâm của .NET bao gồm 2 thành phần là CLR (the comon language runtime) và .NET framework class libary – Các thư viện cơ sở.

1.2 Hoạt động của .NET

Mã chương trình sẽ được biên dịch thành MSIL (Microsoft Intermediate Language).

Dịch IL(Intermediate Language) thành nền cụ thể của .NET bằng CLR(Common Language Runtime).

Có rất nhiều ngôn ngữ biết .NET, bao gồm C++, VB.NET, Managed C++, J+ and J#, Scripting languages, COM. Mã của chúng cũng sẽ được biên dịch thành IL. IL sẽ đảm bảo sự tương thích giữa các ngôn ngữ khác nhau. 1 thành

phần của ngôn ngữ này có thể sử dụng thành phần và thuộc tính của thành phần nằm trong ngôn ngữ khác. Đây có thể nói là một khả năng kì diệu của C#. Để đạt được những điều đó, IL bao hàm những thuộc tính sau:

1. Hỗ trợ hướng đối tượng và giao diện (interface).
2. Phân biệt giữa kiểu giá trị và kiểu tham chiếu.
3. Định kiểu mạnh.
4. Quản lỗi thông qua các ngoại lệ.
5. Sử dụng các thuộc tính.

1.2.1 Hỗ trợ hướng đối tượng và sử dụng giao diện.

IL tạo nhiều thuận lợi với các ngôn ngữ lập trình hướng đối tượng, không phải vô tình mà các thư viện cơ sở của .NET đều được viết = C# (OOP). IL cũng đưa ra ý tưởng về giao diện (interface). Windows cũng hỗ trợ chuẩn giao diện gọi là COM.

COM là một nhị phân chuẩn cho phép các thành phần có thể tương tác với nhau mà không cần quan tâm đến ngôn ngữ nào đã tạo lập ra chúng. Điều đó có nghĩa là các mã dịch ra đều thống nhất và tương thích với COM. Tuy rằng COM không hỗ trợ tính thừa kế, chính vì thế COM đánh mất sự thuận lợi của lập trình hướng đối tượng.

Tương thích chéo

Với khả năng hỗ trợ đồng thời nhiều loại ngôn ngữ, sau khi được biên dịch thành IL, mã của các ngôn ngữ khác nhau có thể làm việc cùng với nhau. Cụ thể là một lớp được tạo ra trong một ngôn ngữ có thể thừa kế từ một lớp được viết trong một ngôn ngữ khác.

Một lớp có thể chứa thể hiện của một lớp khác không quan tâm đến ngôn ngữ đã tạo ra hai lớp đó.

Một đối tượng có thể gọi trực tiếp phương thức của một đối tượng khác được viết bởi một ngôn ngữ khác.

Các đối tượng (hoặc các tham chiếu đến các đối tượng) có thể được truyền qua lại giữa các hàm

Bạn có khả năng bẫy lỗi từng bước chương trình nguồn giữa các ngôn khác nhau

1.2.2 Phân biệt kiểu giá trị và kiểu tham chiếu.

IL có sự phân biệt rõ ràng đối với kiểu giá trị và kiểu tham chiếu. Trên IL, các kiểu giá trị vẫn được lưu trong vùng Stack, các kiểu tham chiếu vẫn được lưu trong vùng Heap.

1.2.3 Định kiểu mạnh.

IL có sự phân biệt rõ ràng đối với từng kiểu dữ liệu trả về, các kiểu dữ liệu luôn được đánh dấu cụ thể. Điều này là hoàn toàn phù hợp với đặc tính hỗ trợ nền cho nhiều loại ngôn ngữ của .NET. Một vấn đề nảy sinh đó là có những kiểu được hỗ trợ trong ngôn ngữ này nhưng lại không được hỗ trợ trong ngôn ngữ khác hoặc là nếu một lớp xuất thân hoặc chứa một lớp khác thì nó cần phải biết tất cả các kiểu dùng trong các lớp đó.

CTS: Để đáp ứng được tác vụ đó, IL sử dụng tiến trình CTS – Common Type System, đây vốn là một bộ con trong .NET, đảm bảo tất cả các kiểu dữ liệu khác nhau của các ngôn ngữ khác nhau đều được biên dịch thành một kiểu chung trên nền .NET

*CLS :*CLS phối hợp với CTS để đảm bảo sự tương thích giữa các ngôn ngữ. CLS là một chuẩn mà tất cả các ngôn ngữ biết .NET đều phải tuân theo. CLS hoạt động theo 2 nguyên tắc

CLS không hoàn tòan bó buộc các ngôn ngữ lập trình, điều này khiến cho các ngôn ngữ hoàn toàn có thể phát triển theo các chiều hướng riêng.

CLS gắn một chuẩn lên các ngôn ngữ lập trình biết .NET, điều này đảm bảo mã của các ngôn ngữ đó luôn được hỗ trợ khi biên dịch.

Garbare Collection

Garbage collector là một thành phần quản lí bộ nhớ của .NET. Tốc độ hoạt động của C# hoàn toàn phụ thuộc vào Garbare collection, GC là một ứng dụng có mục đích giải phóng bộ nhớ trên nền .NET. Nguyên tắc làm việc của GC như sau.

Các mã sau khi được biên dịch, kết quả sẽ được đưa hoàn toàn vào Heap, khi Heap đầy, GC sẽ thực thi so sánh với các mã đang thực hiên, nếu như các kết quả không dùng đến, GC sẽ thực hiện nhiệm vụ dọn dẹp và lấy lại bộ nhớ.

1.2.4 Bắt lỗi xử dụng các ngoại lệ

.NET được thiết kế để đơn giản hoá quá trình bẫy lỗi thông qua các ngoại lệ, ư tưởng ở đây là một vùng mã được thiết kế như là các thủ tục quản ngoại lệ, mỗi đoạn mã có thể giải quyết một điêu kiện lỗi riêng (ví dụ, một file không được ́tm thấy, hoặc không được phép thực thi một số lệnh). Những điều kiện này có thể được định nghĩa kĩ hoặc sơ qua tuỳ bạn. Cấu trúc ngoại lệ bảo đảm rằng khi một điều kiện sinh lỗi xảy ra, ngay lập tức luồn thi hành sẽ nhảy đến thủ tục quản ngoại lệ.

1.2.5 Sử dụng các thuộc tính

Các thuộc tính trong IL cho phép người dùng có thể sử dụng dễ dàng hoặc có thể tự thiết lập các thuộc tính của riêng họ.

Tiến trình biên dịch thành mã .NET – Common language runtime CLR có nhiệm vụ biên dịch mã IL thành mã nền .NET. Đây là một tiến trình biên dịch kiểu just in time (JIT), khác với kiểu thông dịch trong Java. Thay vì phải dịch toàn bộ ứng dụng một lần, trình biên dịch JIT sẽ biên dịch từng phần mã khi nó được gọi Khi mã nguồn được biên dich, mã kết quả của nó sẽ được lưu lại trong bộ nhớ cho tới khi thoát khỏi ứng dụng, và trong các lần xử lý tiếp theo, máy tính sẽ không phải biên dịch lại một lần nữa, đây là lý do các chương trình .NET luôn chạy nhanh hơn trong những lần sau. Một đặc điểm nữa là .NET luôn hỗ trợ tối ưu tuỳ vào loại vi xư lý, đối với các tiến trình biên dịch các ngôn ngữ cấp cao xưa kia sẽ không tối ưu vào loại vi xử lư, nền .NET hỗ trợ tuỳ loại vi xử l mà đưa ra cách thức phù hợp.

1.2 Khái quát về C#

C# là một ngôn ngữ lập trình hướng đối tượng được phát triển bởi Microsoft, là phần khởi đầu cho kế hoạch .NET của họ. Tên của ngôn ngữ bao gồm ký tự thăng theo Microsoft nhưng theo ECMA là **C#**, chỉ bao gồm dấu số thường. Microsoft phát triển C# dựa trên C++ và Java. C# được miêu tả là ngôn ngữ có được sự cân bằng giữa C++, Visual Basic, Delphi và Java.

C# được thiết kế chủ yếu bởi Anders Hejlsberg kiến trúc sư phần mềm nổi tiếng với các sản phẩm Turbo Pascal, Delphi, J++, WFC.

1.2.1 Đặc điểm ngôn ngữ

C#, theo một hướng nào đó, là ngôn ngữ lập trình phản ánh trực tiếp nhất đến .NET Framework mà tất cả các chương trình .NET chạy, và nó phụ thuộc mạnh mẽ vào Framework này. Mọi dữ liệu cơ sở đều là đối tượng, được cấp phát và hủy bỏ bởi trình dọn rác Garbage-Collector (GC), và nhiều kiểu trừu tượng khác chẳng hạn như class, delegate, interface, exception, v.v, phản ánh rõ ràng những đặc trưng của .NET runtime.

So sánh với C và C++, ngôn ngữ C# bị giới hạn và được nâng cao ở một vài đặc điểm nào đó, nhưng không bao gồm các giới hạn sau đây:

- Các con trỏ chỉ có thể được sử dụng trong chế độ không an toàn. Hầu hết các đối tượng được tham chiếu an toàn, và các phép tính đều được kiểm tra tràn bộ đệm. Các con trỏ chỉ được sử dụng để gọi các loại kiểu giá trị; còn những đối tượng thuộc bộ thu rác (*garbage-collector*) thì chỉ được gọi bằng cách tham chiếu.
- Các đối tượng không thể được giải phóng tường minh.
- Chỉ có đơn kế thừa, nhưng có thể cài đặt nhiều interface trừu tượng (abstract interfaces). Chức năng này làm đơn giản hóa sự thực thi của thời gian thực thi.
- C# thì an-toàn-kiểu (*typesafe*) hơn C++.
- Cú pháp khai báo mảng khác nhau("int[] a = new int[5]" thay vì "int a[5]").
- Kiểu thứ tự được thay thế bằng tên miền không gian (*namespace*).
- C# không có tiêu bản.
- Có thêm Properties, các phương pháp có thể gọi các Properties để truy cập dữ liệu.
- Có reflection.

Trong C# 3.0, sẽ có vài bổ sung cơ bản sau:

- Các từ khóa "select, from, where" cho phép truy vấn từ một tập, từ SQL, v.v. (hay còn được gọi là LINQ - viết tắt của Language INtergrated Query)

- Khởi tạo đối tượng: Customer c = new Customer(); c.Name="James"; trở thành Customer c = new Customer { Name="James" };
- Các biểu thức lambda: listOfFoo.Where(delegate(Foo x) { return x.size>10;}) trở thành listOfFoo.Where(x => x.size>10);
- var x = "hello"; có thể hoán đổi với string x = "hello";
- Các phương thức mở rộng

1.2.2 Các kiểu nguyên (primitive)

C# sử dụng hệ thống kiểu/đối tượng trong .NET mà ở đó, các chương trình C# có thể giao tiếp với nhiều ngôn ngữ khác trong .NET mà không gặp rắc rối nào về kiểu. Ví dụ, kiểu int là một bí danh của System.Int32 được kế thừa cuối cùng từ System.Object. Điều này có nghĩa là các kiểu primitive, hay kiểu simple trong hàm C# cũng giống như bất kỳ các đối tượng khác. Ví dụ, điều này là đúng khi gọi phương thức toString hoặc GetType trong bất kỳ một kiểu primitive nào.

