Hệ Thống Dự Đoán Độ Trễ Chuyến Bay Theo Thời Gian Thực

Võ Minh Trí[1,2,3,4], Trần Triệu Vũ[1,2,3,4],
Phạm Đức Thể[1,2,3,4], and Đỗ trọng Hợp[1,2,3,5]

[1] Đại học Quốc gia TP. HCM, Việt Nam
[2] Đại học Công nghệ Thông tin TP. HCM, Việt Nam
[3] Khoa Khoa học và Kỹ thuật Thông tin
[4] {19522396,19522539,19522253}@gm.uit.edu.vn
[5] hopdt@uit.edu.vn

Tóm tắt nội dung Trễ chuyến bay là một sự cố không mong muốn trong lĩnh vực hàng không nói riêng và vận chuyển nói chung. Dự đoán được khả năng hay mức độ trễ của các chuyến bay đóng vai trò quan trọng trong việc chủ động sắp xếp thời gian cho hãng cũng như tăng độ uy tín của hãng đối với người dùng. Trong báo cáo này, chúng tôi thực hiện xây dựng bộ dữ liệu Flight Delays Dataset. Chúng tôi xây dựng pipeline để xử lý dữ liệu và sử dụng các thuật toán Machine Learning được hỗ trợ bởi Machine Learning Library (MLlib) là một thư viện máy học của Spark để xây dựng mô hình dự đoán cho bài toán và so sánh kết quả đạt được thông qua các độ đo: Precision, Recall, Accuracy và F1–score. Kết quả tốt nhất theo độ đo F1-macro là 45.27% sử dụng Decision Tree. Sau đó, chúng tôi tiến hành xây dựng một hệ thống giả lập dự đoán độ trễ chuyến bay theo thời gian thực.

Keywords: Predict Flight Delays · Logistic Regression · Decision Tree · Random Forest · Naive Bayes · Spark · Kafka · Streaming.

1 Giới Thiệu

Một trong những trải nghiệm tồi tệ nhất khi đi máy bay chính là tình trạng chậm trễ hay thậm chí là bị huỷ chuyến. Công bằng mà nói, khoảng 80% các chuyến bay là đúng giờ - nghĩa là trong vòng 15 phút so với thời gian dự kiến. 20% còn lại thường bị chậm trễ. Quả thật, tình trạng bị delay khiến bất cứ ai trong chúng ta cũng bực mình vì vừa phải chờ đợi lâu lại lỡ biết bao kế hoạch dự định. Có nhiều nguyên nhân khiến cho một chuyến bay khởi hành trễ so với thời gian dự kiến ban đầu như: máy bay về muộn, vấn đề về thời tiết, an ninh, sự cố kỹ thuật, ...

Dự đoán độ trễ chuyến bay là một bài toán dựa trên nền tảng phân tích các thông tin được cung cấp trước khi máy bay khởi hành nhằm trích xuất các thông tin có ích sau đó sử dụng các mô hình toán học để dự đoán độ trễ chuyến bay bay. Bài toán được nghiên cứu nhằm hỗ trợ các hãng máy bay có thể dự đoán trước khả năng và mức độ trễ của một chuyến bay nhằm sẵn sàng để xử

lý các tình huống đặc biệt là thông báo đến khách hàng để họ có thể linh động hơn trong thời gian của mình, giúp tăng trải nghiệm người dùng và cạnh tranh với các hãng hàng không khác.

Trong báo cáo này, trước tiên chúng tôi trình bày phương pháp xây dựng bộ dữ liệu Flight Delays Dataset trong Phần 2. Hướng tiếp cận được mô tả chi tiết trong Phần 3. Trong Phần 4, chúng tôi tiến hành xây dựng pipeline thực nghiệm các mô hình Machine Learning trên bộ dữ liệu và phân tích kết quả để tìm ra mô hình tốt nhất và các hạn chế của mô hình. Tiếp theo, chúng tôi tiến hành xây dựng hệ thống giả lập dự đoán độ trễ chuyến bay theo thời gian thực trong Phần 5. Cuối cùng, chúng tôi rút ra kết luận ở Phần 6.

