**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**





**CHUYÊN ĐỀ TỐT NGHIỆP**

**TÌM HIỂU KỸ THUẬT HỌC MÁY RANDOM FOREST**

**ĐỂ LỌC EMAIL SPAM**

**Giảng viên hướng dẫn: TS NGUYỄN ĐỨC THUẦN**

**Sinh viên thực hiện: ĐINH THỊ KIM THOA**

**Mã số sinh viên: 58131418**

Khánh Hoà, Tháng 6/2020

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**





**CHUYÊN ĐỀ TỐT NGHIỆP**

**TÌM HIỂU KỸ THUẬT HỌC MÁY RANDOM FOREST**

**ĐỂ LỌC EMAIL SPAM**

**Giảng viên hướng dẫn: TS NGUYỄN ĐỨC THUẦN**

**Sinh viên thực hiện: ĐINH THỊ KIM THOA**

**Mã số sinh viên: 58131418**

Khánh Hoà, Tháng 6/2019

# LỜI CAM ĐOAN

Tôi xin cam đoan đây là công trình nghiên cứu của tôi và được sự hướng dẫn khoa học của TS. Nguyễn Đức Thuần. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được thu thập từ các nguồn khác nhau có ghi trong phần tài liệu tham khảo. Ngoài ra trong bài viết có sử dụng một số nhân xét, đánh giá của một số tác giả khác, cơ quan tổ chức đều được trích dẫn và có chú thích gốc.

Nếu phát hiện bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung bài viết của mình.

**Khánh Hoà, ngày 21 tháng 06 năm 2020**

Đinh Thị Kim Thoa

# LỜI CẢM ƠN

Trước hết tôi xin chân thành cảm ơn TS. Nguyễn Đức Thuần đã tận tình hướng dẫn, chỉ dạy tôi tìm ra hướng nghiên cứu, tìm kiếm tài liệu và đề xuất những hướng giải quyết mỗi khi tôi gặp khó khăn trong suốt quá trình thực hiện đề tài nhờ đó tôi có thể hoàn thành đề tài này.

Tôi cũng xin chân thành cảm ơn toàn thể quý Thầy Cô Khoa Công Nghệ Thông Tin Trường Đại Học Nha Trang đã tận tình giảng dạy, trang bị cho tôi những kiến thức cần thiêt trong suốt quá trình học tập tại trường.

Mặc dù đã cố gắng trong quá trình nghiên cứu tuy nhiên không tránh những tồn tại hạn chế và thiếu sót. Vì vậy tôi rất mong nhận được sự góp ý chân thành từ thầy cô nhằm bổ sung và hoàn thiện hơn.

Xin chân thành cảm ơn!

Sinh viên thực hiện

Đinh Thị Kim Thoa

**MỤC LỤC**

[LỜI CAM ĐOAN 3](#_Toc43729923)

[LỜI CẢM ƠN 4](#_Toc43729924)

[LỜI MỞ ĐẦU 8](#_Toc43729925)

[Chương 1: TỔNG QUAN 9](#_Toc43729926)

[1.1 Email Spam 9](#_Toc43729927)

[1.2 Kỹ thuật phân loại Email Spam 9](#_Toc43729928)

[1.3 Giới thiệu Random Forest 9](#_Toc43729929)

[Chương 2: CƠ SỞ LÝ THUYẾT 11](#_Toc43729930)

[2.1 Giới thiệu học máy 11](#_Toc43729931)

[2.2 Phân lớp dựa vào cây quyết định 11](#_Toc43729932)

[2.2.1 Xây dựng cây quyết định 12](#_Toc43729933)

[2.2.2 Chỉ số Gini (Gini index) 12](#_Toc43729934)

[2.3 Giới thiệu thuật toán Random Forest 13](#_Toc43729935)

[2.4 Bootstrap và bagging 14](#_Toc43729936)

[2.4.1 Bootstrap 14](#_Toc43729937)

[2.4.2 Bagging 14](#_Toc43729938)

[2.4.3 Tính năng quan trọng 15](#_Toc43729939)

[2.5 Thuật toán Random Forest 15](#_Toc43729940)

[2.6 Đặc điểm của Random Forest 16](#_Toc43729941)

[2.6.1. Ưu điểm 16](#_Toc43729942)

[2.6.2 Hạn chế 17](#_Toc43729943)

[2.6.3 Out-of-bag 17](#_Toc43729944)

[Chương 3: LỌC EMAIL SPAM BẰNG KỸ THUẬT RANDOM FOREST 18](#_Toc43729945)

[3.1 Bộ dữ liệu thí nghiệm 18](#_Toc43729946)

[3.2 Chương trình demo thuật toán Random Forest trên python 18](#_Toc43729947)

[3.3 Triển khai RandomForest với R 27](#_Toc43729948)

[3.4 Random Forest với WEKA 35](#_Toc43729949)

[3.5 Kết luận và hướng phát triển đề tài 38](#_Toc43729950)

[3.5.1 Kết luận 38](#_Toc43729951)

[3.5.2 Hướng phát triển đề tài 39](#_Toc43729952)

[DANH MỤC CÁC TÀI LIỆU THAM KHẢO 40](#_Toc43729953)

**DANH MỤC HÌNH**

[**Hình 2.1: Sơ đồ hoạt động của thuật toán RF** 12](file:///C:\Users\Dinh%20Van%20Thai\Documents\spambase\dtkThoa.docx#_Toc43341410)

**DANH MỤC BẢNG**

[**Bảng 1: Đánh giá hiệu suất của thuật toán Random Forest** 37](#_Toc43342122)

[**Bảng 2: Lỗi huấn luyện và mô phỏng của thuật toán Random Forest** 38](#_Toc43342123)

[**Bảng 3: Đánh giá hiệu suất của thuật toán RFs dựa trên TPF, FPR, Precision, F-Score** 38](#_Toc43342124)

[**Bảng 4: Ma trận nhầm lẫn của thuật toán Random Forest** 38](#_Toc43342125)

# LỜI MỞ ĐẦU

Email là hình thức giao tiếp điện tử phổ biến nhất hiện nay cho cả doanh nghiệp và người dùng. Vấn đề Email Spam (thư rác) là một trong những thách thức lớn hàng ngày của mọi người dùng email trên thế giới. Trung bình một người dùng email cơ sở hàng ngày nhận được hàng trăm thư rác có nội dung mới, từ địa chỉ ẩn danh được tạo tự động bởi các phần mềm robot. Chúng chiếm một lượng không gian và băng thông khổng lồ. Các phương pháp lọc thư rác truyền thống như danh sách đen và danh sách trắng bằng cách sử dụng (tên miền, IP địa chỉ, địa chi gởi thư) đã được chứng minh là không hiệu quả trong việc cắt giảm mối đe doạ của tin nhắn rác. Điều này mang lại nhu cầu phát minh ra các bộ lọc thư rác có độ tin cậy và hiệu quả. Gần đây, phương pháp học máy đã được áp dụng thành công trong việc phát hiện và lọc Email Spam.

Trong chuyên đề này tôi tìm hiểu kỹ thuật học máy Random Forest (Rừng ngẫu nhiên) và sử dụng kỹ thuật này để phân loại hiệu quả các tin nhắn Email Spam. Chuyên đề đã nghiên cứu, cài đặt thuật toán Random Forest và chạy thử nghiệm trên bộ dữ liệu Spambase của UCI phân lớp Email Spam và Non-Spam.

# Chương 1: TỔNG QUAN

## 1.1 Email Spam

Email là một phương thức trao đổi tin nhắn giữa những người sử dụng các thiết bị điện tử. Chính những ưu điểm về tốc độ, hiệu quả chi phí và dễ dàng sử dụng từ máy tính cá nhân, điện thoại và các thiết bị điện tử thế hệ mới đã khiến email trở nên phổ biến toàn cầu. Một tài khoản email được yêu cầu để đăng ký bất cứ hoạt động trực tuyến nào bao gồm các trang mạng xã hội, nhắn tin tức thời và bất kì hình thức tài khoản nào khác hiện diện trên internet. Mua hàng trực tuyến cũng liên quan đến một địa chỉ email để xác nhận, nhận thông báo và nhiều hơn. Theo báo cáo của Radicati, tổng số email doanh nghiệp và người dùng được gởi và nhận mỗi ngày sẽ vượt quá 306 tỷ vào năm 2020 và được dự báo sẽ tăng lên hơn 361 tỷ vào cuối năm 2020. Số lượng người dùng email trên toàn thế giới sẽ đứng đầu 4.0 vào năm 2020 và dự kiến tăng lên 4,4 tỷ người dùng vào cuối năm 2024. Hơn một nửa thế giới sử dụng email trong năm 2020. Việc sử dụng rộng rãi các email để liên lạc và giao dịch đã dẫn đến sự gia tăng số lượng Email Spam gởi và nhận trên toàn cầu. Email Spam đã thành vấn đề lớn trên internet, là mối đe doạ nghiêm trọng đối với người dùng email.

Email Spam là những bức thư điện tử không yêu cầu, không mong muốn được gởi hàng loạt tới người nhận. Người gởi Email Spam thu thập địa chỉ email người dùng từ các nguồn khác nhau như các mẫu điền, địa chỉ sử dụng trên các trang web, phần mềm độc hại. Email Spam cản trở người dùng Internet tối đa hoá dung lượng lưu trữ và băng thông mạng. Khối lượng lớn thư rác trong máy tính gây bất lợi cho hiệu quả sử dụng bộ nhớ máy chủ, băng thông, tốc độ xử lí CPU và thời gian người dùng. Theo các báo cáo cho thấy Email Spam chiếm hơn 57% lưu lượng email truy cập trên toàn cầu (Kasperky, 2019). Các nghiên cứu được thực hiện để tạo ra các kỹ thuật lọc mới để loại bỏ hoặc ít nhất chặn các Email Spam khỏi các cản trở hoạt động của người dùng email. Nhiều kỹ thuật đã được đề xuất và đã được áp dụng thành công để phân loại email.