Mặc dù các kiểu simple trong C# là những đối tượng, tuy nhiên chúng vẫn được truyền theo tham trị (pass-by-value) tương tự như trong Java. Đây là trường hợp khác, bởi vì ngoài việc là những đối tượng, tất cả các kiểu simple trong C# đều là các đối tượng – cấu trúc (struct) khi được truyền theo tham trị sẽ được truyền theo tham biến một lần nữa.

1.2.3 Khai báo (declarations)

Trong C# có 2 cách để biết một biến hằng:

Đánh dấu một biến bằng từ khóa const sẽ làm cho giá trị được chuyển đổi trước khi biên dịch. Với định nghĩa sau:

const int two = 2;

phát biểu: 2 * two được chuyển thành 2 * 2 xử lý trước khi biên dịch,điều này làm cho nó chạy nhanh hơn không phải tìm những giá trị hằng trong suốt quá trình chạy.

1.2.4. Cấu trúc điều kiện (Conditionals structure)

Trong C# cũng có các cấu trúc “if-then-else”, “switch” nhưng trong switch không cho phép dòng điều khiển phải rơi vào chính xác trong các trường hợp khác nhau của phát biểu switch

1.2.5. Các vòng lặp (Loops)

Ngoài những dòng lặp: while, do-while, for. C# còn có foreach

Ví dụ:

foreach(int member in array)
Console.Writeln(member);

* Giao tiếp Ienumerable cung cấp khả năng nhận được một sự thay thế Ienumerator cho một đối tượng. Bất kỳ cái gì bổ sung các giao tiếp Ienumerable và Ienumerator đều có thể được tính toán trên vòng lặp foreach.

1.2.6. Các phát biểu nhảy (Jumps)

Các phát biểu nhảy trong trong C#: continue, break, goto, return. Các phát biểu này được trình bày vắn tắt như sau: thoát khỏi các vòng lặp hoặc trả dòng điều khiển cho một khối lệnh khác.

Trong C# đều là các ngoại lệ run-time, trình biên dịch sẽ không giúp đỡ các lập trình viên giữ lại trạng thái của các ngoại lệ.

Có sẵn một cách đi tắt nếu đối tượng ngoại lệ không cần thiết bằng việc sử dụng framework dưới đây

```
Try
 {
// những lệnh có thể gây ra ngoại lệ
}
catch {
// xử lý những ngoại lệ mà không cần nhận một bộ xử lý cho đối tượng ngoại lệ
}
```

Nhưng nếu bạn cần bắt một ngoại lệ cụ thể (trong khi không yêu cầu đối tượng ngoại lệ), có thể dùng tương tự như sau:

```
try {
}
catch (IOException) {
}
```

1.2.7. Các phương thức (methods)

```
public void methodCaller ( params int[] a );
```

Và phương thức có thể được gọi bất kỳ

```
methodCaller ( 1 );
method Caller ( 1, 2, 3, 4 );
```

Bên trong phương thức, các tham số có thể được truy cập thông qua dãy "a" đã được định nghĩa. Máy tính có thể hiểu, kết quả được tìm thấy

1.2.8. Các thuộc tính (properties)

Các thuộc tính là các khởi dựng của C# thường được dùng với mô hình (pattern) getter/setter trong nhiều lớp của Java.

```
private int property;
public int Property () {
get {
return this.property;
}
set {
// value là một biến được tạo ra bởi trình biên dịch để thay thế các tham
số
this.property = value;
}
}
```

Có thể dễ dàng sử dụng bên trong một chương trình C#

```
int currentValue = Property;
Property = new Value;
```

1.2.9. Từ chỉ định truy cập (Accessbility Modifiers)

Access modifier bao gồm: Public, protected, internal, protected internal, private.

Các modifier trên có thể được áp dụng cho cùng các cấu trúc mà Java cho phép bạn sử dụng chúng. Bạn có thể thay đổi khả năng truy cập đến các đối tượng, các phương thức và các biến. Chúng ta sẽ nói về chúng ngay dưới đây, và chúng ta có thể nói về các đối tượng và kế thừa ở phần tiếp theo.

1.2.10. Các đối tượng, các lớp và các cấu trúc

Kế thừa các lớp ta dùng ":"

Ví dụ: class B : A {} nghĩa là lớp B kế thừa từ lớp A

_Struct được truyền theo tham trị thay vì theo tham biến

_Struct không thể kế thừa, tuy nhiên chúng có thể bổ sung các giao tiếp

_Struct không thể được định nghĩa một khởi dựng (contructor) mà không có tham số

_Struct định nghĩa các contructor với các tham số phải định nghĩa chính xác tất cả các field bởi vì nó sẽ trả về điều khiển cho phương thức nào gọi nó

1.2.11. Chuyển đổi kiểu

C# cho phép khả năng định nghĩa chuyển đổi kiểu tự tạo cho hai đối tượng bất kỳ. Hai kiểu chuyển đổi là: chuyển đổi tương đối và chuyển đổi tuyệt đối

1.2.12. Tải chồng toán tử (Operator overloading)

Tải chồng toán tử trong C# rất đơn giản. Lớp FlooredDouble có thể được thừa kế để chứa một phương thức static

```
public static FloorDouble operator + ( FloorDouble fd1, FloorDouble fd2 ) {
return new FloorDouble( fd1.Value + fd2.Value );
}
```

Và các phát biểu sau là đúng

```
FloorDouble fd1 = new FloorDouble( 3.5 );
FloorDouble fd2 = new FloorDouble( 4 );
FloorDouble sum = fd1 + fd2;
```

1.2.13. Tổ chức lại mã nguồn

C# không đặt bất kỳ yêu cầu nào trong việc tổ chức file – một lập trình viên có thể sắp xếp toàn bộ chương trình C# bên trong một file .cs

1.2.14.Giao diện

*Giống như class chỉ gồm toàn các hàm trừu tượng

*Khi một class thiết đặt(implement) một giao diện thì phải thi công tất cả các hàm giao diện này.

+Thiết đặt một giao diện.

+Truy xuất các hàm giao diện.

+Override một thiết đặt giao diện.

+Dùng giao diện như thông số.

+Thiết đặt kế thừa giao diện.

CHƯƠNG II
NHẬN DẠNG CHỮ VIẾT TAY SỬ DỤNG HẠT NHÂN PHÂN TÍCH BIỆT THỨC

2.1. Phân tích biệt thức tuyến tính

Phân tích biệt thức tuyến tính (Linear Discriminant Analysis (LDA)) là một phương pháp được sử dụng trong thống kê và máy học để tìm một sự kết hợp tuyến tính của các tính năng đặc trưng nhất hoặc chia tách hai hay nhiều lớp của đối tượng, sự kiện. Kết quả của sự kết hợp có thể được sử dụng như một bộ phân loại tuyến tính (linear classifier), hoặc thông thường hơn đối với giảm đa chiều (dimensionality reduction) trước khi phân loại cuối.

Phân tích biệt thức tuyến tính liên quan chặt chẽ tới phân tích thành phần chủ yếu (Principal Component Analysis (PCA)) trong trường hợp cả hai tìm kiếm sự kết hợp tuyến tính của các biến được miêu tả dữ liệu một cách tốt nhất. LDA đã nỗ lực để xây dựng lên sự khác biệt giữa các lớp dữ liệu, tăng tối đa tính đúng đắn của công thức sau:

$$J(w) = \frac{w^T S_B w}{w^T S_W w}$$

Trong đó:

$$S_B = \sum_{c=1}^{C} (\mu_c - \bar{x})(\mu_c - \bar{x})^T \qquad S_W = \sum_{c=1}^{C} \sum_{i \in c} (x_i - \mu_c)(x_i - \mu_c)^T$$

Tương ứng là các Ma trận tán xạ lớp giữa (Between-Class Scatter Matrix) và Ma trận phân tán lớp trong (Within-Class Scatter Matrix). Giải pháp tối ưu có thể được tìm ra được bởi máy tính là Giá trị riêng (Eigen values) của $S_B^{-1}S_W$ và lấy Véctơ riêng ứng với giá riêng lớn nhất để định ra cơ sở mới cho dữ liệu.

2.2. Nhân Phân tích biệt thức

KDA là một mở rộng của LDA để trở thành phi tuyến tính phân tán, cũng giống như KPCA là PCA. Mục tiêu của KDA cũng là tìm sự biến đổi giữa phương sai lớp giữa cực đại và phương sai lớp trong cực tiểu. Điều đó cho thấy rằng, với nhân, công thức gốc cuối cùng được biểu diễn như sau:

$$J(\alpha) = \frac{\alpha^T S_B \alpha}{\alpha^T S_W \alpha}$$

Trong đó:

$$S_B = \sum_{c=1}^{C} (\mu_c - \bar{x})(\mu_c - \bar{x})^T \qquad S_W = \sum_{c=1}^{C} K_c (I - 1_{l_c}) K_c^T$$

Với **K** $_c$ là ma trận nhân cho **lớp c**;

u $_c$ là cột biểu diễn vecto **K** $_c$,

I là ma trận nhận dạng;

l $_c$ là số mẫu trong lớp **c** ;

1 $_{lc}$ là ma trận (**l** $_c$ **xl** $_c$) với tất cả phần tử **1 / l** $_c$ (nghĩa là **I - 1 / l** $_c$ là trung ma trận của kích thước I_c).

2.3. Chức năng nhân trick và nhân chuẩn.

Nhân trick là một công cụ rất tiện ích và mạnh mẽ. Mạnh mẽ bởi vì nó cung cấp một cầu nối từ tuyến tính đến phi tuyến tính cho bất kỳ thuật toán mà chỉ phụ thuộc vào giao điểm (dot product) giữa hai vectơ. Nó đến từ thực tế là, nếu đầu tiên chúng ta xây dựng bản đồ dữ liệu đầu vào vào một không gian chiều vô hạn, một thuật toán tuyến tính sẽ hoạt động trong không gian này sẽ xử lý phi tuyến tính trong không gian ban đầu.