2 Xây Dựng và Phân Tích Bộ Dữ Liệu

2.1 Tạo Bộ Dữ Liệu

Bộ dữ liệu sử dụng trong bài toán dự đoán độ trễ chuyến bay được lấy từ website Bureau of Transportation Statistics (BTS) [6]. BTS là một bộ phận thuộc Department of Transportation (DOT) là nguồn thống kê ưu việt về hàng không thương mại, hoạt động vận tải hàng hóa đa phương thức và kinh tế vận tải, đồng thời cung cấp bối cảnh cho các nhà hoạch định và công chúng để hiểu số liệu thống kê về giao thông vận tải. BTS cung cấp thông tin dữ liệu của các chuyến bay từ năm 1987 đến hiện tại [7]. Dữ liệu về thông tin các chuyến bay được cung cấp theo từng tháng với trung bình gần nửa triệu thông tin chuyến bay trong một tháng. Vì số lượng dữ liệu khá lớn và bị giới hạn về phần cứng (RAM & ROM không đủ) nên chúng tôi chỉ sử dụng một vài tháng đại diện cho các quý trong năm để tạo ra bộ dữ liệu huấn luyện mô hình. Chúng tôi sử dụng dữ liệu thông tin các chuyến bay của tháng 6/9/12–2021, tháng 3/2022 để huấn luyện mô hình và dữ liệu thông tin các chuyến bay của tháng 4/2022 để treaming giả lập real–time cho hệ thống.

Chúng tôi đã thu thập dữ liệu của 4 tháng kể trên và thu được bộ dữ liệu chứa 2,200,913 dữ liệu thông tin các chuyến bay và có nhiều chuyến bay bị missing values. Vì bộ dữ liệu có missing values, nên chúng tôi tiến hành các bước tiền xử lý để xử lý missing values. Với các chuyến bay có missing values, chúng tôi tôi sẽ xóa bỏ thông tin của chuyến bay đó. Thuộc tính DEP_DELAY được chúng tôi sử dụng làm thuộc tính dự đoán cho bài toán, nó chứa các giá trị liên tục là sự chênh lệch về số phút giữa thời gian khỏi hành dự kiến và thực tế của các chuyến bay. Để phù hợp với mục tiêu ban đầu và đơn giản hóa bài toán chúng tôi tiến hành chuẩn hóa thuộc tính DEP_DELAY thành 3 nhãn như sau: với các chuyến bay có thời gian khởi hành thực tế sớm hoặc đúng giờ sẽ được gán nhãn "0", các chuyến bay có thời gian khởi hành thực tế muộn không quá 30 phút sẽ được gán nhãn "1", còn lại các chuyến bay có thời gian khởi hành thực tế muộn hơn 30 phút hoặc có thể bị hủy chuyến sẽ được gán nhãn "2". Thuộc tính DEP_DELAY

[6] Dữ liệu được thu thập tại: Reporting Carrier On-Time Performance

[7] Dữ liệu mới nhất sẽ cách hiện tại 3 tháng. Ví dụ: Hiện tại là tháng 7 thì dữ liệu mới nhất được cung cấp là dữ liệu trong tháng 4.

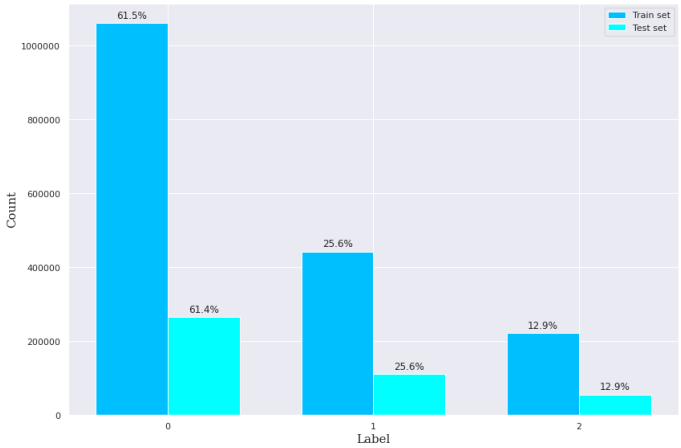
cũng được chúng tôi đổi tên thành LABEL để phù hợp với dữ liệu và bài toán. Sau khi xử lý xong, chúng tôi thu được bộ dữ liệu *Flight Delays Dataset* chứa 2,157,737 dòng dữ liệu và 11 thuộc tính là các thông tin chuyển bay. Tương tự, chúng tôi tạo tiếp bộ dữ liệu để streaming giả lập real–time cho hệ thống chứa 542,117 dòng dữ liệu và 11 thuộc tính. Thông tin chi tiết các thuộc tính của bộ dữ liệu được thể hiện tại Bảng 1.