## 1.2 Kỹ thuật phân loại Email Spam

Những kỹ thuật bao gồm xác suất (probabilistic), cây quyết định (decision tree), hệ thống miễn dịch nhân tạo (Artificial Immune System - AIS), máy véc-tơ hỗ trợ (Support Vector Machine - SVM), mạng lưới thần kinh nhân tạo (Artificial Neural Networks - ANN) và kỹ thuật dựa trên trường hợp (case-based technique). Các kỹ thuật đã được chứng minh rằng có thể sử dụng các kỹ thuật học máy này để lọc Email Spam bằng cách sử dụng các phương pháp lọc dựa trên nội dung có khả năng xác định cụ thể các tính năng trong tin nhắn email (thường là từ khoá thường được sử dụng trong Email Spam). Tần số của các tính năng này xuất hiện trong email được đo theo giá trị ngưỡng. Khi tin nhắn email vượt quá ngưỡng giá trị được phân loại là Email Spam. Vấn đề chính của nhiều kỹ thuật trên là hiệu suất các bộ lọc thấp và cần phải tăng độ chính xác phân loại của bộ lọc. Ngoài ra, nhiều kỹ thuật không quá mạnh mẽ để đối phó với Email Spam ngày càng tinh vi. Bài viết này đề cập việc sử sụng kỹ thuật Random Forest để lọc Email Spam, một trong những kỹ thuật đạt hiệu suất và độ chính xác cao, là một thuật toán đầy hứa hẹn trong việc giải quyết vấn đề lọc Email Spam.

## 1.3 Giới thiệu Random Forest

Random Forest (RF) là một ví dụ cổ điển về kỹ thuật học máy và hồi quy giải quyết các vấn đề phân loại dữ liệu. Thuật toán được phát triển bởi Breiman và Culter. Thuật toán phân loại dữ liệu thành các lớp khác nhau sử dụng cây quyết định. Trong giai đoạn huấn luyện, một số cây quyết định được tạo ra và sau đó được sử dụng cho các nhiệm vụ phân loại. Điều này hoạt động bằng cách xem xét các lớp bầu của từng cây và lớp có số phiếu bầu cao nhất được coi là kết quả cuối cùng.

Thuật toán Random Forest đã trở nên phổ biến trong những năn qua và đang được áp dụng giải quyết các bài toán phân loại và dự báo dữ liệu trong các lĩnh vực khác nhau. Ứng dụng lọc Email Spam là một trong bài toán phân loại dữ liệu của RF, RF trích xuất các tính năng quan trọng từ email và phân loại email thành một trong hai email thông thường hoặc Email Spam.

Đầu tiên các lý thuyết liên quan được trình bày trong chương 2, ứng dụng lọc Email Spam thí nghiệm trên bộ dữ liệu Spambase của UCI được trình bày trong chương và cuối cùng trình bày kết luận và hướng phát triển.

# Chương 2: CƠ SỞ LÝ THUYẾT

## 2.1 Giới thiệu học máy

Theo Arthur Samuel (1959), học máy (Machine Learning) là ngành học cung cấp cho máy tính khả năng học hỏi mà không cần được lập trình một cách rõ ràng. Theo định nghĩa của Wikipedia, máy học là một lĩnh vực của trí tuệ nhân tạo, nghiên cứu về các thuật toán máy tính cải thiện tự động thông qua kinh nghiệm. Các thuật toán học máy xây dựng một mô hình toán học dựa trên dữ liệu mẫu, được gọi là “dữ liệu huấn luyện” để đưa ra dự đoán hoặc quyết định mà không cần lập trình cụ thể. Các thuật toán máy học được phân loại theo kết quả mong muốn của thuật toán. Hai thuật toán được chấp nhận rộng rãi là học có giám sát (supervised learning) và học không giám sát (unsupervised learning). Ngoài ra còn có các thuật toán khác như học bán giám sát (semisupervised learning), học tăng cường (reinforcement learning), tự học (self-learning)…

Học có giám sát là thuật toán dự đoán đầu ra (outcome) của một dữ liệu mới dựa trên các cập (input, outcome) đã biết từ trước. Cặp dữ liệu còn được gọi là (data, label). Học có giám sát có hai loại chính là phân lớp và hồi quy.

Học không giám sát không có nhãn nào được đưa cho thuật toán. Thuật toán sẽ dựa vào cấu trúc dữ liệu để thực hiện một công việc nào đó. Học không giám sát cũng có hai loại chính là phân nhóm và hội.

Các thuật toán học máy áp dụng rộng rãi trong nhiều ứng dụng như lọc email, phân tích thị trường chứng khoán, thị giác máy tính và bảo mật mạng, là nơi rất khó hoặc không thể phát triển các thuật toán thông thường để thực hiện tác vụ cần thiêt.

Random Forest (RF) là một thuật toán học máy phổ biến thuộc về kỹ thuật học có giám sát. RF có thể sử dụng cho cả các vấn đề phân loại và hồi quy trong học máy. Nó dựa trên khái niệm học tập kết hợp (ensemble learning), là một quá trình kết hợp nhiều phân loại để giải quyết một vấn đề phức tạp và cải thiện hiệu suất của mô hình.

## 2.2 Phân lớp dựa vào cây quyết định

Phân lớp (Classification) là quá trình xếp một đối tượng vào một trong những lớp đã biết (ví dụ: phân lớp tin tức thuộc lĩnh vực tài chính, thể thao, thời sự... phân lớp học sinh theo các mức yếu, trung bình, khá, giỏi..., phân lớp email theo loại spam, non-spam…). Phân lớp thuộc lớp bài toán học có giám sát (supervised learning), mục đích là gán các đối tượng/mẫu mới vào các lớp với độ chính xác cao nhất có thể.

Dự đoán (Prediction) liên quan đến việc sử dụng các biến hoặc các trường trong cơ sở dữ liệu để chiết xuất ra các mẫu nhằm dự đoán những giá trị chưa biết hoặc giá trị của các biến sẽ được quan tâm trong tương lai. Mô tả tập trung vào việc tìm kiếm các mẫu biểu diễn dữ liệu mà con người có thể hiểu được.

Cây quyết định (Decision tree) là một mô hình dự báo (predictive model), là một ánh xạ từ các dữ liệu quan sát được về một sự vật/hiện tượng đến các kết luận về giá trị đích của sự vật/hiện tượng đó. Mỗi nút trong (internal node) tương ứng với một biến; đường nối giữa một nút trong với các nút con thể hiện giá trị cụ thể cho biến. Mỗi nút lá đại diện cho giá trị dự đoán của biến (các giá trị dự đoán của các biến được biểu diễn bởi đường đi từ nút gốc tới nút lá). Kỹ thuật học máy dùng trong cây quyết định được gọi là học bằng cây quyết định, hay gọi ngắn gọn là cây quyết định. Cây quyết định là sự kết hợp các kỹ thuật toán học nhằm hỗ trợ việc mô tả, phân lớp và tổng quát hóa một tập dữ liệu cho trước.

Cây quyết định là phương pháp qui nạp học tri thức dựa trên phân lớp. Để sử dụng cây quyết định tập mẫu cần có các điều kiện:

* Các đối tượng cần xử lý phải được mô tả bởi một tập các thuộc tính\_giá trị (attribute\_value description).
* Các lớp phải được xác định trước (predefined class) (dữ liệu có giám sát – supervised data)
* Đầy đủ dữ liệu
* Các lớp rời rạc

### 2.2.1 Xây dựng cây quyết định

Cây được thiết lập bằng cách thực hiện chia đệ quy tập mẫu dữ liệu huấn luyện cho đến khi các đối tượng/mẫu ở mỗi nút lá thuộc cùng một lớp. Các bước cụ thể của giai đoạn này như sau:

* Rời rạc hóa các thuộc tính dạng phi số
* Các mẫu huấn luyện xuất phát nằm ở gốc của cây
* Chọn một thuộc tính để phân chia tập mẫu huấn luyện thành các nhánh. Thuộc tính được chọn
* Dựa trên độ đo thống kê hoặc độ đo heuristic
* Tiếp tục lặp việc xây dựng cây quyết định cho các nhánh, quá trình lặp dừng khi:
* Tất cả các mẫu đều được phân lớp (thuộc một nút lá)
* Không còn thuộc tính nào có thể dùng để phân chia mẫu được nữa

Cắt tỉa cây: là việc làm nhằm tối ưu hoá cây. Cắt tỉa cây chính là việc trộn một cây con vào trong một nút lá.

Đánh giá cây: Dùng để đánh giá độ chính xác của cây kết quả. Tiêu chí đánh giá là tỷ số giữa tổng số đối tượng được phân lớp chính xác trên tổng số đối tượng cần phân lớp.