Bây giờ, các nhân trick thực sự hữu ích bởi vì nó lập bản đồ không bao giờ cần phải được tính toán. Nếu thuật toán của chúng ta chỉ có thể được thể hiện với điều kiện trong miền giữa hai vectơ thì tất cả điều chúng ta cần là thay thế tích trong này với tích trong khác từ một số không gian thích hợp khác. Đó là nơi chứa "trick": bất cứ nơi nào một giao điểm được sử dụng thì nó được thay thế bằng một hàm nhân. Hàm nhân biểu thị một tích trong vùng đặc trưng và thường được ký hiệu là:

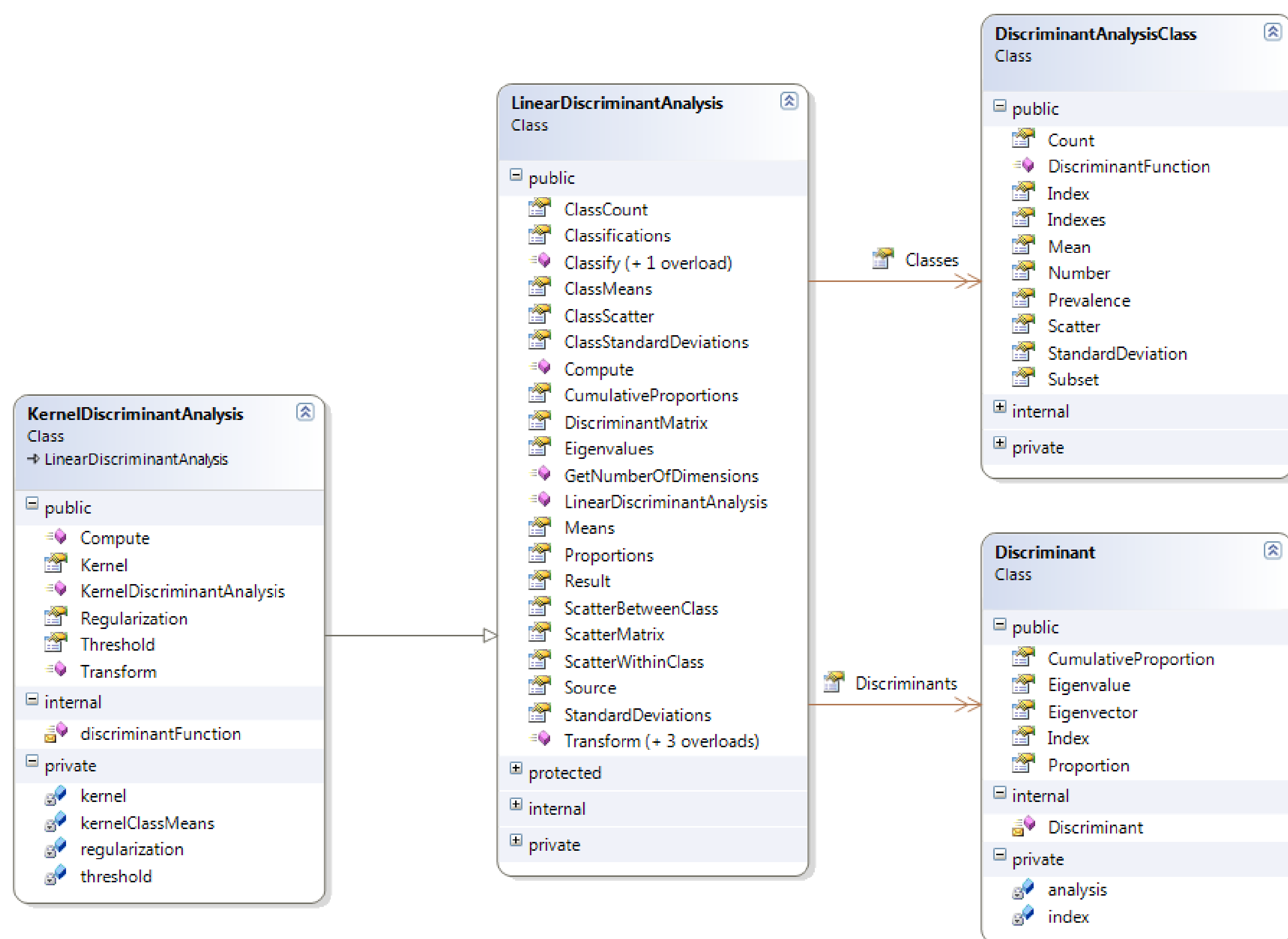
K(x,y) = <φ(x),φ(y)> K (x, y) = <φ (x), φ (y)>

Sử dụng hàm nhân, thuật toán sau đó có thể được áp dụng vào một không gian nhiều chiều hơn kích thước mà không lập bản đồ một cách rõ ràng các điểm đầu vào không gian này. Một số hàm nhân thông dụng bao gồm các nhân tuyến tính, nhân đa thức, nhân Gaussian. Dưới đây là danh sách lược các nhân với đặc điểm thú vị nhất.

Nhân tuyến tính	Nhân tuyến tính là hàm nhân đơn giản nhất. Nó được đưa ra bởi các tích trong <x,y> cộng với một hằng số **c** tùy chọn. Thuật toán nhân sử dụng một nhân tuyến tính thường tương đương với các nhân phi tuyến, tức là KPCA với nhân tuyến tính tương đương chuẩn PCA.	$k(x,y) = x^T y + c$
Nhân đa thức	Nhân đa thức là một nhân không tĩnh. Nó rất thích hợp cho các vấn đề tất cả dữ liệu là bình thường.	$k(x,y) = (\alpha x^T y + c)^d$
Nhân Gaussian	Nhân Gaussian là một trong rất nhiều những nhân linh hoạt nhất. Đây là một hàm nhân cơ sở dạng tia và là nhân được ưa thích khi chúng ta không biết nhiều về cấu trúc của dữ liệu chúng ta đang cố gắng xây dựng mô hình.	$k(x,y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$

2.4. Mô hình các lớp được sử dụng trong KDA

Các mã nguồn có trong bài báo cáo tốt nghiệp này chỉ chứa các tập con của mảng cần thiết cho KDA. Các lớp cũng được hiển thị trong hình dưới đây.



Các lớp KDA kế thừa từ Phân tích biệt thức tuyến tính, mở rộng thuật toán cốt lõi của nó với nhân. Nguồn bao gồm hơn 20 nhân để lựa chọn, mặc dù nhân Gaussian sẽ là thích hợp nhất trong hầu hết các ứng dụng.

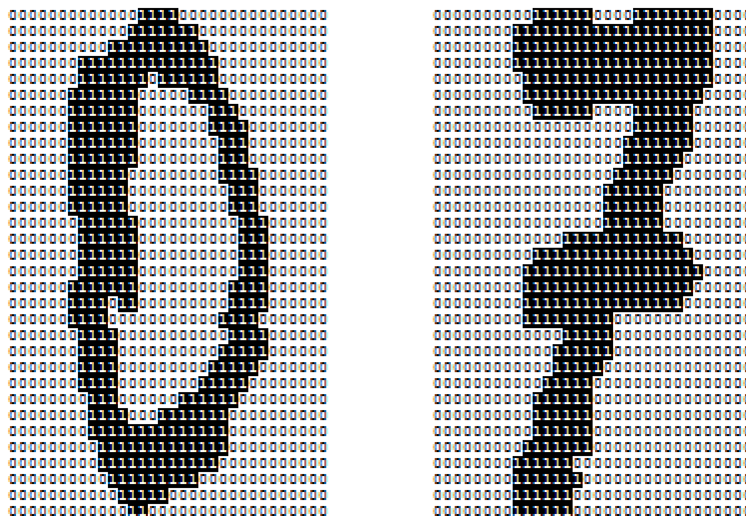
2.5. Nhận dạng chữ số

2.5.1. Dữ liệu số của UCI

Các kho dữ liệu Máy Học UCI (UCI Machine Learning Repository) là tập hợp các cơ sở dữ liệu, lý thuyết miền, và máy xây dựng dữ liệu được sử dụng bởi các máy học tập cộng đồng cho việc phân tích thực nghiệm của các thuật toán học máy. Một trong những kho dữ liệu có sẵn là Nhận dạng chữ số viết tay.

Trong dữ liệu chữ số, chữ số được biểu diễn là các ma trận 32x32. Chúng cũng có sẵn trong một hình thức tiền xử lý, trong đó chữ số đã được chia thành các khối không chồng khớp nhau 4x4 và số lượng các điểm ảnh trên đã được tính trong mỗi khối, tạo ra ma trận đầu vào 8x8 mà mỗi phần tử là một số nguyên trong khoảng 0 .. 16.

Đa chiều giảm là một bước rất cần thiết nếu chúng ta sẽ sử dụng phân lớp được tạo bởi Con trỏ của đa chiều (Curse of Dimensionality). Phương pháp nhân nói chung không gặp trở ngại trong việc xử lý các vấn đề về đa chiều lớn bởi vì chúng không bị giới hạn.



Mẫu chữ số được lấy ra từ kho số liệu chữ số thô.

Phương pháp nhân được sử dụng nhiều vì chúng có thể được áp dụng trực tiếp cho các vấn đề yêu cầu suy nghĩ kỹ càng dựa trên dữ liệu tiền xử lý và kiến thức rộng về cấu trúc của dữ liệu đang được mô hình hóa. Thậm chí nếu chúng ta biết rất ít về các dữ liệu, một ứng dụng của phương pháp nhân mù thường thấy kết quả khá đúng. Đạt được tối ưu hóa bằng cách sử dụng các phương pháp nhân có thể đươc, tuy nhiên, rất khó vì chúng ta có một sự lựa chọn vô hạn các hàm nhân và với mỗi hàm nhân có một không gian vô hạn để tinh chỉnh thông số.

Các mã nguồn sau cho ta thấy một ví dụ cụ thể về KDA. Lưu ý xử lý thế nào với đầu vào là các véc tơ đầy đủ cho 1024 vị trí. Điều này sẽ là không thực tế nếu chúng ta sẽ sử dụng Neural Networks, ví dụ:

```
/ / Giải nén đầu vào và đầu ra
 int samples = 500 ;
 double [,] input = new double [samples, 1024 ];
 int [] output = new int [samples];
    ... ...
/ / Tạo các lựa chọn nhân với các thông số đã cho
```