Index	**Thuộc tính**	**Ý nghĩa**
0	ID	ID của chuyến bay.
1	QUARTER	Quý trong năm (1-4).
2	MONTH	Tháng trong năm (1-12).
3	DAY_OF_MONTH	Ngày trong tháng (1-31).
4	DAY_OF_WEEK	Ngày trong tuần (1-7).
5	OP_UNIQUE_CARRIER	Mã nhà cung cấp dịch vụ.
6	ORIGIN	Sân bay xuất phát.
7	DEST	Sân bay đến.
8	DISTANCE	Khoảng cách giữa các sân bay (dặm).
9	CRS_DEP_TIME	Giờ khởi hành dự kiến.
10	LABEL	Output của bài toán, gồm 3 nhãn: – 0: Không trễ. – 1: Trễ không quá 30 phút. – 2: Trễ hơn 30 phút hoặc hủy chuyến.

Bảng 1: Thông tin chi tiết của các thuộc tính.

2.2 Phân Tích Bộ Dữ Liệu

Hình 1 thể hiện phân phối và tỉ lệ của ba nhãn trong tập train/test của bộ dữ liệu Flight Delays Dataset của chúng tôi. Ta có thể thấy phân phối các nhãn trên tập train và test là tương đồng nhau với các tỉ lệ nhãn 0 – 1 – 2 lần lượt là 61.5% – 25.6% – 12.9%. Bộ dữ liệu của chúng tôi có xu hướng bị *mất cân bằng dữ liệu (imbalance data)* khi mà số lượng nhãn 0 chiếm 61.5% của bộ dữ liệu, gấp 2.4 lần nhãn 1 (25.6%) và gấp 4.8 lần nhãn 2 (12.9%). Đây cũng là một thách thức của bộ dữ liệu của chúng tôi.



Hình 1: Thống kê tập dữ liệu Train/Test.

3 Hướng Tiếp Cận

Từ bộ dữ liệu Flight Delays Dataset mà chúng tôi đã tạo ra ở Phần 2, chúng tôi thực hiện các bước chuẩn hóa dữ liệu trước khi huấn luyện mô hình như: String Indexer, One Hot Encoding, Vector Assembler, Data Standardization (được trình bày rõ hơn tại Phần 3.1). Để tạo ra mô hình dự đoán độ trễ chuyến bay, chúng tôi tiến hành xây dựng các mô hình *Máy Học (Machine Learning)* như: Logistic Regression, Decision Tree Classifier, Random Forest Classifier, Naive Bayes được hỗ trợ bởi Machine Learning Library (MLlib) là một thư viện máy học của Spark. Để đánh giá mô hình chúng tôi sử dụng 4 độ đo là: Precision, Recall, Accuracy và F1–score (chi tiết Phần 3.3). Từ việc đánh giá mô hình, chúng tôi sẽ chọn ra mô hình tốt và khả thi nhất để tiến hành xây dựng hệ thống giả lập real–time dự đoán độ trễ chuyến bay của chúng tôi.

3.1 Data Normalization

Chuẩn hóa dữ liệu (Data Normalization) là một khái niệm chung đề cập đến hành động chuyển đổi các giá trị ban đầu của tập dữ liệu thành các giá trị mới. Các giá trị mới thường được mã hóa liên quan đến chính tập dữ liệu và được chia tỷ lệ theo một cách nào đó.

String Indexer Mã hóa một cột chuỗi nhãn thành một cột chỉ số nhãn. String Indexer có thể mã hóa nhiều cột. Nếu cột đầu vào là số, nó sẽ được ép kiểu thành kiểu chuỗi và lập chỉ mục các giá trị của chuỗi. Các chỉ số nằm trong [0, numLabels). Theo mặc định, điều này được sắp xếp theo tần số nhãn vì vậy nhãn thường xuyên nhất nhận được chỉ số 0.