### 2.2.2 Chỉ số Gini (Gini index)

Để chọn thuộc tính phân nhánh và tiến hành phân nhánh đối với một thuộc tính liên tục (Continuos Attribute) người ta còn sử dụng chỉ số GINI (GINI Index):

Chỉ số Gini của nút t:

Trong đó:

* p(j/t) là tần suất của lớp j trong nút t
* Lớn nhất là 1-1/nc khi các mẫu phân bố đều trên các lớp
* Thấp nhất là 0 khi các mẫu chỉ thuộc về một lớp

Khi phân chia nút p thành k nhánh, chất lượng của phép chia được tính bằng:

Trong đó:

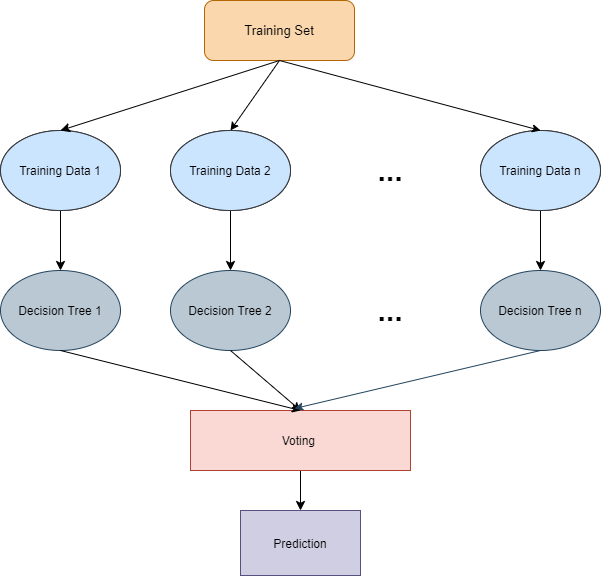
* ni là số mẫu trong nút i
* n là số mẫu trong nút p

Người ta chọn thuộc tính GINIchia nhỏ nhất để phân nhánh

## 2.3 Giới thiệu thuật toán Random Forest

Random Forest là một bộ phân loại bao gồm nhiều cây quyết định, mỗi cây quyết định dự đoán một kết quả và kết quả nào được nhiều cây quyết định dự đoán thì đó là kết quả cuối cùng. Như vậy, thay vì dựa vào một cây quyết định, RF sẽ đưa ra dự đoán từ mỗi cây và dựa trên số phiếu bầu đa số dự đoán và dự đoán kết quả cuối cùng. Số lượng các cây trong rừng càng nhiều thì độ chính xác càng cao. Hình 2.1 giải thích hoạt động của thuật toán RF. Do RF kết hợp nhiều cây để dự đoán lớp của tập dữ liệu, nên có thể một số cây quyết định có thể dự đoán đầu ra chính xác trong khi những cây khác thì không. Nhưng với rừng cây, dự đoán sẽ chính xác. Do đó dưới đây là hai giả định để rừng cây phân loại tốt hơn:

* Cần có một số giá trị thực tế trong biến tính năng của bộ dữ liệu để bộ phân loại có thể dự đoán kết quả chính xác thay vì kết quả đoán.
* Các dự đoán từ mỗi cây phải có mối tương quan rất thấp.

Random Forest (được mô tả trong hình 2.1) tạo ra một tập hợp các cây quyết định không cắt nhánh, mỗi cây được xây dựng trên tập mẫu bootstrap, tại mỗi nút phân hoạch tốt nhất được thực hiện từ việc chọn ngẫu nhiên một tập con các thuộc tính. Lỗi tổng quát của rừng phụ thuộc vào độ chính xác của từng cây thành viên trong rừng và sự phụ thuộc lẫn nhau giữa các cây thành viên.

**Hình 2.1: Sơ đồ hoạt động của thuật toán RF**

## 2.4 Bootstrap và bagging

### 2.4.1 Bootstrap

Bootstrap là một phương pháp rất nổi tiếng trong thống kê được giới thiệu bởi Bradley Efron vào năm 1979. Phương pháp này chủ yếu dùng để ước lượng lỗi chuẩn (standard errors), độ lệch (bias) và tính toán khoảng tin cậy (confidence interval) cho các tham số. Phương pháp này được thực hiện như sau: Từ một quần thể ban đầu lấy ra một mẫu L = (x1, x2,..xn) gồm 4 thành phần, tính toán các tham số mong muốn. Trong các bước tiếp theo lặp lại b lần việc tạo ra mẫu Lb cũng gồm n phần từ từ L bằng cách lấy lại mẫu với sự thay thế các thành phần trong mẫu ban đầu sau đó tính toán các tham số mong muốn.

Trong Random Forest, bootstrap được sử dụng để tạo mẫu ngẫu nhiên cho từng cây p vì thế mỗi cây sẽ có một tập mẫu ngẫu nhiên riêng biệt. Ngoài ra còn được sử dụng để đánh giá nội tại của thuật toán (Out-of-bag).

### 2.4.2 Bagging

Phương pháp Bagging (hay tổng hợp Bootstrap) được xem như là một phương pháp tổng hợp kết quả có được từ các bootstrap. Tư tưởng chính của phương pháp này như sau: Cho một tập huấn luyện D={(xi,yi): i=1,2,…,n} và giả sử chúng ta muốn có một một dự đoán nào đó đối với biến x. Một mẫu gồm B tập dữ liệu, mỗi tập dữ liệu gồm n phần tử được chọn lựa ngẫu nhiên từ D với sự thay thế (giống như bootstrap). Do đó B=(D1, D2, ….,DB) trông giống như là một tập các tập huấn luyện được nhân bản. Tập huấn một máy hoặc một mô hình đối với mỗi tập Db (b=1, 2, …,B) và lần lượt thu thập các kết quả dự báo có được trên mỗi tập Db. Kết quả tổng hợp cuối cùng được tính toán bằng cách trung bình hóa (regression) hoặc thông qua số phiếu bầu nhiều nhất (classification).

Quy trình trên dẫn đến hiệu suất mô hình tốt hơn vì nó làm giảm phương sai của độ lệch. Điều này có nghĩa là trong khi các dự đoán của một cây rất nhạy cảm với độ nhiễu trong tập huấn luyện của nó, thì trung bình của nhiều cây là không, miễn là các cây không tương quan. Chỉ nhiều cây trên một tập huấn luyện duy nhất sẽ cho các cây tương quan mạnh. Lấy mẫu bootstrap là một cách để loại bỏ các cây bằng cách hiển thị cho chúng các bộ huấn luyện khác nhau.

Số lượng mẫu/cây, B, là một tham số tự do. Thông thường, một vài trăm đến vài nghìn cây được sử dụng, tùy thuộc vào kích thước và tính chất của bộ huấn luyện. Có thể tìm thấy số lượng cây B tối ưu bằng cách xác thực chéo hoặc bằng cách quan sát lỗi out-of-bag: lỗi dự đoán trung bình trên mỗi mẫu huấn luyện xi, chỉ sử dụng các cây không có xi trong mẫu bootstrap của chúng. Lỗi và kiểm tra có xu hướng chững lại sau khi một số cây đã phù hợp.

Random Forest là một cải tiến đối với cây quyết định đóng gói (Bagged decision trees). Các khu rừng ngẫu nhiên thay đổi thuật toán theo cách mà các cây con được học để dự đoán kết quả từ tất cả các cây con có ít mối tương quan. Khi chọn điểm phân tách RF để thuật toán học được giới hạn trong một mẫu ngẫu nhiên các tính năng cần tìm kiếm, một tập hợp con ngẫu nhiên của các tính năng. Lý do cho việc này là sự tương quan của các cây trong một mẫu bootstrap thông thường: nếu một hoặc một vài tính năng là các yếu tố dự báo rất mạnh cho biến phản ứng (đầu ra đích), các tính năng này sẽ được chọn trong nhiều cây B, gây ra chúng để trở nên tương quan. Số lượng các tính năng có thể được tìm kiếm tại mỗi điểm phân chia (m) phải được chỉ định làm tham số cho thuật toán. Bạn có thể thử các giá trị khác nhau và điều chỉnh nó bằng cách sử dụng xác nhận chéo. Thông thường, đối với vấn đề phân loại với các tính năng p, m = sqrt(p) (làm tròn xuống) là số lượng các tính năng được chọn ngẫu nhiên có thể được tìm kiếm tại một điểm phân tách. Ví dụ: nếu một tập dữ liệu có 64 biến đầu vào cho một vấn đề phân loại, thì: m = sqrt (64), m = 8. Đối với quy hồi, mặc định tốt nhất là m = p/3.

### 2.4.3 Tính năng quan trọng

Khi các cây quyết định đóng gói được xây dựng, chúng ta có thể tính toán hàm lỗi cho một tính năng tại mỗi điểm phân chia. Với các vấn đề hồi quy, đó có thể là lỗi tổng bình phương và trong vấn đề phân lớp đó có thể là điểm Gini. Những giọt lỗi này có thể được tính trung bình trên tất cả cây quyết định và đầu ra để đưa ra ước tính về tầm quan trọng của từng biến đầu vào. Độ giảm càng lớn khi biến được chọn, tầm quan trọng càng lớn.

Các đầu ra này có thể giúp xác định các tập hợp con của các biến đầu vào có thể liên quan nhiều nhất hoặc ít nhất đến vấn đề và đề xuất các thử nghiệm lựa chọn tính năng có thể. Bạn có thể thực hiện khi một số tính năng bị xóa khỏi bộ dữ liệu.

## 2.5 Thuật toán Random Forest

Random Forest hoạt động theo hai giai đoạn, giai đoạn đầu tiên là tạo rừng ngẫu nhiên bằng cách kết hợp N cây quyết định, thứ hai là đưa ra dự đoán cho từng cây được tạo trong giai đoạn đầu tiên.

Các bước để trồng cây được nêu dưới dây:

1. Giả sử N là số lượng trường hợp huấn luyện, đại diện ngẫu nhiên N thể hiện có thể được thay thế từ dữ liệu hiện có. Những trường hợp như vậy được sử dụng như huấn luyện thiết lập để trồng cây.
2. Giả sử có các biến đầu vào P, một số p << P được chỉ định sao cho mỗi biến nút tương ứng, các biến p được chọn ngẫu nhiên từ P và phần tốt nhất trên trên p được sử dụng để phân vùng nút để p hiện tại có giá trị cố định trong suốt khoảng thời gian trồng rừng.
3. Mỗi cây được xây dựng đến độ sâu tối đa (không cắt nhánh).