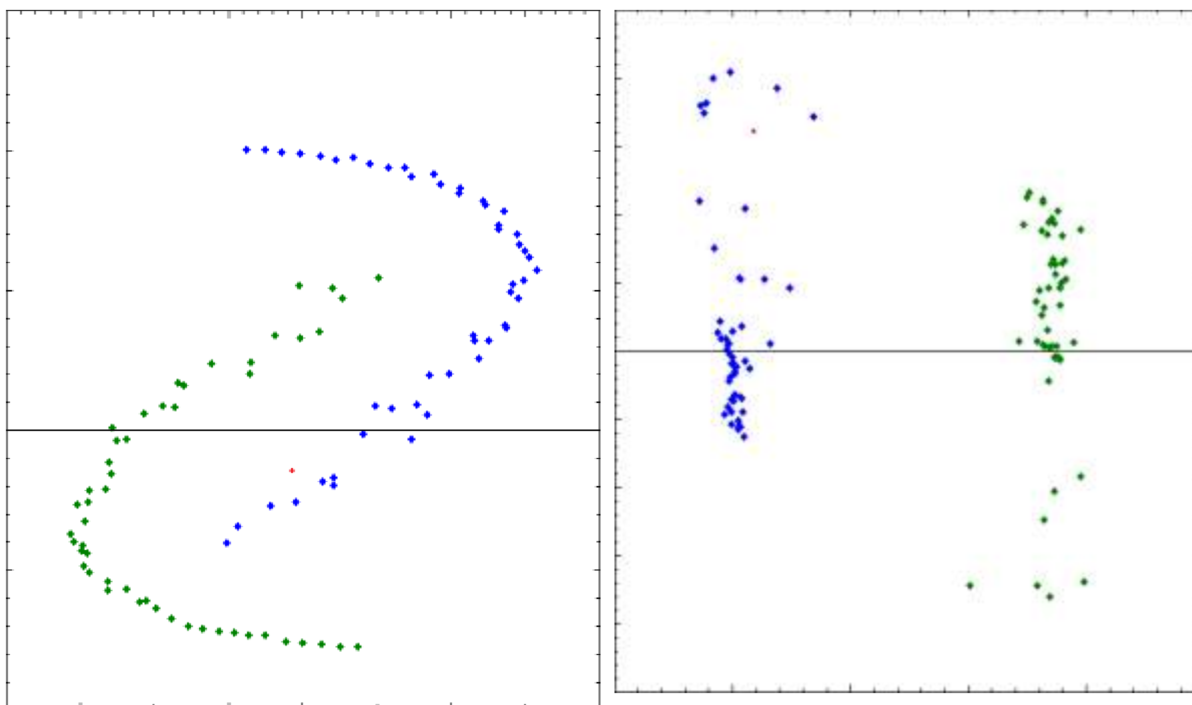
```
    IKernel kernel = new Gaussian(( double )numSigma.Value);
// Tạo các nhân phân tích biệt thức bằng cách sử dụng các nhân được lựa chọn
    kda = new KernelDiscriminantAnalysis(input, output, kernel);
// Thiết lập các ngưỡng tỷ lệ tối thiểu để giữ cho các thành phần trong phân tích
        kda.Threshold = ( double )numThreshold.Value;
// Thiết lập các qui chuẩn so sánh để tránh giải pháp đơn lẻ
        kda.Regularization = ( double )numRegularization.Value;
// Tính toán phân tích.
        kda.Compute();
// Hiển thị thông tin về các dạng phân tích
// (Hầu hết các thuộc tính có thể được databound trực để điều khiển trực quan)
       dgvPrincipalComponents.DataSource = kda.Discriminants;
       dgvFeatureVectors.DataSource = new
       ArrayDataView(kda.DiscriminantMatrix);
       dgvClasses.DataSource = kda.Classes;
```

2.5.2 Phân lớp các chữ số số bằng KDA.

Như đã đề cập trước đó, Nhân Phân tích biệt thức không bị làm xấu đi sau khi áp dụng đa chiều. Điều này có nghĩa nó có thể được áp dụng cho các tập dữ liệu số nguyên 32x32 mà không cần bất kỳ loại tiền xử lý. Việc xử lý chỉ cần thiết sẽ là chuyển vị của ma trận 32x32 vào vector có độ dài 1024.

Tuy nhiên, KDA không phải là một phương pháp rất hiệu quả, và nó cũng không phải là phương pháp phù hợp đối với một tập dữ liệu lớn. Cho dù có sự gia tăng kích thước song nó vẫn ít ảnh hưởng (thậm chí là không ảnh hưởng) tới thời gian phân tích, độ phức tạp KDA là O (n^3) với số lượng mẫu. Bên cạnh đó, các giải pháp của nó không ít ỏi như là trường hợp với hỗ trợ Vector Machines (SVMs). Điều này có nghĩa một số lượng đáng kể không gian bộ nhớ sẽ được sử dụng để chứa các Ma trận nhân đầy đủ trong quá trình phân loại.

Việc phân lớp sử dụng KDA thường được thực hiện bằng cách xem xét khoảng cách tối thiểu giữa một dữ liệu điểm dự kiến tới không gian đặc trưng và các lớp chỉ xét trong không gian đặc trưng. Chúng ta có thể nhìn thấy mục tiêu đằng sau phương pháp này trong ví dụ sau đây.



Ví dụ vấn đề phân lớp Yin Yang. Hình ảnh bên trái cho thấy hình ảnh ban đầu trong không gian đầu vào, hình ảnh bên trái cho thấy kết quả một nhân Phân tích biệt thức biểu diễn bởi một nhân Gaussian với sigma thiết lập như là 1,0.

Các dấu chấm màu đỏ trong hình ảnh bên phải đánh dấu phép chiếu của dấu chấm màu đỏ của hình ảnh trái. Chú ý dấu chấm là gần với lớp màu xanh trong cả hai hình ảnh. Tuy nhiên trong phân tích không gian đặc trưng, các lớp đã được trải ra thành nhiều tuyến tính. Trong trường hợp này, khoảng cách Euclide (hoặc tương đương là khoảng cách Malahanobis) đến lớp giá trị trung bình trong không gian đặc trưng sẽ trở thành một khoảng cách hợp lý giữa các lớp gần.

Các dòng mã sau đây chứng minh việc phân lớp của các trường hợp mới sử dụng một nhân đã được tính toán phân tích biệt thức.

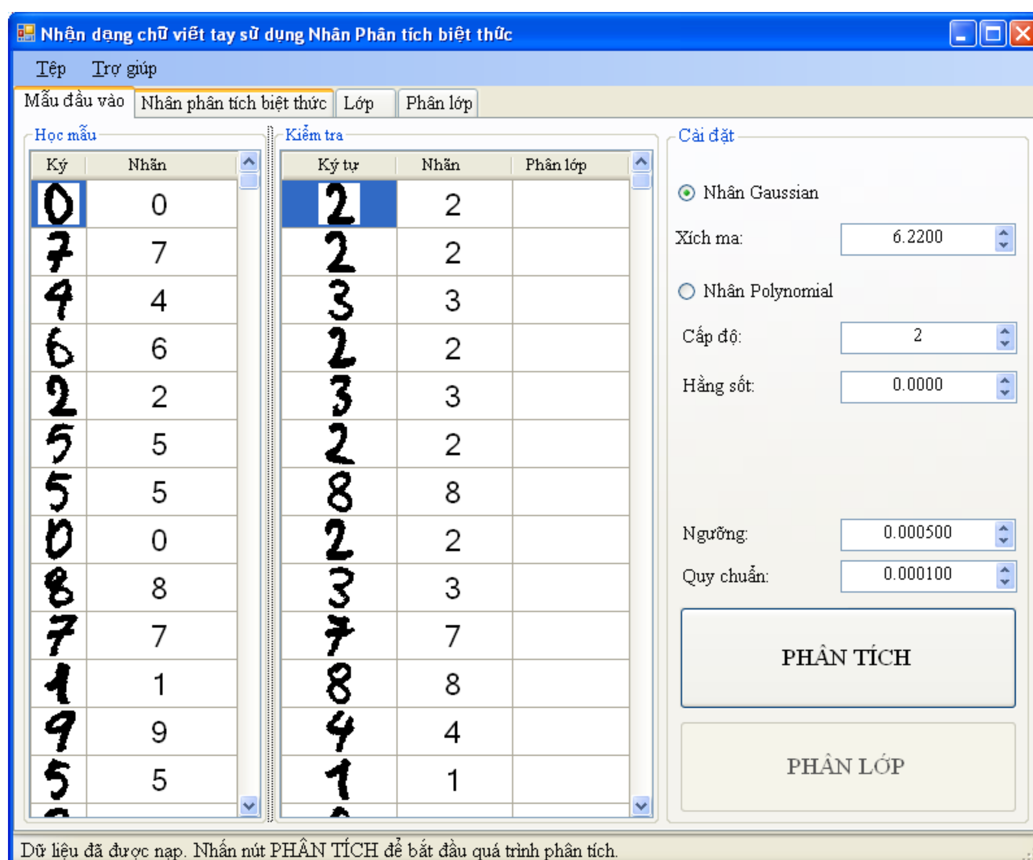
```
// Lấy các vector đầu vào
 double [] input = canvas.GetDigit();

// Phân loại các vector đầu vào
 int num = kda.Classify(input);

// Thiết lập câu trả lời phân loại thực tế
lbClassification.Text = num.ToString();
```

CHƯƠNG III
CHƯƠNG TRÌNH THỬ NGHIỆM

3.1. Kiểm tra ứng dụng

3.1.1 Phân tích

Việc chương trình thử nghiệm đi kèm với mã nguồn sẽ thể hiện rõ việc nhận dạng các chữ số viết tay bằng cách sử dụng KDA. Đầu tiên, chúng ta sẽ chạy ứng dụng, tốt nhất là chạy chương trình không thông qua Visual Studio 2008. Nhắp chuột vào menu Tệp và chọn Mở. Thao tác này sẽ tải một số mục từ kho dữ liệu chữ số vào ứng dụng.