One Hot Encoding Ánh xạ một đặc trưng phân loại (categorical feature), được biểu thị dưới dạng nhãn chỉ mục thành một vector nhị phân có nhiều nhất một giá trị duy nhất cho biết sự hiện diện của một giá trị đặc trưng cụ thể trong số tất cả các giá trị đặc trưng. Mã hóa này cho phép các thuật toán mong đợi các đặc trưng liên tục chẳng hạn như Logistic Regression sử dụng các đặc trưng phân loại. Đối với dữ liệu đầu vào kiểu chuỗi, người ta thường mã hóa các đặc trưng phân loại bằng cách sử dụng String Indexer trước. OneHotEncoder có thể chuyển đổi nhiều cột, trả về một cột vector đầu ra được one-hot-encoded cho mỗi cột đầu vào. Thông thường, hợp nhất các vector này thành một vector đặc trưng duy nhất bằng cách sử dụng Vector Assembler.

Vector Assembler Là một cách biến đổi kết hợp một danh sách các cột nhất định thành một cột vector duy nhất. Nó rất hữu ích để kết hợp các đặc trưng thô và các đặc trưng được tạo bởi các biến đổi đặc trưng khác nhau thành một vector đặc trưng duy nhất, để đào tạo các mô hình ML như Logistic Regression và Decision Trees. Vector Assembler chấp nhận các kiểu cột đầu vào sau: tất cả các kiểu số, kiểu boolean và kiểu vector. Trong mỗi hàng, giá trị của các cột đầu vào sẽ được nối thành một vector theo thứ tự được chỉ định.

Data Standardization Là một loại kỹ thuật chuẩn hóa cụ thể. Nó đôi khi được gọi là z-score normalization. z-score còn gọi là điểm tiêu chuẩn, là giá trị được biến đổi cho mỗi điểm dữ liệu. Để chuẩn hóa tập dữ liệu bằng cách sử dụng standardization, chúng ta lấy mọi giá trị x_i bên trong tập dữ liệu và biến đổi nó thành giá trị z_i tương ứng bằng công thức sau:

$$z_i = \frac{x_i - \mu_k}{\sigma_k} \tag{1}$$

Trong đó: μ_k và σ_k lần lượt là trung bình và độ lệch chuẩn của từng thuộc tính thứ k. Quá trình standardization này chuyển giá trị trung bình của tập dữ liệu thành 0 và độ lệch chuẩn của nó thành 1.

3.2 Machine Learning Algorithm

Logistic Regression *Hồi quy logistic (Logistic Regression – LR)* là một quá trình mô hình hóa xác suất của một kết quả rời rạc cho một biến đầu vào. Hồi quy logistic là một phương pháp phổ biến để dự đoán một bài toán phân loại. Đây là một trường hợp đặc biệt của *mô hình Tuyến tính Tổng quát (Generalized Linear models*) dự đoán xác suất của các kết quả. Trong hồi quy logistic spark.ml có thể được sử dụng để dự đoán kết quả phân loại nhị phân bằng cách sử dụng *hồi quy logistic nhị thức (binomial logistic regression)* hoặc nó có thể được sử dụng để dự đoán kết quả phân loại đa lớp bằng cách sử dụng *hồi quy logistic đa thức (multinomial logistic regression)* [1].

Phân loại đa lớp (Multiclass classification) được hỗ trợ thông qua hồi quy logistic đa thức (softmax). Trong hồi quy logistic đa thức, thuật toán tạo ra **K**

bộ hệ số hoặc ma trận có kích thước $\mathbf{K} \times \mathbf{J}$ trong đó $\mathbf{K}$ là số lớp kết quả và $\mathbf{J}$ là số đặc trưng. Nếu thuật toán phù hợp với một số hạng *chặn (intercept)* thì độ dài $\mathbf{K}$ vector của các điểm chặn có sẵn.

Xác suất có điều kiện của các lớp kết quả $k \in 1, 2 \ldots,$ $\mathbf{K}$ được mô hình hóa bằng cách sử dụng hàm softmax.

$$P(Y = K|\mathbf{X}, \beta_k, \beta_{0k}) = \frac{e^{\beta_k \cdot \mathbf{X} + \beta_{0k}}}{\sum_{k'=0}^{K-1} e^{\beta_{k'} \cdot \mathbf{X} + \beta_{0k'}}} \tag{2}$$

Decision Tree Classifier *Cây Quyết Định (Decision Tree – DT)* là một họ phổ biến của các phương pháp phân loại và hồi quy. Cây quyết định và tập hợp của chúng là các phương pháp phổ biến cho các tác vụ phân loại và hồi quy của máy học. Cây quyết định được sử dụng rộng rãi vì chúng dễ diễn giải, xử lý các đặc trưng phân loại, mở rộng sang cài đặt phân loại đa lớp, không yêu cầu tỷ lệ đối tượng và có thể nắm bắt các tương tác đối tượng và phi tuyến tính.