Cây được gọi là phân loại mạnh khi nó có tỷ lệ lỗi nhỏ. Hơn nữa, lỗi tốc độ của rừng giảm khi nồng độ của từng cây trong rừng tăng. Giảm giá trị của p làm giảm mối quan hệ và sức mạnh của rừng trong khi tăng cường giá trị của p tăng cả trong khu vực ranh giới tốt nhất của p thường rất mở rộng. Giá trị của p có thể được tính bằng lỗi Out-of-bag (OOB) (còn được gọi là ước tính ngoài túi) một giá trị p trong giới hạn có thể được tính vị kịp thời. Đây là chỉ yếu tố số mà các khu rừng ngẫu nhiên hởi nhạy cảm với sự tinh chỉnh của nó.

Thuật toán dưới dây phát thảo chính xác các bước cần thiết để tạo rừng cây:

* **Bắt đầu**

**Đầu vào:** X: số lượng các nút

N: số lượng các tính năng

Y: số cây được được trồng

**Đầu ra:** G: lớp có số phiếu bầu cao nhất

* **Thuật toán**

**While** (tiêu chí dùng là không đúng) **do**

Chọn ngẫu nhiên mẫu S tự khỏi động từ dữ liệu huấn luyện Y

Tạo cây R𝑖 từ mẫu S tự khởi động đã chọn từ các bước bên dưới:

(1) Chọn n tính năng ngẫu nhiên từ N; trong đó n≪N

(2) Tính điểm phân chia tốt nhất cho nút d trong số các đặc trưng n

(3) Chia nút cha cho hai nút con thông qua phân chia tối ưu

(4) Thực hiện các bước 1 – 3 cho đến khi số nút (x) tối đa được tạo

Tạo rừng bằng cách lặp lại các bước 1 – 4 cho số lần Y

**EndWhile**

Tạo đầu ra cho mỗi cây được tạo {Rt}1 Y

Sử dụng một mẫu mới cho mỗi cây được tạo bắt đầu từ nút gố=c

Gán mẫu cho lớp khớp với nút lá

Hợp nhất phiếu bầu hoặc kết quả của mỗi cây

Xuất ra lớp có số phiếu bầu cao nhất (G)

**End RF Algorithm**

****

**https://commons.wikimedia.org/wiki/File:Random\_forest\_diagram\_complete.png**

## 2.6 Đặc điểm của Random Forest

### 2.6.1. Ưu điểm

* RF có khả năng thực hiện cả hai nhiệm vụ phân loại và hồi quy.
* RF giảm lỗi phân loại tốt hơn khi so sánh với một số kỹ thuật máy học khác. Hiệu suất nói chung của thuật toán tốt bằng hoặc thậm chí hoạc vượt trội so vói SVM.
* RF có thể xử lý hiệu quả bộ dữ liệu không cân bằng có giá trị thiếu. Nó phục vụ như một thuật toán hiệu quả để tính toán giá trị ước tính của dữ liệu bị thiếu và duy trì độ chính xác của dữ liệu trong các trường hợp trong đó một tỷ lệ đáng kể của dữ liệu bị thiếu.
* Thời gian huấn luyện cho các RF thường là ngắn hơn so với SVM và Neural Networks (mặc dù điều này phụ thuộc vào từng cá nhân thực hiện).
* RF tốt hơn hầu hết các thuật toán máy học hiện có về độ chính xác. Hiệu suất của RF trong cơ sở dữ liệu lớn là rất tốt. Nó có thể xử lý hiệu quả hàng trăm của hàng ngàn biến đầu vào.
* RF tạo ra một dự đoán không thiên vị nội bộ của tập thể lỗi trong quá trình canh tác rừng. Nó cung cấp cách tiếp cận cho các lỗi làm dịu trong lớp dân số có tập dữ liệu sai lệch.
* RF cũng có khả năng xử lý hiệu quả việc tạo dữ liệu không nhãn. Nó là một kỹ thuật rất thích hợp để phân cụm dữ liệu không nhãn.
* RF không phức tạp và nó sử dụng ít tham số hơn khi so sánh với số lượng quan sát.

### 2.6.2 Hạn chế

* Dữ liệu huấn luyện cần được đa dạng hóa và cân bằng về số nhãn lớp. Việc không cân bằng nhãn lớp khiến kết quả dự đoán của thuật toán có thể lệch về số đông nhãn lớp.
* Thời gian huấn luyện của rừng có thể kéo dài tùy số cây và số thuộc tính phân chia.
* Tuy RF được sử dụng cho cả phân loại và hồi quy. Tuy nhiên, RF không thực sự phù hợp với các nhiệm vụ hồi quy.
* Ta không thể hiểu được cơ chế hoạt động bên trong mô hình của thuật toán vì cấu trúc quá phức tạp.

### 2.6.3 Out-of-bag

Khi tập mẫu được rút ra từ một tập huấn luyện của một cây với sự thay thế (bagging), thì theo ước tính có khoảng 1/3 các phần từ không có nằm trong mẫu này. Điều này có nghĩa là chỉ có khoảng 2/3 các phần tử trong tập huấn luyện tham gia vào trong các tính toán của chúng ta, và 1/3 các phần tử này được gọi là dữ liệu Out-of-bag (OOB). Dữ liệu Out-of-bag được sử dụng để ước lượng lỗi tạo ra từ việc kết hợp các kết quả tổng hợp từ các cây trong Random Forest, ước tính lỗi kiểm tra độ tin cậy và tương quan tốt với các ước tính xác nhận chéo.

# Chương 3: LỌC EMAIL SPAM BẰNG KỸ THUẬT RANDOM FOREST

## 3.1 Bộ dữ liệu thí nghiệm

Bộ dữ liệu được sử dụng là bộ dữ liệu Spambase của UCI. Cột cuối cùng của “spambase.data” hiển thị xem email có phải là Email Spam (1) hay Non Spam (0). Hầu hết các thuộc tính cho biết liệu từ hoặc ký tự cụ thể có thường xuyên xuất hiện trong email hay không. Các thuộc tính run-length (55 - 57) đo dộ dài của chuỗi các chữ in hoa liên tiếp. Spanbase có 4601 trường hợp, trong đó 2788 là non-spam (chiếm 60.6%), 1813 là spam (chiếm 39.4%). Bộ dữ liệu gồm 58 thuộc tính trong đó có 57 thuộc tính thuộc kiểu liên tục và 1 thuộc tính phân lớp như sau:

* 48 thuộc tính kiểu thực liên tục [0,100] thuộc loại word\_freq\_WORD

= Tỷ lệ % các từ trong email khớp với WORD nghĩa là 100 \* (Số lần từ WORD xuất hiện trong email) / tổng số từ tỏng email. Một "word" trong trường hợp này là bất kỳ chuỗi ký tự chữ và số nào được giới hạn bởi các ký tự không chữ và số hoặc kí tự kết thúc chuỗi (end-of-line).

* 6 thuộc tính kiểu thực liên tục [0,100] thuộc loại char\_freq\_CHAR]

= Tỷ lệ % các kí tự trong email khớp với CHAR, nghĩa là 100 \* (Số lần xuất hiện CHAR) / tổng số ký tự trong email.

* 1 thuộc tính kiểu thực liên tục [1,...] thuộc loại capital\_run\_length\_average

= Độ dài trung bình của chuỗi kí tự in hoa không bị gián đoạn .

* 1 thuộc tính kiểu nguyên liên tục [1,...] thuộc loại capital\_run\_length\_longest

= độ dài của chuỗi chữ in hoa dài nhất không bị gián đoạn.

* 1 thuộc tính kiểu nguyên liên tục [1,...] thuộc loại capital\_run\_length\_total

= Tổng chiều dài của chuỗi chữ in hoa không gián đoạn.

= Tổng chữ số in hoa trong email.

* 1 thuộc tính định dạng {0,1} phân lớp spam

= biểu thị xem email có được xem là spam (1) hay không spam (0)

## 3.2 Chương trình demo thuật toán Random Forest trên python

Áp dụng thuật toán Random Forest trên bộ dữ liệu Spambase. Dữ liệu cần chuyển đổi thành một tệp \*CSV làm đầu vào của thuật toán.

Xây dựng một load\_csv() để tải file dữ liệu, giả định rằng một file CSV của tập dữ liệu với tên tệp emailspam.csv nằm trong thư mục làm việc hiện tại.

Theo như lý thuyết đã trình bày trong chương 2, với cây quyết định, các điểm phân chia được chọn bằng cách tìm thuộc tính và giá trị của thuộc tính đó dẫn đến chi phí thấp nhất. Đối với vấn đề phân loại, hàm chi phí này thường là chỉ số GINI, tính toán độ thuần khiết của các nhóm dữ liệu được tạo bởi các điểm phân tách. Chỉ số GINI = 0 là độ thuần khiết hoàn hảo trong đó các lớp giá trị được phân tách thành hai nhóm, trong trường hợp vấn đề phân loại có hai lớp. Tìm kiếm điểm phân chia tốt nhất trong cây quyết định liên quan đến việc đánh giá chi phí của từng giá trị trong tập dữ liệu huấn luyện cho từng biến đầu vào.

Đối với Random Forest, quy trình trên được thực hiện được thực hiện dựa trên mẫu thay thế. Lấy mẫu với thay thế có nghĩa là cùng một hàng có thể được chọn và thêm vào mẫu nhiều lần. Mẫu thuộc tính đầu vào này có thể được chọn ngẫu nhiên và không thay thế, nghĩa là mỗi thuộc tính đầu vào chỉ cần được xem xét một lần khi tìm kiếm điểm phân chia với chi phí thấp nhất.

Hàm get\_list() xây dựng để thực hiện công việc này. Nó lấy một tập dữ liệu và một số tính năng đầu vào cố định để dánh giá là đối số đầu vào, trong đó tập dữ liệu có thể là một mẫu của tập dữ liệu huấn luyện thực tế.