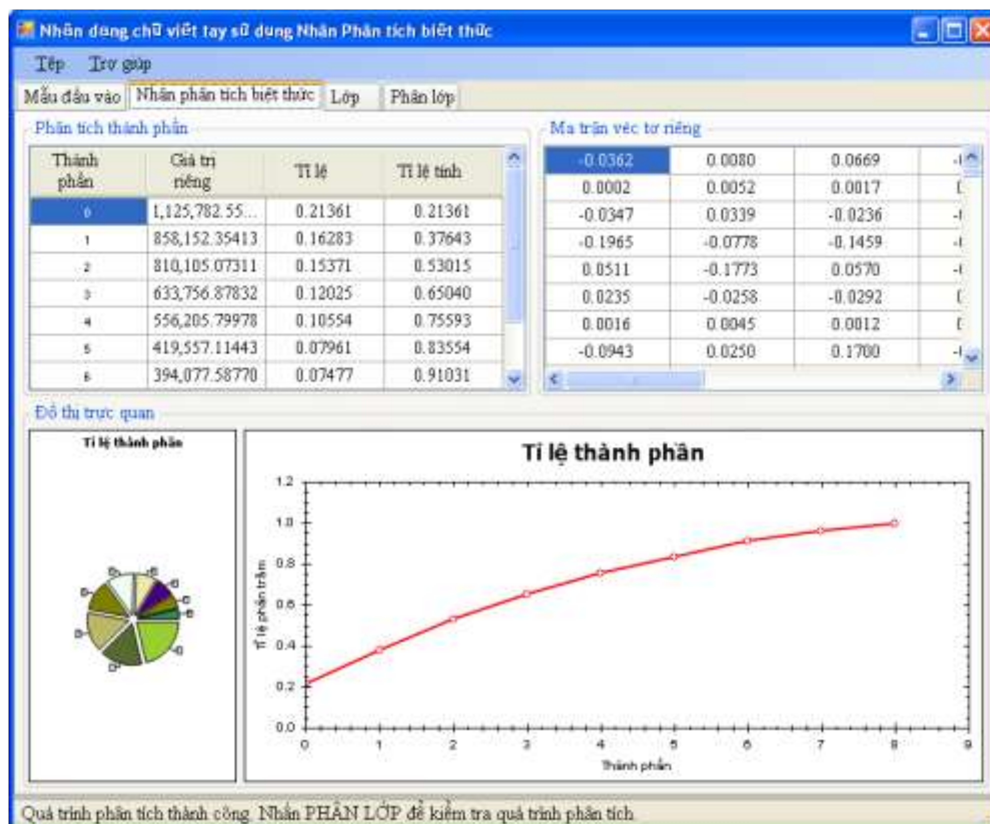


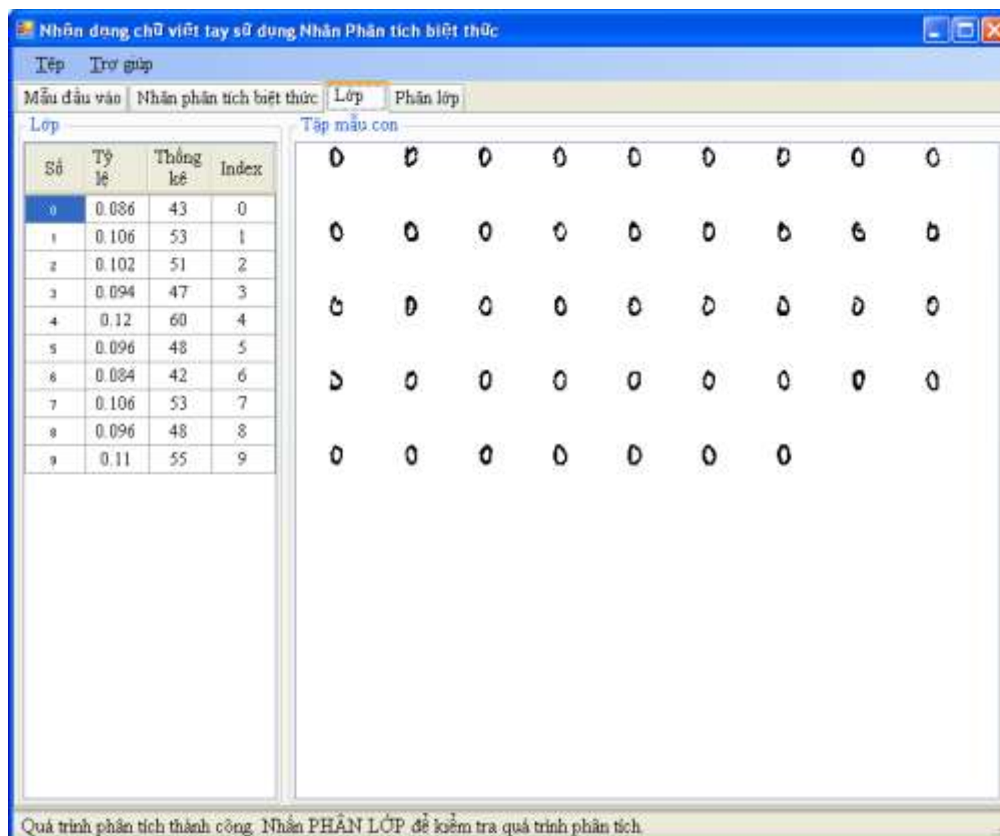
Dữ liệu chữ số được tải vào ứng dụng

Bên phải: Chi tiết thông tin về các lớp riêng biệt của các chữ số viết tay.

Để thực hiện việc phân tích, nhấp vào nút PHÂN TÍCH. Quá trình phân tích được thực hiện, sau khi phân tích xong, các tab khác trong việc ứng dụng mẫu sẽ được đưa ra thông tin phân tích. Mức độ quan trọng của mỗi yếu tố được

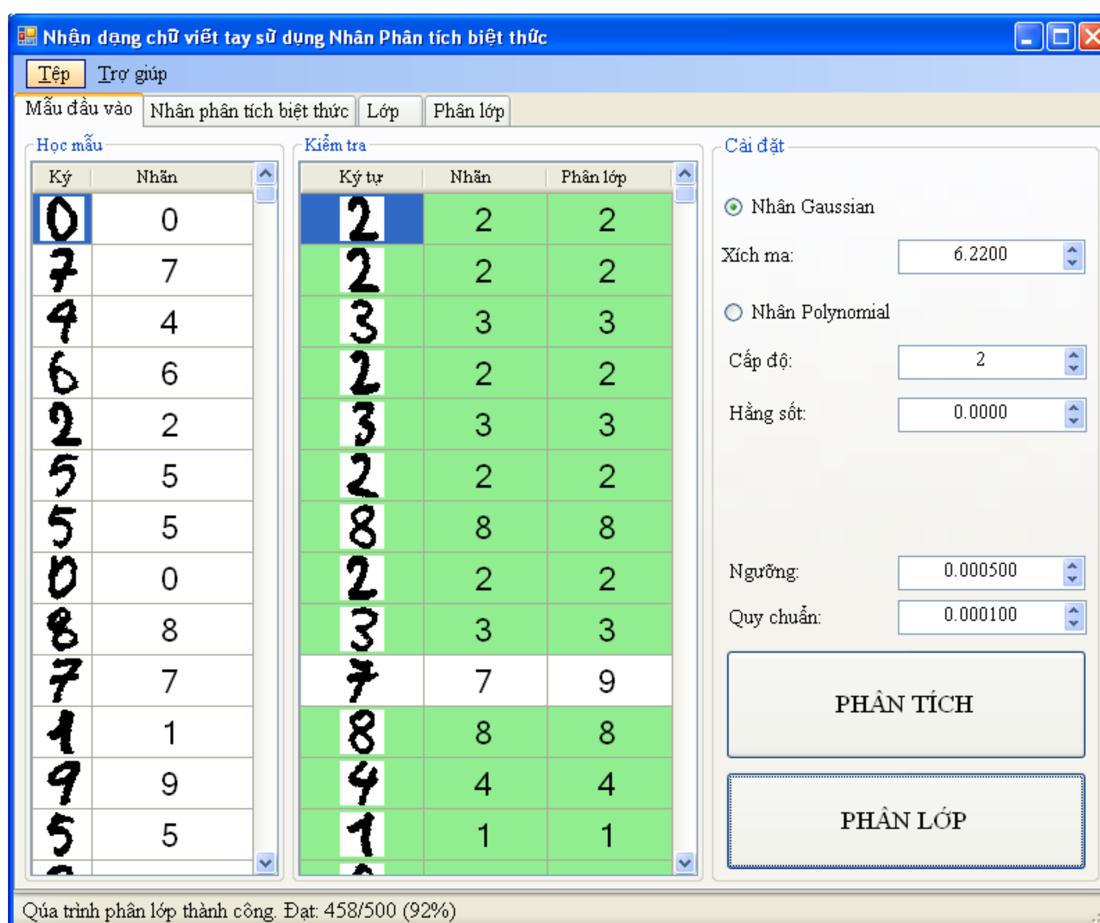
đưa ra trong phân tích biệt thức được biểu diễn trong một đồ thị hình tròn để kiểm tra dễ dàng.





Các yếu tố được tìm thấy trong Phân tích biệt thức và mối liên quan tầm quan trọng của chúng. Từ không gian đầu vào kích thước ban đầu kích thước 1024, chỉ có 9 là quan trong để được chọn cho việc phân lớp.

Sau khi phân tích xong, chúng ta có thể thử nghiệm khả năng phân lớp của nó trong việc kiểm tra kho dữ liệu. Các hàng cây xanh đã được xác định chính xác bởi khoảng cách biệt thức phân loại không gian Euclide. Chúng ta có thể nhìn thấy nó nhận dạng chính xác 92% các dữ liệu thử nghiệm. Các thử nghiệm này và dữ liệu đào tạo được phân chia và độc lập.

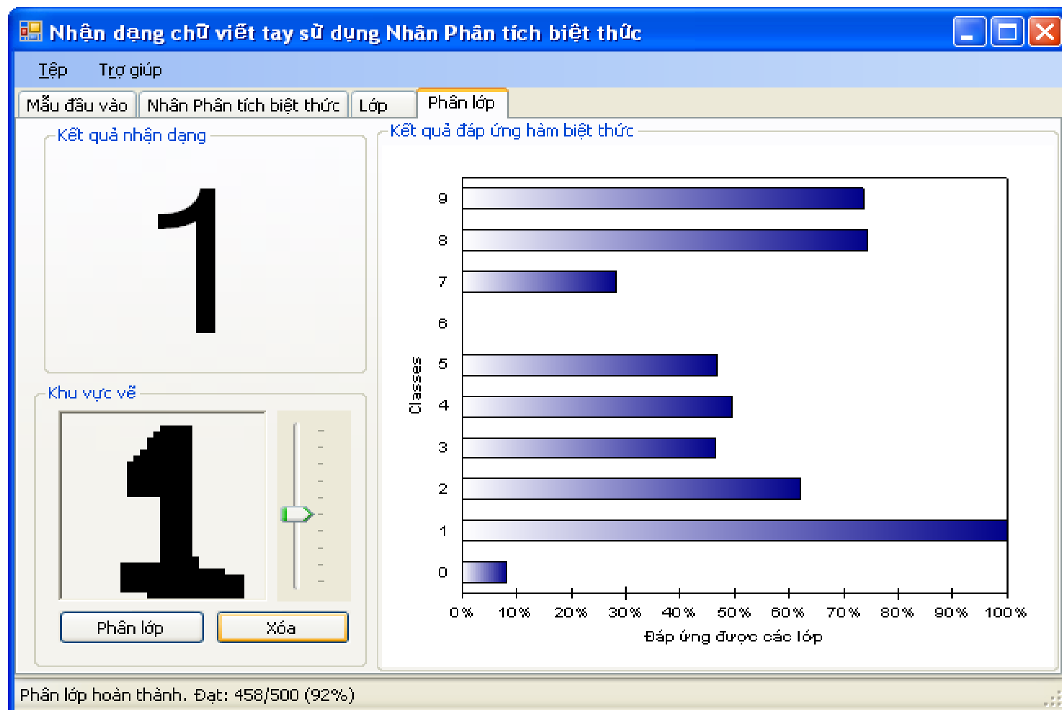


Sử dụng các giá trị mặc định trong ứng dụng, chương trình đạt được độ chính xác 92%.