Cây quyết định là một thuật toán tham lam thực hiện phân vùng nhị phân đệ quy của không gian đặc trưng. Cây dự đoán cùng một nhãn cho mỗi phân vùng (lá) dưới cùng. Mỗi phân vùng được chọn một cách tham lam bằng cách chọn phần tách tốt nhất từ một tập hợp các phần tách có thể có, để tối đa hóa *độ lợi thông tin (information gain)* tại một nút cây. Nói cách khác, sự phân tách được chọn tại mỗi nút cây được chọn từ tập hợp:

$$\arg\max_{x} IG(D, s) \tag{3}$$

Trong đó $IG(D, s)$ là độ lợi thông tin khi một sự phân chia s được áp dụng cho tập dữ liệu D.

Random Forest Classifier *Rừng ngẫu nhiên (Random Forest – RF)* là một nhóm phương pháp phân loại và hồi quy phổ biến. Rừng ngẫu nhiên là quần thể của các cây quyết định. Rừng ngẫu nhiên kết hợp nhiều cây quyết định, tuy nhiên mỗi cây quyết định sẽ khác nhau (có yếu tố random). Sau đó kết quả dự đoán được tổng hợp từ các cây quyết định, nhãn của đầu ra là nhãn được dự đoán nhiều nhất (chiếm đa số) trong tất cả các cây quyết định để giảm nguy cơ bị *quá khớp (overfitting)*. Việc triển khai spark.ml hỗ trợ rừng ngẫu nhiên để phân loại nhị phân và đa lớp và cho hồi quy, sử dụng cả các đặc trưng liên tục và phân loại.

Naive Bayes *Naive Bayes (NB)* là một thuật toán phân loại dựa trên tính toán xác suất áp dụng định lý *Bayes*. Thuật toán này thuộc nhóm *học có giám sát (supervised learning)*. Đây là hướng tiếp cận phân lớp theo mô hình xác suất. Dự đoán xác suất một đối tượng mới thuộc về thành viên của lớp đang xét. Naive Bayes có thể được huấn luyện rất hiệu quả. Với một lần huấn luyện qua dữ liệu, nó sẽ tính toán phân phối xác suất có điều kiện của từng đặc trưng cho

từng nhãn. Để dự đoán, nó áp dụng định lý Bayes để tính toán phân phối xác suất có điều kiện của mỗi nhãn cho một quan sát.

Theo định lý Bayes, ta có công thức tính xác suất ngẫu nhiên của sự kiện Y khi biết X như sau:

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} \tag{4}$$

Giả sử ta phân chia 1 sự kiện $\mathbf{x}$ thành n thành phần khác nhau $x_1, x_2, \ldots, x_n$. Naive Bayes theo đúng như tên gọi dựa vào một giả thiết ngây thơ rằng $x_1, x_2, \ldots, x_n$ là các thành phần độc lập với nhau. Từ đó ta có thể tính được:

$$P(\mathbf{x}|y) = P(x_1 \cap x_2 \cap \cdots \cap x_n|y) = P(x_1|y)P(x_2|y)\ldots P(x_n|y) \tag{5}$$

Do đó ta có:

$$P(\mathbf{x}|y) \propto P(y)\prod_{i=1}^{n} P(x_i|y) \tag{6}$$

Trên thực tế thì ít khi tìm được dữ liệu mà các thành phần là hoàn toàn độc lập với nhau. Tuy nhiên giả thiết này giúp cách tính toán trở nên đơn giản, training data nhanh, đem lại hiệu quả bất ngờ với các lớp bài toán nhất định.