#chọn phân chia tốt nhất cho bộ dữ liệu

def get\_split(dataset, n\_features):

class\_value = list(set(row[-1] for row in dataset))

b\_index, b\_value, b\_score, b\_groups = 999, 999, 999, None

features = list()

while len(features) < n\_features:

index = randrange(len(dataset[0]) - 1)

if index not in features:

features.append(index)

for index in features:

for row in dataset:

groups = test\_split(index, row[index], dataset)

gini = gini\_index(groups, class\_value)

if gini < b\_score:

b\_index, b\_value, b\_score, b\_groups = index, row[index], gini, groups

return {'index':b\_index, 'value':b\_value, 'groups':b\_groups}

Hàm trợ giúp test\_list() được sử dụng để phân chia tập dữ liệu theo điểm phân chia ứng viên và gini\_index() được sử dụng để đánh giá chi phí của một phân chia nhất định cho các nhóm hàng được tạo.

Chúng ta có thể thấy rằng một danh sách các tính năng được tạo bằng cách chọn ngẫu nhiên các chỉ số tính năng và thêm chúng vào danh sách (được gọi là features), danh sách các tính năng này sau đó được liệt kê và các giá trị cụ thể trong tập dữ liệu huấn luyện được đánh giá là điểm phân chia.

Số lượng thuộc tính được xem xét để phân tách được giới hạn ở căn bậc hai của số lượng tính năng đầu vào: num\_features\_for\_split = sqrt(total\_input\_features)

Sử dụng xác nhận chéo K-fold để ước tính hiệu suất của mô hình. Điều này có nghĩa là xây dựng và đánh giá các mô hình k và ước tính hiệu suất là lỗi mô hình trung bình. Độ chính xác phân loại để sẽ được sử dụng để đánh giá từng mô hình. Các hành vi này được cung cấp trong các hàm trợ giúp cross\_validation\_split(), accuracy\_metric() và evaluate\_algorithm().

Các hàm test\_split(), gini\_index(), get\_split(), to\_terminal(), split() và build\_tree() được sử dụng để tạo một cây quyết định duy nhất. Hàm predict() để đưa ra dự đoán với một cây quyết định. Hàm subsample() để tạo các mẫu con của tập huấn luyện và bagging\_predict() để đưa ra dự đoán với danh sách các cây quyết định.

Một hàm Random\_forest() được phát triển trước tiên tạo ra một danh sách các cây quyết định từ mẫu con của tập dữ liệu huấn luyện và sau đó sử dụng chúng để đưa ra dự đoán.

Code demo:

from random import seed

from random import randrange

from csv import reader

from math import sqrt

#load a CSV file  
def load\_csv(filename):

dataset = list()

with open(filename, 'r') as file:

csv\_reader = reader(file)

for row in csv\_reader:

if not row:

continue

dataset.append(row)

return dataset

# tạo mẫu ngẫu nhiên S từ tập dữ liệu huấn luyện Y có thay thế

def subsample(dataset, ratio):

sample = list()

n\_sample = round(len(dataset) \* ratio)

while len(sample) < n\_sample:

index = randrange(len(dataset))

sample.append(dataset[index])

return sample

# phân tách tập dữ liệu dựa trên 1 thuộc tính và 1 giá trị thuộc tính

def test\_split(index, value, dataset):

left, right = list(), list()

for row in dataset:

if row[index] < value:

left.append(row)

else:

right.append(row)

return left, right

# tính chỉ số gini cho tập dữ liệu phân tách

def gini\_index(groups, classes):

n\_instances = float(sum([len(group) for group in groups ]))

gini = 0.0

for group in groups:

size = float(len(group))

if(size == 0):

continue

score = 0.0

#tính điểm score group dựa trên điểm của mỗi lớp

for class\_val in classes:

p = [row[-1] for row in group].count(class\_val)/size

score += p\*p

gini += (1.0 - score) \* (size/n\_instances)

return gini

#chọn phân chia tốt nhất cho bộ dữ liệu

def get\_split(dataset, n\_features):

class\_value = list(set(row[-1] for row in dataset))

b\_index, b\_value, b\_score, b\_groups = 999, 999, 999, None

features = list()

while len(features) < n\_features:

index = randrange(len(dataset[0]) - 1)

if index not in features:

features.append(index)

for index in features:

for row in dataset:

groups = test\_split(index, row[index], dataset)

gini = gini\_index(groups, class\_value)

if gini < b\_score:

b\_index, b\_value, b\_score, b\_groups = index, row[index], gini, groups

return {'index':b\_index, 'value':b\_value, 'groups':b\_groups}

# Tạo 1 node giá trị đầu ra

def to\_terminal(group):

outcomes = [row[-1] for row in group]

return max(set(outcomes), key=outcomes.count)

#tạo các nút phân tách cho các nut con

def split(node, max\_depth, min\_size, n\_features, depth):

left, right = node['groups']

del(node['groups'])

if not left or not right:

node['left'] = node['right'] = to\_terminal(left + right)

return

# check for max depth

if depth >= max\_depth:

node['left'], node['right'] = to\_terminal(left), to\_terminal(right)

return

# process left child

if len(left) <= min\_size:

node['left'] = to\_terminal(left)

else:

node['left'] = get\_split(left, n\_features)

split(node['left'], max\_depth, min\_size, n\_features, depth+1)

# process right child

if len(right) <= min\_size:

node['right'] = to\_terminal(right)

else:

node['right'] = get\_split(right, n\_features)

split(node['right'], max\_depth, min\_size, n\_features, depth+1)

# Xây dựng cây quyết định

def build\_tree(train, max\_depth, min\_size, n\_features):

root = get\_split(train, n\_features)

split(root, max\_depth, min\_size, n\_features, 1)

return root

# tạo 1 dự đoán với 1 cây quyết định

def predict(node, row):

if row[node['index']] < node['value']:

if isinstance(node['left'], dict):

return predict(node['left', row]

else:

return node['left']

else:

if isinstance(node['right'], dict):

return predict(node['right'], row)

else:

return node['right']

# Tạo một dự đoán với một danh sách các cây bagged

def bagging\_predict(trees, row):

predictions = [predict(tree, row) for tree in trees]

return max(set(predictions), key=predictions.count)

# Random Forest Algorithm

def random\_forest(train, test, max\_depth, min\_size, sample\_size, n\_trees, n\_features):

trees = list()

for i in range(n\_trees):

sample = subsample(train, sample\_size)

tree = build\_tree(sample, max\_depth, min\_size, n\_features)

trees.append(tree)

predictions = [bagging\_predict(trees, row) for row in test]

return(predictions)

# phân tách dữ liệu thành k-fold

def cross\_validation\_split(dataset, n\_folds):

dataset\_split = list()

dataset\_copy = list(dataset)

fold\_size = int(len(dataset) / n\_folds)

for i in range(n\_folds):

fold = list()

while len(fold) < fold\_size:

index = randrange(len(dataset\_copy))

fold.append(dataset\_copy.pop(index))

dataset\_split.append(fold)

return dataset\_split

# tính toán phần trăm chính xác

def accuracy\_metric(actual, predicted):  
 correct = 0

for i in range(len(actual)):

if actual[i] == predicted[i]:

correct += 1

return correct / float(len(actual)) \* 100.0

# Đánh giá thuật toán bằng cách sử dụng phân tách chéo k-fold  
def evaluate\_algorithm(dataset, algorithm, n\_folds, \*args):

folds = cross\_validation\_split(dataset, n\_folds)

scores = list()

for fold in folds:

train\_set = list(folds)

train\_set.remove(fold)

train\_set = sum(train\_set, [])

test\_set = list()

for row in fold:

row\_copy = list(row)

test\_set.append(row\_copy)

row\_copy[-1] = None

predicted = algorithm(train\_set, test\_set, \*args)

actual = [row[-1] for row in fold]

accuracy = accuracy\_metric(actual, predicted)

scores.append(accuracy)

return scores  
   
# test with spambase

seed(222)

# load data

filename = 'emailspam.csv'

dataset = load\_csv(filename)

n\_folds = 5

max\_depth = 10

min\_size = 1

sample\_size = 1

n\_features = int(sqrt(len(dataset[0])-1))

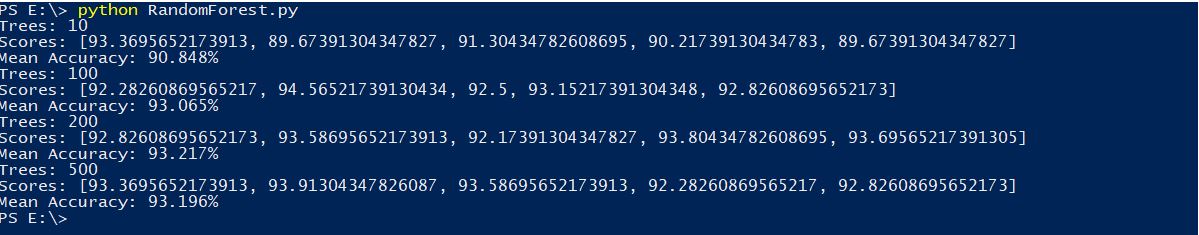
for n\_trees in [10, 100]:

scores = evaluate\_algorithm(dataset, random\_forest, n\_folds, max\_depth, min\_size, sample\_size, n\_trees, n\_features)

print('Trees: %d' % n\_trees)

print('Scores: %s' % scores)

print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))

 Output:

**Nhận xét:**

Từ kết quả cho thấy khu rừng càng nhiều cây thì độ chính xác càng cao.

## 3.3 Triển khai RandomForest với R

Package R “randomForest” được sử dụng để tạo các khu rừng ngẫu nhiên. Sử dụng câu lệnh install.packages(“randomForest”) trong R console để cài đặt các gói. Ngoài ra cần phải cài đặt các gói phụ thuộc khác nếu có. Trong gói “randomForest” có hàm randomForest() để tạo và phân tích các khu rừng ngẫu nhiên.