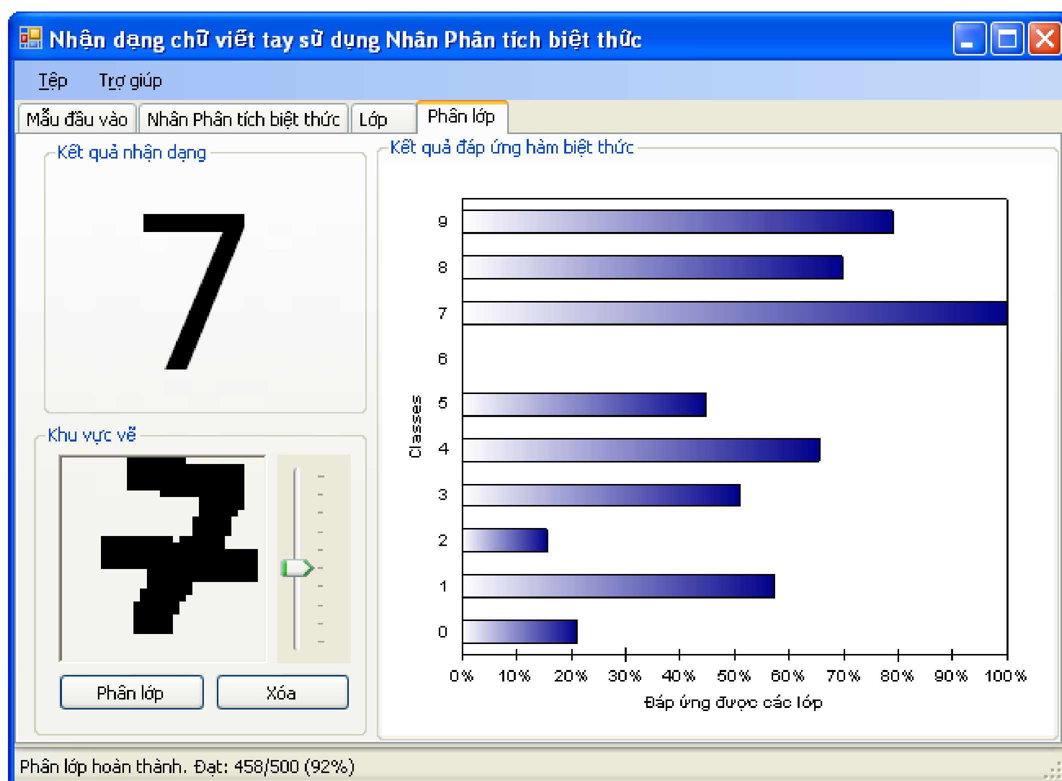
3.1.2 Kết quả

Sau khi phân tích đã được hoàn thành và xác nhận, chúng ta có thể sử dụng chương trình để phân loại các con số mới được viết trực tiếp trong ứng dụng. Các biểu đồ cột trên bên phải thể hiện kết quả tương đối cho từng công

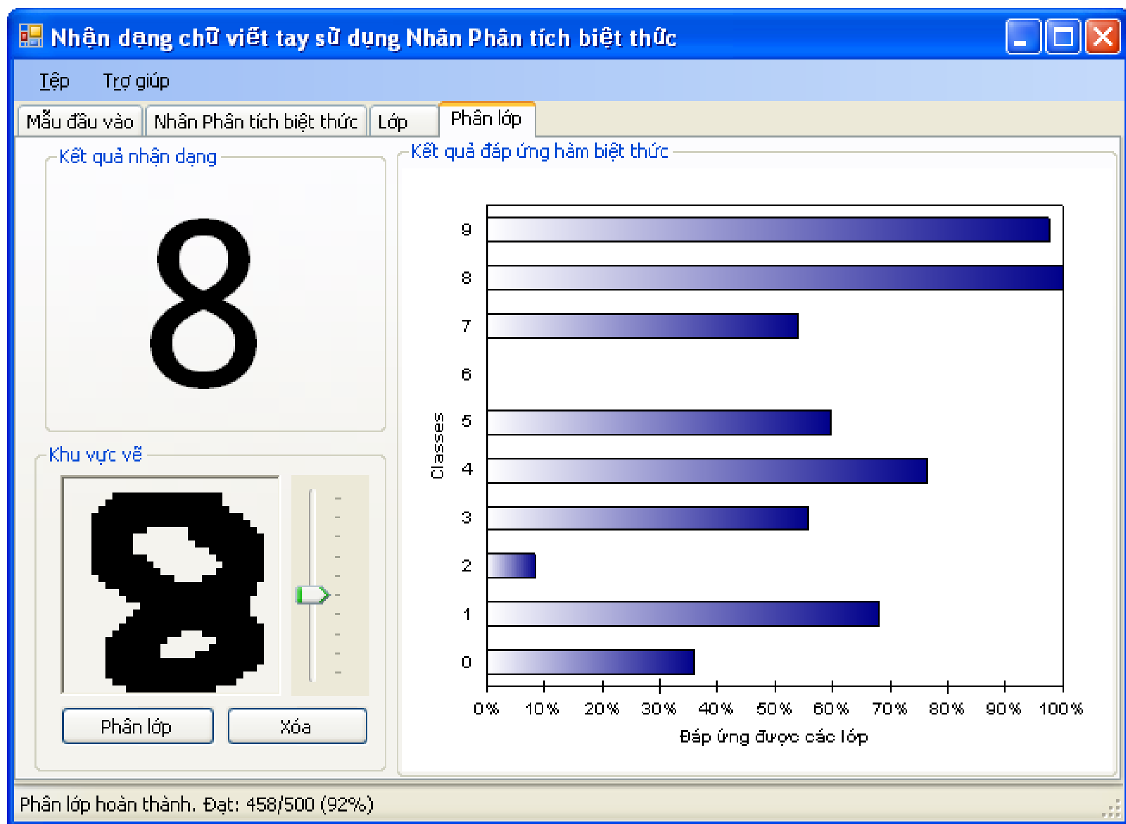
thức biệt thức. Mỗi lớp có một công thức biệt thức rằng kết quả đầu ra một phương pháp gần nhất cho mỗi điểm đầu vào. Việc phân loại này dựa trên công thức để đầu ra là tối đa.



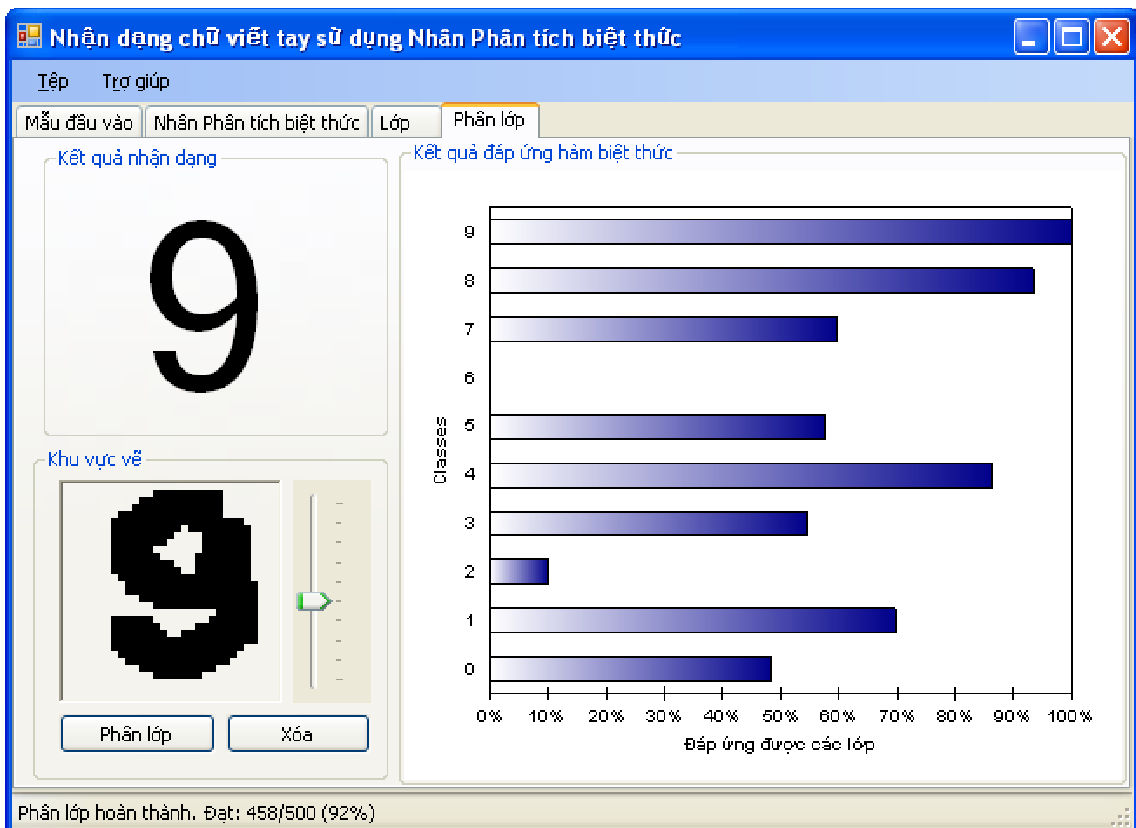
Nhận dạng số 1



Nhận dạng số 7



Nhận dạng số 8



Nhận dạng số 9

3.2 Mã lệnh trong chương trình viết cho một số các class

3.2.1 Class Linear Discrimnant Analysis

```
using System;
using System.Collections.ObjectModel;
using System.Linq;
using Accord.Math;
using Accord.Math.Decompositions;

namespace Accord.Statistics.Analysis
{
    public class LinearDiscriminantAnalysis
    {
        private int dimension;
        private int samples;
        private int classes;

        private double[] totalMeans;
        private double[] totalStdDevs;

        private int[] classCount;
        private double[][] classMeans;
        private double[][] classStdDevs;
        private double[][,] classScatter;
        private double[][] classMeansTransformed;

        private double[,] eigenVectors;
        private double[] eigenValues;
        private double[] bias;

        private double[,] result;
        private double[,] source;
        private int[] outputs;
```

```
double[,] Sw, Sb, St; // Scatter matrices

private double[] discriminantProportions;
private double[] discriminantCumulative;

DiscriminantCollection discriminantCollection;
DiscriminantAnalysisClassCollection classCollection;
//-------------------------------------------
```

Phần kiến tạo

```
public LinearDiscriminantAnalysis(double[,] inputs, int[] output)
{
   // Lấy số từ các class
   int startingClass = output.Min();
   this.classes = output.Max() - startingClass + 1;

   // Lưu trữ các dữ liệu gốc
   this.source = inputs;
   this.outputs = output;
   this.samples = inputs.GetLength(0);
   this.dimension = inputs.GetLength(1);

   // Tạo các cấu trúc đơn để lưu thong tin về sau
   this.classCount = new int[classes];
   this.classMeans = new double[classes][];
   this.classStdDevs = new double[classes][];
   this.classScatter = new double[classes][,];

   // Tạo cấu trúc hướng đối tượng để giữ các thong tin về class
  DiscriminantAnalysisClass[] collection = new
 DiscriminantAnalysisClass[classes];
   for (int i = 0; i < classes; i++)
    collection[i] = new DiscriminantAnalysisClass(this, i, startingClass + i);
```

```
    this.classCollection = new
  DiscriminantAnalysisClassCollection(collection);
}
# kết thúc phần

//--------------------------------------------
```

```
#  Phần thuộc tính
      /// Quay lại nguồn dữ lieu cung cấp ban đầu để phân tích
    public double[,] Source
    {
      get { return this.source; }
    }

    public double[,] Result
    {
      get { return this.result; }
    }

    public int[] Classifications
    {
      get { return this.outputs; }
    }

    /// Lấy giá trị trung bình của dữ liệu nguồn để đưa phướng án
    public double[] Means
    {
      get { return totalMeans; }
      protected set { totalMeans = value; }
    }

    /// Lấy trung bình tiêu chuẩn của các dữ liệu ban đầu cho phương pháp
  public double[] StandardDeviations
    {
      get { return totalStdDevs; }
```

```
    protected set { totalStdDevs = value; }
}

/// sử dụng ma trận lớp trong cho dữ liệu
public double[,] ScatterWithinClass
{
    get { return Sw; }
    protected set { Sw = value; }
}

///  sử dụng ma trận lớp giữa cho dữ liệu
public double[,] ScatterBetweenClass
{
    get { return Sb; }
    protected set { Sb = value; }
}
/// Lấy tổng ma trận lớp cho dữ liệu
public double[,] ScatterMatrix
{
    get { return St; }
    protected set { St = value; }
}

public double[,] DiscriminantMatrix
{
    get { return eigenVectors; }
    protected set { eigenVectors = value; }
}

public double[] Eigenvalues
{
    get { return eigenValues; }
    protected set { eigenValues = value; }
}
```

```
/// Sử dụng các lớp trong mỗi biệt thức
/// trong không gian biệt thức
public double[] Proportions
{
   get { return discriminantProportions; }
}

public double[] CumulativeProportions
{
   get { return discriminantCumulative; }
}

public DiscriminantCollection Discriminants
{
   get { return discriminantCollection; }
}

public DiscriminantAnalysisClassCollection Classes
{
   get { return classCollection; }
}

public double[][,] ClassScatter
{
   get { return classScatter; }
}

public double[][] ClassMeans
{
   get { return classMeans; }
}

public double[][] ClassStandardDeviations
```

```
{
    get { return classStdDevs; }
}

public int[] ClassCount
{
    get { return classCount; }
}
# Kết thúc phần

//--------------------------------------------

#  Phần khai báo phương thức
public virtual void Compute()
{
    // Tính toán toàn bộ dữ liệu thiết lập các biện pháp
    Means = Tools.Mean(source);
    StandardDeviations = Tools.StandardDeviation(source, totalMeans);
    double total = dimension;

    // Khởi tạo các ma trận tán xạ
    this.Sw = new double[dimension, dimension];
    this.Sb = new double[dimension, dimension];

    // Cho mỗi lớp
    for (int c = 0; c < Classes.Count; c++)
    {
        // Lấy các lớp con…
        double[,] subset = Classes[c].Subset;
        int count = subset.GetLength(0);

        // Lấy lớp trung bình
        double[] mean = Tools.Mean(subset);
```

```
    // Tiếp tục xây dựng Ma trận tán xạ lớp trong
    double[,] Swi = Tools.Scatter(subset, mean, (double)count);

    // Sw = Sw + Swi
    for (int i = 0; i < dimension; i++)
        for (int j = 0; j < dimension; j++)
            Sw[i, j] += Swi[i, j];


    // Tiếp tục xây dựng Ma trận tán xạ lớp giữa
    double[] d = mean.Subtract(totalMeans);
    double[,] Sbi = d.Multiply(d.Transpose()).Multiply(total);

    // Sb = Sb + Sbi
    for (int i = 0; i < dimension; i++)
        for (int j = 0; j < dimension; j++)
            Sb[i, j] += Sbi[i, j];

    // Lưu các thông tin được thêm
    this.classScatter[c] = Swi;
    this.classCount[c] = count;
    this.classMeans[c] = mean;
    this.classStdDevs[c] = Tools.StandardDeviation(subset, mean);
}

// Tính tóan sự phân ly của giá trị riêng
      EigenvalueDecomposition evd = new
      EigenvalueDecomposition(Matrix.Inverse(Sw).Multiply(Sb));

// Lấy các giá trị riêng, tương ứng với các vecto riêng
      double[] evals = evd.RealEigenvalues;
      double[,] eigs = evd.Eigenvectors;

// Sắp xếp các giá treiSort eigen values and vectors in ascending order
```

```
            eigs = Matrix.Sort(evals, eigs, new
            GeneralComparer(ComparerDirection.Descending, true));

        // Lưu trữ thông tin
            this.Eigenvalues = evals;
            this.DiscriminantMatrix = eigs;

        // Tạo hàm biệt thức độ xiên
            bias = new double[classes];
            for (int i = 0; i < classes; i++)
            {
                bias[i] = (-0.5).Multiply(classMeans[i]).Multiply(
                eigs.Multiply(classMeans[i])) +
                System.Math.Log(classCount[i] / total);
            }

        // Tạo các phép chiếu
        this.result = new double[dimension, dimension];
        for (int i = 0; i < dimension; i++)
          for (int j = 0; j < dimension; j++)
            for (int k = 0; k < dimension; k++)
              result[i, j] += source[i, k] * eigenVectors[k, j];
        createDiscriminants();
    }
    public double[,] Transform(double[,] data)
    {
        return Transform(data, discriminantCollection.Count);
    }
    public virtual double[,] Transform(double[,] data, int discriminants)
    {
        int rows = data.GetLength(0);
        int cols = data.GetLength(1);

        double[,] r = new double[rows, discriminants];
```

```
    // Cộng dữ liệu ma trận bằng cách chọn các véctơ riêng
    for (int i = 0; i < rows; i++)
        for (int j = 0; j < discriminants; j++)
            for (int k = 0; k < cols; k++)
                r[i, j] += data[i, k] * eigenVectors[k, j];

    return r;
}
public double[] Transform(double[] data)
{
    return Transform(data.ToMatrix()).GetRow(0);
}
public double[] Transform(double[] data, int discriminants)
{
    return Transform(data.ToMatrix(),discriminants).GetRow(0);
}
public int GetNumberOfDimensions(float threshold)
{
    if (threshold < 0 || threshold > 1.0)
        throw new ArgumentException("Threshold should be a value between
0 and 1", "threshold");

    for (int i = 0; i < discriminantCumulative.Length; i++)
    {
        if (discriminantCumulative[i] >= threshold)
            return i + 1;
    }

    return discriminantCumulative.Length;
}
public int Classify(double[] input)
{
    double[] projection = Transform(input);
```

```
    // Chọn lớp với hàm biệt thức cao hơn
    int imax = 0;
    double max = discriminantFunction(0, projection);
    for (int i = 1; i < classCollection.Count; i++)
    {
        double fy = discriminantFunction(i, projection);
        if (fy > max)
        {
            max = fy;
            imax = i;
        }
    }

    return classCollection[imax].Number;
}
public int Classify(double[] input, out double[] responses)
{
    double[] projection = Transform(input);
    responses = new double[classCollection.Count];

   int imax = 0;
    double max = discriminantFunction(0, projection);
    responses[0] = max;
    for (int i = 1; i < classCollection.Count; i++)
    {
        double fy = discriminantFunction(i, projection);
        responses[i] = fy;
        if (fy > max)
        {
            max = fy;
            imax = i;
        }
    }
```

```
    return classCollection[imax].Number;
}
public int[] Classify(double[][] inputs)
{
    int[] output = new int[inputs.Length];
    for (int i = 0; i < inputs.Length; i++)
        output[i] = Classify(inputs[i]);
    return output;
}
internal virtual double discriminantFunction(int c, double[] projection)
{
    return classMeans[c].Multiply(projection) + bias[c];
}
# Kết thúc phần

//--------------------------------------------
#  Phần bảo vệ phương thức
protected void createDiscriminants()
{
    int numDiscriminants = eigenValues.Length;
    discriminantProportions = new double[numDiscriminants];
    discriminantCumulative = new double[numDiscriminants];

    // Tính toán tổng ma trận phân tán
    int size = Sw.GetLength(0);
    St = new double[size, size];
    for (int i = 0; i < size; i++)
        for (int j = 0; j < size; j++)
            St[i, j] = Sw[i, j] + Sb[i, j];

    // Tính toán tỉ lệ
    double sum = 0.0;
    for (int i = 0; i < numDiscriminants; i++)
```

```
            sum += System.Math.Abs(eigenValues[i]);
        sum = (sum == 0) ? 0 : (1.0 / sum);

        for (int i = 0; i < numDiscriminants; i++)
            discriminantProportions[i] = System.Math.Abs(eigenValues[i]) * sum;

        // Tính toán tỉ lệ dồn lại
        this.discriminantCumulative[0] = this.discriminantProportions[0];
        for (int i = 1; i < this.discriminantCumulative.Length; i++)
            this.discriminantCumulative[i] = this.discriminantCumulative[i - 1] +
this.discriminantProportions[i];

        // Taọ cấu trúc hướng đối tượng để giữ biệt thức tuyến tính
        Discriminant[] discriminants = new Discriminant[numDiscriminants];
        for (int i = 0; i < numDiscriminants; i++)
            discriminants[i] = new Discriminant(this, i);
                this.discriminantCollection = new
       DiscriminantCollection(discriminants);
      }
      #Kết thúc phần
   }

   #  Phần hỗ trợ các lớp
   public class DiscriminantAnalysisClass
   {
      private LinearDiscriminantAnalysis analysis;
      private int classNumber;
      private int index;

      internal DiscriminantAnalysisClass(LinearDiscriminantAnalysis analysis,
int index, int classNumber)
      {
         this.analysis = analysis;
         this.index = index;
```

```
    this.classNumber = classNumber;
}
public int Index
{
    get { return index; }
}
public int Number
{
    get { return classNumber; }
}
public double Prevalence
{
    get { return (double)Count / analysis.Source.GetLength(0); }
}

/// Lấy lớp vecto trung bình
public double[] Mean
{
    get { return analysis.ClassMeans[index]; }
}

///   Lấy lớp vec tơ độ lệch quần phương
public double[] StandardDeviation
{
    get { return analysis.ClassStandardDeviations[index]; }
}

///  Lấy ma trận phân tán cho lớp này
public double[,] Scatter
{
    get { return analysis.ClassScatter[index]; }
}
public int[] Indexes
{
```

```
            get { return Matrix.Find(analysis.Classifications, y => y ==
classNumber); }
        }
        public double[,] Subset
        {
            get
            {
                return analysis.Source.Submatrix(Indexes);
            }
        }

        public int Count
        {
            get { return analysis.ClassCount[index]; }
        }
        ///   Hàm biệt thức cho lớp
        public double DiscriminantFunction(double[] projection)
        {
            //return Mean.Multiply(projection) + Bias[index];
            return analysis.discriminantFunction(index, projection);
        }
    }
    public class Discriminant
    {
        private LinearDiscriminantAnalysis analysis;
        private int index;
        internal Discriminant(LinearDiscriminantAnalysis analysis, int index)
        {
            this.analysis = analysis;
            this.index = index;
        }
        public int Index
        {
            get { return index; }
```

```
        }
        public double[] Eigenvector
        {
            get { return analysis.DiscriminantMatrix.GetColumn(index); }
        }
        public double Eigenvalue
        {
            get { return analysis.Eigenvalues[index]; }
        }
        public double Proportion
        {
            get { return analysis.Proportions[index]; }
        }
        public double CumulativeProportion
        {
            get { return analysis.CumulativeProportions[index]; }
        }

    }
    public class DiscriminantCollection : ReadOnlyCollection<Discriminant>
    {
        internal DiscriminantCollection(Discriminant[] components)
            : base(components)
        {
        }
    }
    public class DiscriminantAnalysisClassCollection :
ReadOnlyCollection<DiscriminantAnalysisClass>
    {
        internal DiscriminantAnalysisClassCollection(DiscriminantAnalysisClass[]
components)
            : base(components)
        {
        }
```

```
    }
    # Kết thúc phần
}
```