Cú pháp cơ bản để triển khai một rừng ngẫu nhiên trong R là :

randomForest(formula, ntree=n, mtry=FALSE, maxnodes = NULL)

Trong đó: - formula: Công thức của mô hình được trang bị

- ntree: Số cây trong rừng

- mtry: Số lượng thí sinh rút ra để nuôi thuật toán. Theo mặc định, nó là bình phương của số lượng cột.

- maxnodes: Số lượng nút đầu ra tối đa trong rừng

- importance=TRUE: Đánh giá tầm quan trọng của các biến độc lập trong rừng ngẫu nhiên

Chú ý: rừng ngẫu nhiên có thể được huấn luyện với nhiều thông số hơn.

Xác thực chéo K-Fold được điều khiển bởi hàm trainControl()

Cú pháp hàm trainControl() như sau:

trainControl(method = "cv", number = n, search ="grid")

Trong đó:   
 - method = "cv": Phương pháp được sử dụng để lấy mẫu lại tập dữ liệu

- number = n: Số lượng k-fold cần tạo

- search = "grid": Sử dụng phương pháp tìm kiếm lưới. Đối với phương pháp ngẫu nhiên sử dụng "grid"

Sử dụng thư viện caret để đánh giá mô hình. Thư viện có một hàm train() để đánh giá gần như tất cả các thuật toán học máy. Và hàm predict() có chức năng đưa ra dự đoán.

Cú pháp hàm train() được xác định như sau:

train(formula, df, method = "rf", metric= "Accuracy", trControl = trainControl(), tuneGrid = NULL)

Trong đó: - formula: Xác định công thức của thuật toán

- method: Xác định mô hình cần huấn luyện

- metric = "Accuracy": Xác định cách chọn mô hình tối ưu

- trControl = trainControl(): Xác định các tham số điều khiển

- tuneGrid = NULL`: Trả về một khung dữ liệu với tất cả sự kết hợp có thể

Cú pháp của hàm predict():

predict(model, newdata= df)  
Trong đó:   
 - model: Xác định mô hình đánh giá trước

- newdata: Xác định tập dữ liệu để dưa ra dự đoán

Các bước triển khai thuật toán Random Forest trên R như sau:

Tải các thư viện

**library**(plyr)

**library**(tidyr)

**library**(dplyr)

**library**(ggplot2)

**library**(caret)

**library**(randomForest)

Tải dữ liệu lên và khám phá

> emails = read.csv('C:/Users/Dinh Van Thai/Documents/spambase/emailspam.csv', stringsAsFactors = FALSE)

> str(emails)

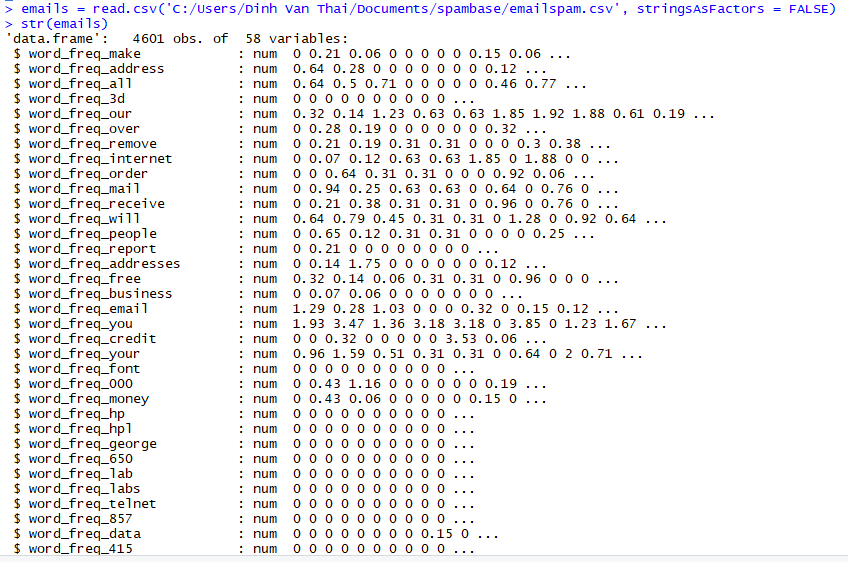
Function kiểm tra giá trị NA

detectNA <- **function**(inp) {

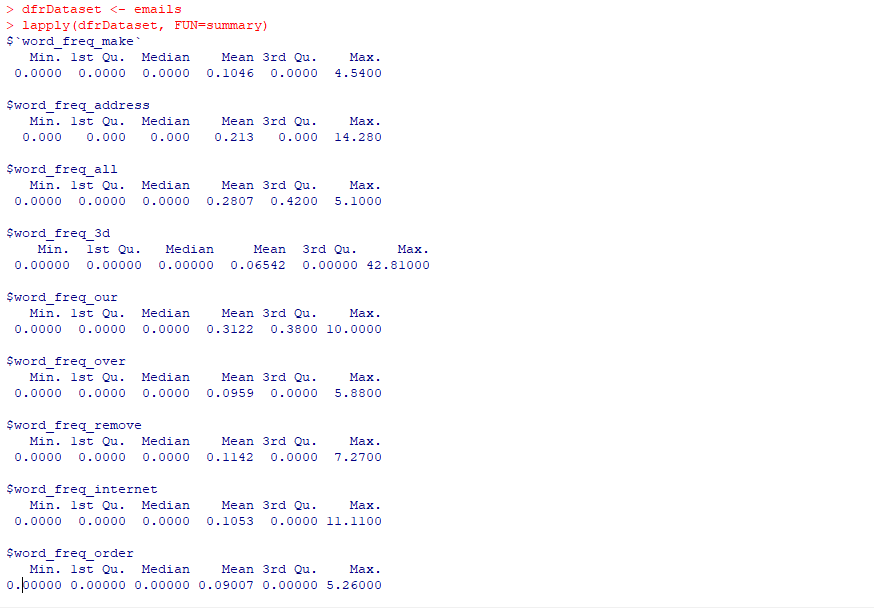
sum(is.na(inp))

}

> lapply(emails, FUN=detectNA)

   
  
  
  
Tóm tắt dữ liệu

> dfrDataset <- emails

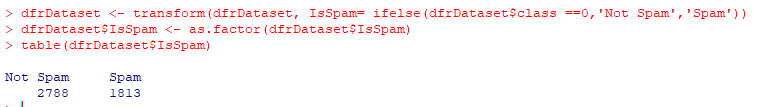
> lapply(dfrDataset, FUN=summary)

Dữ liệu đầu ra

> dfrDataset <- transform(dfrDataset, IsSpam= ifelse(dfrDataset$class ==0,'Not Spam','Spam'))

> dfrDataset$IsSpam <- as.factor(dfrDataset$IsSpam)

> table(dfrDataset$IsSpam)

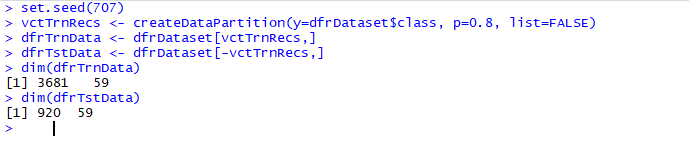


Tiến hành phân chia tập dữ liệu: 80% dữ liệu huấn luyện, 20% dữ liệu kiểm tra

> set.seed(707)

> vctTrnRecs <- createDataPartition(y=dfrDataset$class, p=0.8, list=FALSE)

> dfrTrnData <- dfrDataset[vctTrnRecs,]

> dfrTstData <- dfrDataset[-vctTrnRecs,]

Tạo mô hình với Random Forest, với tham số ntree = 500, mtry = sqrt(số thuộc tính đầu vào)

> set.seed(707)  
> # mtry

> myMtry=sqrt(ncol(dfrTrnData)-2)

> myNtrees=500

> # start time

> vctProcStrt <- proc.time()

> # Random Forest (default)

> mdlRndForDef <- randomForest(IsSpam~.-class, data=dfrTrnData,

+ mtry=myMtry, ntree=myNtrees)  
> # end time

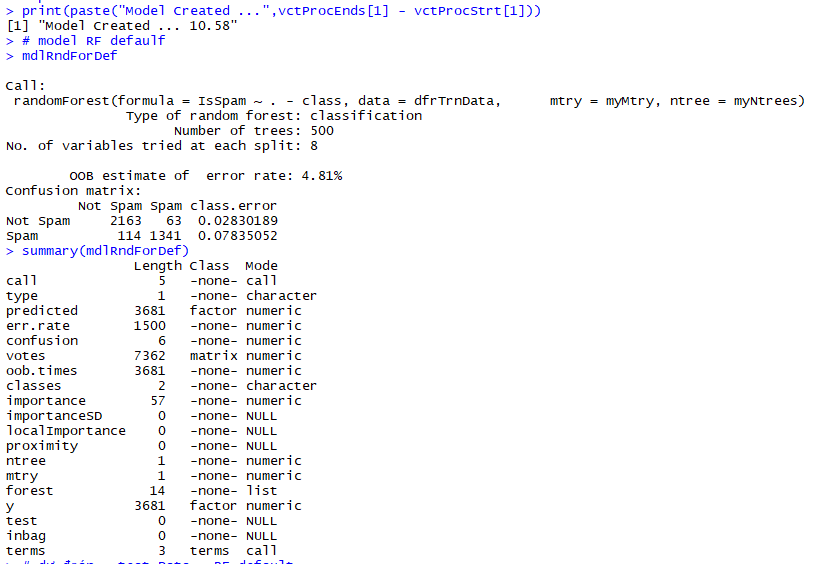
> vctProcEnds <- proc.time()

> # print

> print(paste("Model Created ...",vctProcEnds[1] - vctProcStrt[1]))

> # model RF defaulf

> mdlRndForDef

Như vậy, mô hình Random Forest thực hiện mất 10.58s.

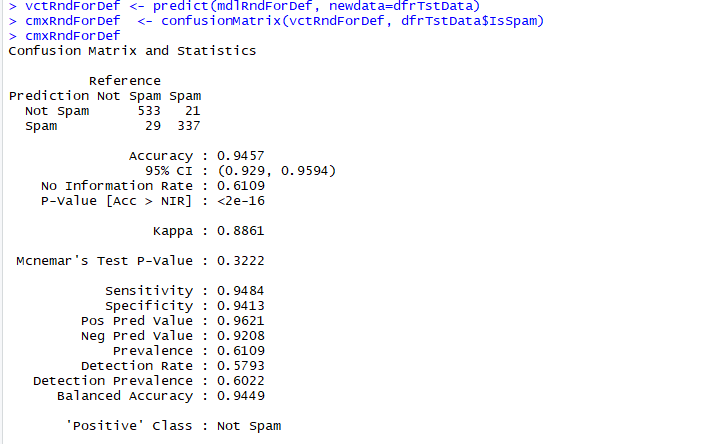
Ước tính tỉ lệ lỗi OOB là 4.81%, như vậy độ chính xác của thuật toán là 95,19%, con số này là rất cao.

Dự đoán với dữ liệu test với Random Forest

> vctRndForDef <- predict(mdlRndForDef, newdata=dfrTstData)

> cmxRndForDef <- confusionMatrix(vctRndForDef, dfrTstData$IsSpam)

> cmxRndForDef

Độ chính xác trên dữ liệu test khoảng hơn 94% là con số rất cao, gần với độ chính xác của mô hình huấn luyện.

Sử dụng xác thực chéo K-Fold để đánh giá mô hình, k-fold = 10

set.seed(707)

# start time

vctProcStrt <- proc.time()

# Random Forest (default)

myControl <- trainControl(method="cv", number=10, repeats=3)

myMetric <- "Accuracy"

myMtry <- sqrt(ncol(dfrTrnData)-2)

myNtrees <- 500

myTuneGrid <- expand.grid(.mtry=myMtry)

mdlRndForRfm <- train(IsSpam~.-class, data=dfrTrnData, method="rf",

verbose=F, metric=myMetric, trControl=myControl,

tuneGrid=myTuneGrid, ntree=myNtrees)

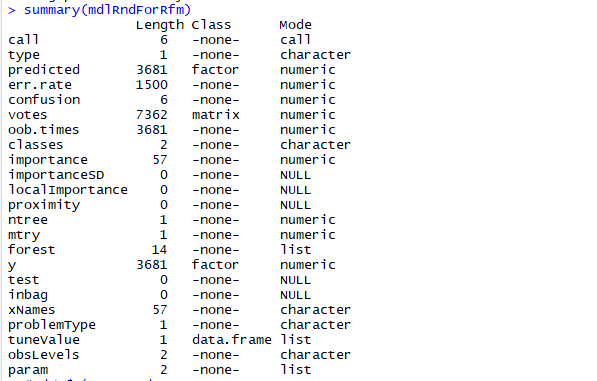
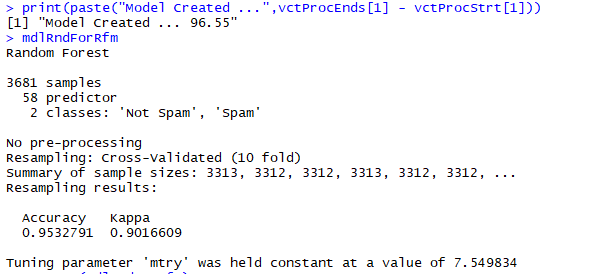
# end time

vctProcEnds <- proc.time()

# print

print(paste("Model Created ...",vctProcEnds[1] - vctProcStrt[1]))

mdlRndForRfm  
summary(mdlRndForRfm)



Độ chính các với xác thực chéo k-fold là khoảng 95,32%, cao hơn so với mô hình Random Forest mặc định.

Dự đoán lại với bộ kiểm tra

vctRndForRfm <- predict(mdlRndForRfm, newdata=dfrTstData)

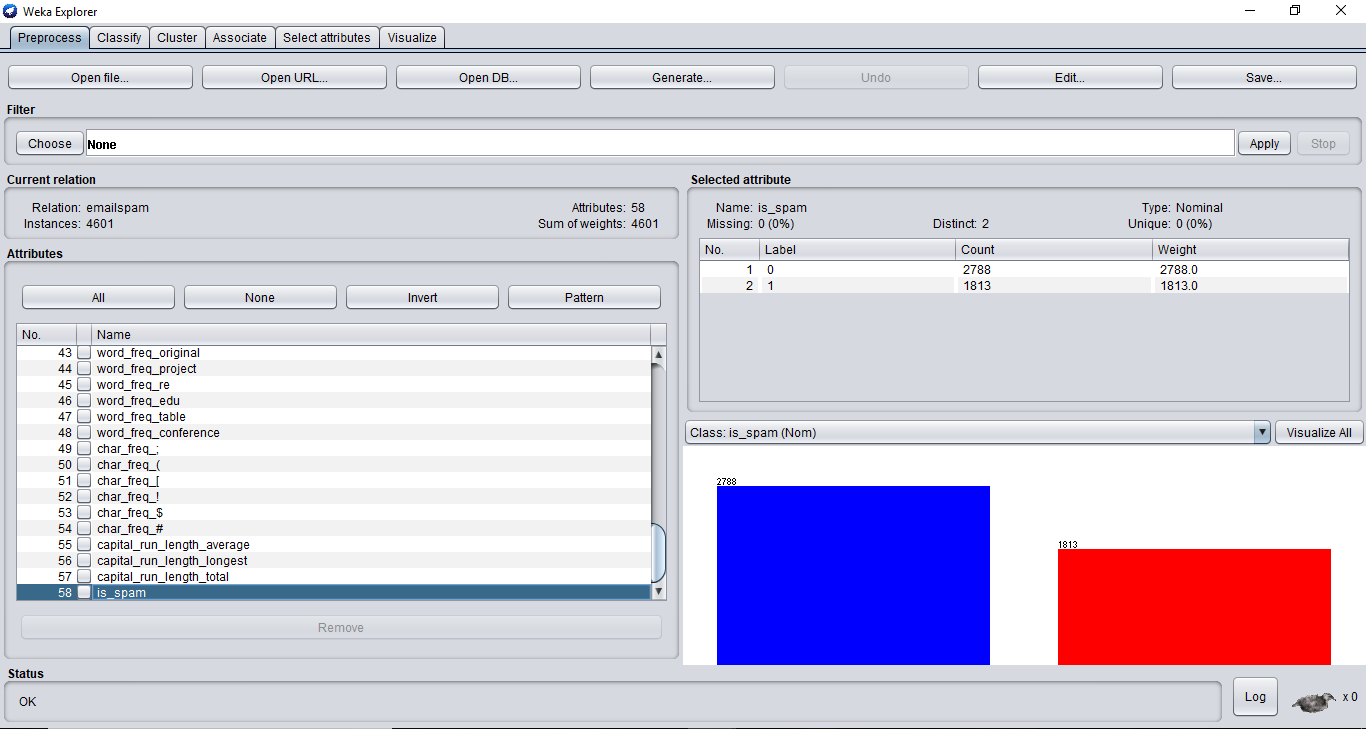
cmxRndForRfm <- confusionMatrix(vctRndForRfm, dfrTstData$IsSpam)