3.2.2 Kernel Discrimnant Analysis

```
using Accord.Math;
using Accord.Math.Decompositions;
using Accord.Statistics.Kernels;
using System.Collections.Generic;

namespace Accord.Statistics.Analysis
{

    public class KernelDiscriminantAnalysis : LinearDiscriminantAnalysis
    {
        private IKernel kernel;
        private double regularization = 0.0001;
        private double threshold = 0.001;
        private double[][] kernelClassMeans;

        //--------------------------------------------



        #  Phần kiến thiết tạo
        public KernelDiscriminantAnalysis(double[,] inputs, int[] output, IKernel
kernel)
            : base(inputs, output)
        {
            this.kernel = kernel;
            this.kernelClassMeans = new double[Classes.Count][];
        }
        # Kết thúc phần
```

```
//--------------------------------------------

#  Phần khai báo thuộc tính
public IKernel Kernel
{
   get { return kernel; }
}

public double Regularization
{
   get { return regularization; }
   set { regularization = value; }
}

public double Threshold
{
   get { return threshold; }
   set { threshold = value; }
}
# Kết thúc phần

//--------------------------------------------

# Phần khai báo Phương thức
  ///   Tính toán thuật toán KDA nhiều lớp
  public override void Compute()
{
   // Lấy mốt số thông tin ban đầu
   int dimension = Source.GetLength(0);
   double[,] source = Source;
   double total = dimension;

   // Tạo ma trận Gram (Kernel)
   double[,] K = new double[dimension, dimension];
```

```
for (int i = 0; i < dimension; i++)
{
    for (int j = i; j < dimension; j++)
    {
        double s = kernel.Function(source.GetRow(i), source.GetRow(j));
        K[i, j] = s;
        K[j, i] = s;
    }
}
// Tính toán toàn bộ dữ liệu
base.Means = Tools.Mean(K);
base.StandardDeviations = Tools.StandardDeviation(K, Means);

// Gán giá trị ban đầu cho ma trận phân tán tương tự nhân
double[,] Sb = new double[dimension, dimension];
double[,] Sw = new double[dimension, dimension];

// Cho mỗi lớp
for (int c = 0; c < Classes.Count; c++)
{
    // Lấy lớp con ma trận nhân
    double[,] Kc = K.Submatrix(Classes[c].Indexes);
    int count = Kc.GetLength(0);

    // Lấy trung bình lớp ma trận Nhân
    double[] mean = Tools.Mean(Kc);

    // Đặt ma trận tương đương của ma trận phân tán lớp trong
    double[,] Swi = Tools.Scatter(Kc, mean, (double)count);

    // Sw = Sw + Swi
    for (int i = 0; i < dimension; i++)
        for (int j = 0; j < dimension; j++)
            Sw[i, j] += Swi[i, j];
```

```
        // Đặt ma trận tương đương của ma trận phân tán lớp giữa
        double[] d = mean.Subtract(base.Means);
        double[,] Sbi = d.Multiply(d.Transpose()).Multiply(total);

        // Sb = Sb + Sbi
        for (int i = 0; i < dimension; i++)
            for (int j = 0; j < dimension; j++)
                Sb[i, j] += Sbi[i, j];

        // Lưu trữ dữ liệu thêm vào
        base.ClassScatter[c] = Swi;
        base.ClassCount[c] = count;
        base.ClassMeans[c] = mean;
        base.ClassStandardDeviations[c] = Tools.StandardDeviation(Kc,
mean);
    }

    // Thêm quy tắc
    for (int i = 0; i < dimension; i++)
        Sw[i, i] += regularization;

    // Phân tích sự phân tán của giá trị riêng
    double[,] C = Matrix.Inverse(Sw).Multiply(Sb);
    EigenvalueDecomposition evd = new EigenvalueDecomposition(C);

    // Lấy giá trị riêng và tương ứng với vecsto riêng
    double[] evals = evd.RealEigenvalues;
    double[,] eigs = evd.Eigenvectors;

    // Sắp xếp giá trị riêng và vecto riêng theo giá trị tăng dần
   eigs = Matrix.Sort(evals, eigs, new
   GeneralComparer(ComparerDirection.Descending, true));
```

```
if (threshold > 0)
{
   // Calculate proportions earlier
   double sum = 0.0;
   for (int i = 0; i < dimension; i++)
      sum += System.Math.Abs(evals[i]);

   if (sum > 0)
   {
      sum = 1.0 / sum;

      // Không lưu những thông tin không quan trọng
      int keep = 0; while (keep < dimension &&
      System.Math.Abs(evals[keep]) * sum > threshold) keep++;
      eigs = eigs.Submatrix(0, dimension - 1, 0, keep - 1);
      evals = evals.Submatrix(0, keep - 1);
   }
}

// Lưu trữ thông tin
base.Eigenvalues = evals;
base.DiscriminantMatrix = eigs;
base.ScatterBetweenClass = Sb;
base.ScatterWithinClass = Sw;

// Tính toán khoảng trống cho việc phân lớp cuối cùng
for (int c = 0; c < Classes.Count; c++)
{
   double[] mean = new double[eigs.GetLength(1)];
   for (int i = 0; i < eigs.GetLength(0); i++)
      for (int j = 0; j < eigs.GetLength(1); j++)
         mean[j] += ClassMeans[c][i] * eigs[i, j];
   kernelClassMeans[c] = mean;
```

```
        }
        // Tính toán các thông tin được thêm về việc phân tích và tạo
        //  cấu trúc hướng đối tượng để giữ các biệt thức tìm thấy
        createDiscriminants();
    }
    public override double[,] Transform(double[,] data, int discriminants)
    {
        // Lấy một vài thông tin
        int rows = data.GetLength(0);
        int cols = data.GetLength(1);
        int N = Source.GetLength(0);

        // Tạo ma trận nhân
        double[,] K = new double[rows, N];
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < N; j++)
                K[i, j] = kernel.Function(Source.GetRow(j), data.GetRow(i));

        // Xem xét khoảng trống biệt thức nhân
        double[,] result = new double[rows, discriminants];
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < discriminants; j++)
                for (int k = 0; k < N; k++)
                    result[i, j] += K[i, k] * DiscriminantMatrix[k, j];
        return result;
    }
    internal override double discriminantFunction(int i, double[] projection)
    {
        return -Distance.SquareEuclidean(projection, kernelClassMeans[i]);
    }
    # Kết thúc
  }
}
```

KẾT LUẬN

Đồ án tốt nghiệp đã tìm hiểu sử dụng KDA như thế nào để giải quyết trong bài toán nhận dạng. Tuy nhiên KDA gặp một số hạn chế: không hỗ trợ Vector Machines (SVMs), giải pháp của nó không trải đều. Quy mô không mở rộng với kích thước mẫu đầu vào là O (n^3) mặc dù đã xử lý được vấn đề đưa kích thước lớn vào vecto.

Dù vậy, KDA có một lợi thế, đó là tổng quát cho trường hợp đa lớp. Sự lựa chọn phù hợp các chức năng nhân (và điều chỉnh các thông số của nó), chi phí hoạt động ít tốn kém, cả về năng lượng xử lý và dữ liệu đào tạo sẵn có. Hiệu quả bởi bất kỳ giải pháp nào tìm ra cũng sẽ là giải pháp tốt nhất có thể cho các giá trị tham số được học (Ví dụ Neural Networks, trong đó người ta phải thử rất nhiều điểm đầu khác nhau và nhiều trường hợp thử nghiệm để đảm bảo kết quả nhất quán)

Đồ án cũng đã xây dựng và cài đặt thành công chương trình Nhận dạng chữ viết tay sử dụng Nhân Phân tích biệt thức với giao diện tiếng Việt. Xây dựng bộ cài đặt và chương trình riêng độc lập.

Hướng phát triển của đề tài: Sau khi đồ án được hoàn thành, chương trình sẽ được bổ xung thêm các chức năng trong việc nhận dạng chữ cái latinh, chữ cái tiếng Việt, phù hợp hơn so với việc chỉ nhận dạng chữ số như trong phần nêu trên.

TÀI LIỆU THAM KHẢO

Tài liệu tham khảo tiếng Việt

[1.] PGS.TS Nguyễn Năng Hoan, *Xử lý ảnh,* Học viện Bưu Chính viễn thông,

[2.] TS. Đỗ Năng Toàn, TS. Phạm Việt Bình, *Giáo trình Xử lý ảnh*

[3.] Đinh Mạnh Tường(2002), *Trí tuệ nhân tạo*, NXB Khoa học và Kỹ thuật

[4.] Nguyễn Hữu Tình, Lê Tấn Hùng, Phạm Ngọc Yến, Nguyễn Lan Hương(1999), *Cơ sở và ứng dụng Matlab*, NXB Khoa học và Kỹ thuật.

[5.] Trịnh Thế Tiến. Nguyễn Minh, *Các Cơ Sở Dữ Liệu Microsoft Visual C# 2008 - Lập Trình Căn Bản Và Nâng Cao*, NXB Hồng Đức

[6.] Lương Mạnh Bá, Nguyễn Thanh Thủy, *Nhập Môn Xử Lý Ảnh Số (Xuất Bản Lần Thứ 4 Có Chỉnh Lý Bổ Sung)*, NXB KHKT

Tài liệu tham khảo tiếng Anh

[7.] Oh S.H., Lee Yj(1995), *A modified error function to improve the error Back-Propagation algorithm for Multi-layer perceptrons*, ETRI Journal Vol 17, No 1.

[8.] Ooyen A. V., Nienhuis B(1992), *Improving the Convergence of the Back-Propagation algorithm*, Neural networks, Vol. 5, pp.465-471.

[9.] T. masters(1993), Practical neural network Recipes in C++, Academic Press, Inc.

[10.] Tom M. Mitchell(1997), Machine learning, The McGraw-Hill Companies, Inc.