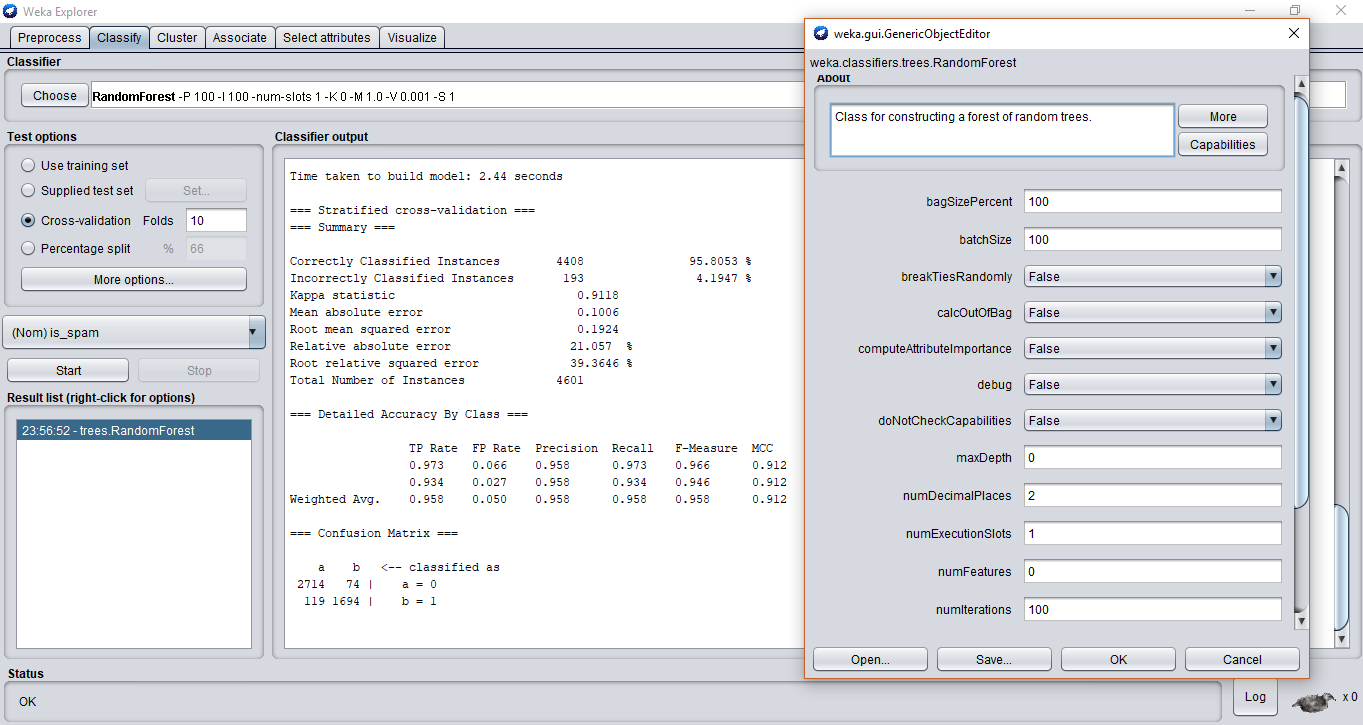
cmxRndForRfm

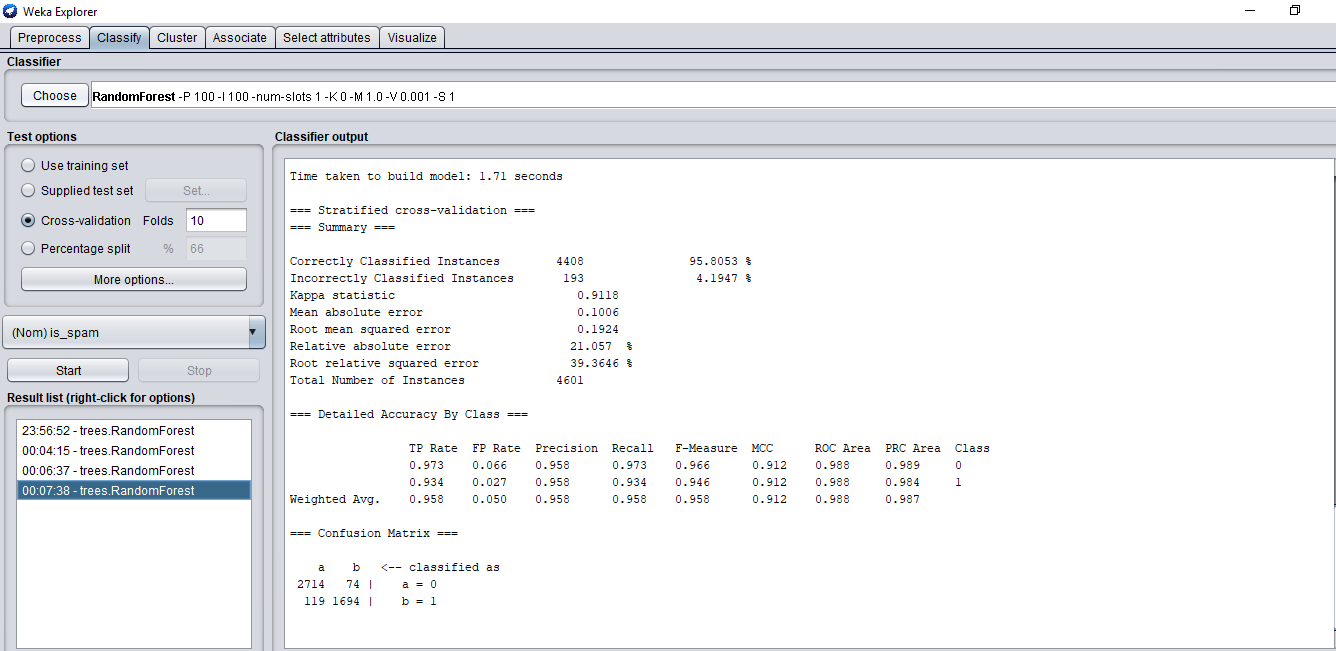
## 3.4 Random Forest với WEKA

Để phục vụ cho đầu vào WEKA, bộ dữ liệu phải được chuẩn hoá trước khi thử nghiệm. Dữ liệu phải được chuyển đổi thành một têp \*arff trước khi cung cấp đầu vào cho WEKA để huấn luyện, kiểm ra và so sánh.

WEKA có một bộ thuật toán máy học để tiền xử dữ liệu, phân loại, hồi quy, phân cụm và quy tắc kết hợp. Kỹ thuật học máy thực hiện trong WEKA rất hữu ích trong việc giải quyết các vấn đề thực tế khác nhau. Bộ công cụ cung cấp một cấu trúc được xác định rõ ràng để các nhà nghiên cứu và nhà phát triển triên thử nghiệm thuật toán, xây dựng và đánh giá mô hình.

Giao diện weka

Random Forest trong weka

Kết quả đầu ra của thuật toán trên môi trường mô phỏng trên WEKA

**Hình 3.1: Kết quả phân lớp với RFs với bộ dữ liệu Spambase**

Đánh giá hiệu suất của thuật toán theo thời gian thực hiện để tạo mô hình, các trường hợp phân loại chính xác và không chính xác. Các kết quả được thể hiện trong bảng 1 sau đây:

|  |  |
| --- | --- |
| Tiêu chí đánh giá |  |
| Time taken to build model(s) | 1.71 |
| Correctly classified instances | 4408 |
| Incorrectly classified instances | 193 |
| Accurary(%) | 95.8053 |

**Bảng 1: Đánh giá hiệu suất của thuật toán Random Forest**

Để thực hiện đánh giá hiệu suất công bằng và tốt hơn cho các thuật toán học máy, lỗi mô phỏng cũng được tính đến. Hiệu quả của những thuật toán được đánh giá bằng các thuật ngữ sau: Kappa Statistics(KS), Mean Absolute Error (MAE), Root Mean Square Error (RMSE), Relative Absolute Error (RAE), Root Relative Squared Absolute Error (RRAE). Các KS, MAE và RAE nằm trong các giá trị số. RAE, RRSE ở tỷ lệ phần trăm. Các kết qảu được thể hiện ở bảng 2.

|  |  |
| --- | --- |
| Tiêu chí đánh giá | |
| Kappa Statistics(KS) | 0.9118 |
| Mean Absolute Error (MAE) | 0.1006 |
| Root Mean Square Error (RMSE) | 0.1924 |
| Relative Absolute Error (RAE) | 21.057 |
| Root Relative Squared Absolute Error (RRAE) | 39.3646 |

**Bảng 2: Lỗi huấn luyện và mô phỏng của thuật toán Random Forest**

Sau khi tạo mô hình dự đoán, hiệu quả của thuật toán Random Forest được đánh giá thể hiện trong bảng 3 dưới đây

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Kỹ thuật | TTR | FPR | Precision | F-Sore | Class |
| RFs | 0.973 | 0.066 | 0.958 | 0.966 | 0 |
| 0.934 | 0.027 | 0.958 | 0.946 | 1 |

**Bảng 3: Đánh giá hiệu suất của thuật toán RFs dựa trên TPF, FPR, Precision, F-Score**

Dưới dây là ma trận nhầm lẫn của thuật toán Random Forest thể hiện trong bảng 4, cũng là một cách thực tế để đánh giá hiệu suất của các bộ phân loại, mỗi hàng của bảng biểu thị tỉ lệ thực tế của lớp trong khi mỗi cột chỉ ra dự đoán.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Non-spam | Spam | Class |
| RFs | 2714 | 74 | Non-spam |
|  | 119 | 1694 | spam |

**Bảng 4: Ma trận nhầm lẫn của thuật toán Random Forest**

Từ bảng 4 trên, Random Rorest dự đoán chính xác 2714 trường hợp trong số 2788 trường hợp của lớp non-spam, 74 trường hợp dự đoán sai (74 trường hợp của lớp non-spam dự đoán là spam). Nó cũng dự đoán được chính xác 1694 trường hơp trong số 1813 trường hợp của lớp spam, 119 trường hợp dự đoán sai (119 trường hợp của lớp spam dự đoán là non-spam).

## 3.5 Kết luận và hướng phát triển đề tài

### 3.5.1 Kết luận

Thông qua những nghiêm cứu về thuật toán Random Forest và ứng dụng, cho thấy Random Forest là một thuật toán có độ chính xác và hiệu suất cao. Qua việc đánh giá hiệu suất và hiệu quả của thuật toán Random Forest trên bộ dữ liệu spambase có thể thấy đươc Random Forest là một thuật toán đầy hứa hẹn để xử lý vấn đề lọc Email Spam giảm khối lượng thư rác trong hộp thư người dùng email, là mối quan tâm lớn của người dùng internet hiện nay.

Chuyên đề này đã hoàn thành một số việc cụ thể như sau:

* Tìm hiểu và cài đặt thuật toán Random Forest bằng Python
* Triển khai thuật toán trên các môi trường R và WEKA
* Đánh giá hiệu suất và hiệu năng của Random Forest trên bộ dữ liệu Spambase

Tuy nhiên trong quá trình hoàn thiện chuyên đề, nhiều kiến thức còn chưa hiểu sâu và có sự thiếu sót, cần cố gắng nghiêm cứu và bổ sung thêm.

### 3.5.2 Hướng phát triển đề tài

* Tiếp tục phát triển đề tài đang nghiên cứu
* Tìm kiếm các kỹ thuật để tăng hiệu suất và hiệu quả của thuật toán
* Thay đổi bộ dữ liệu thử nghiệm
* So sánh RF với các kỹ thuật học máy phân loại khác và đánh giá chúng.
* Triển khai và cài đặt các kỹ thuật trên các bộ công cụ khác

# DANH MỤC CÁC TÀI LIỆU THAM KHẢO

[1] Random Forests Machine Learning Technique for Email Spam Filtering -

E. G. Dada and S. B. Joseph

[2] RANDOM FORESTS - Leo Breiman

[3] Khai phá dữ liệu – TS. Nguyễn Đức Thuần

[4] Nghiên cứu, xây dựng phương pháp trích chọn thuộc tính nhằm làm tăng hiệu quả phân lớp đối với dữ liệu đa chiều - Đồng Thị Ngọc Lan