3.3 Độ Đo Đánh Giá

		Predicted annotation	
		Positive	Negative
Gold annotation	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Bảng 2: Confusion matrix

Precision & Recall Hai độ đo được sử dụng để đo lường hiệu suất của hệ thống truy xuất là *precision* và *recall*. Precision đo lường số lượng các trường hợp đúng được truy xuất chia cho tất cả các trường hợp được truy xuất, xem Công thức 7. Recall đo lường số lượng các trường hợp đúng được truy xuất chia cho tất cả các trường hợp đúng, xem Công thức 8 [2].

$$Precision : P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{7}$$

$$Recall : R = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{8}$$

F1–score F–score được định nghĩa là giá trị trung bình có trọng số precision và recall tùy thuộc vào hàm trọng số β, xem Công thức 9. F1–score có nghĩa là trung bình hài hòa giữa precision và recall, xem Công thức 10, khi nó được viết là F–score, nó thường có nghĩa là F1–score. F–score còn được gọi là F–Measure. F1–score có thể có các chỉ số khác nhau tạo ra các trọng số khác nhau về precision và recall [2].

$$F - score : F_\beta = (1 + \beta^2) \times \frac{P \times R}{\beta^2 \times P + R} \tag{9}$$

Với $\beta = 1$, F–score tiêu chuẩn thu được, xem Công thức 9.

$$F - score : F_1 = F = 2 \times \frac{P \times R}{P + R} \tag{10}$$

Accuracy Là một độ đo khác được định nghĩa là tỷ lệ các trường hợp đúng được truy xuất, cả positive và negative, trong số tất cả các trường hợp được truy xuất. Accuracy là một trung bình có trọng số của precision và precision nghịch đảo. Accuracy càng cao, mô hình càng hiệu quả. Tuy nhiên điều này không đúng với các bài toán có bộ dữ liệu bị mất cân bằng, xem Công thức 11 [2].

$$Accuracy : A = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{11}$$

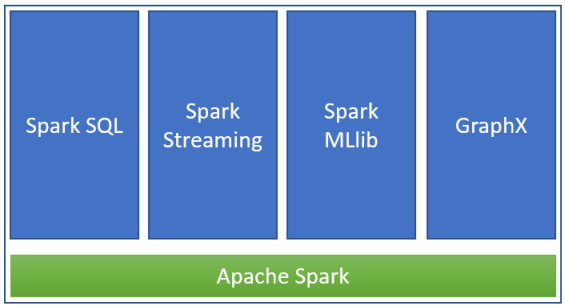
3.4 BigData Frameworks

Hiện nay dữ liệu được tạo ra ngày càng nhiều bằng những cách gián tiếp hoặc trực tiếp. Cùng với sự phát triển của các thiết bị công nghệ thì chúng ta ngày càng có thể thu thập dữ liệu bằng nhiều cách khác nhau điển hình là từ mạng xã hội, các thiết bị IOT trong cuộc sống hằng ngày âm thầm thu thập dữ liệu của chúng ta khiến cho dữ liệu ngày càng phì đại. Dữ liệu lớn có năm đặc trưng (5V's of Bigdata [8]) cụ thể để nhận biết, đó là: khối lượng (volumn), tính đa dạng (variety), giá trị (value), vận tốc (velocity) và tính xác thực (veracity). Dữ liệu lớn hiện nay có rất nhiều ứng dụng khác nhau, từ ngân hàng và chứng khoán đến truyền thông hoặc giáo dục, nhưng làm thế nào để sử dụng dữ liệu lớn là một vấn đề mà các công cụ và kỹ thuật truyền thống không thể xử lý được. Đây là nguyên nhân tại sao một số framework được sinh ra để làm việc với dữ liệu lớn. Một trong những framework mạnh mẽ là Apache Spark mà chúng tôi sử dụng cho bài toán này: dự đoán độ trễ của chuyến bay.

Apache Spark là một framework xử lý dữ liệu mã nguồn mở quy mô lớn, được phát triển ban đầu vào năm 2009 bởi AMPLab. Sau này, Spark đã được trao cho Apache Software Foundation vào năm 2013 và được phát triển cho đến nay. chúng ta hay được quen thuộc với việc sử dụng Spark với 2 ngôn ngữ là Scala và Python (trên Python sử dụng Pyspark). Hiện này Spark được quan tâm khi xử lý dữ liệu lớn, làm việc với machine learning rất tốt vì có sức mạnh tính toán

[8] Xem chi tiết tại: https://www.ibm.com/blogs/watson-health/the-5-vs-of-big-data/

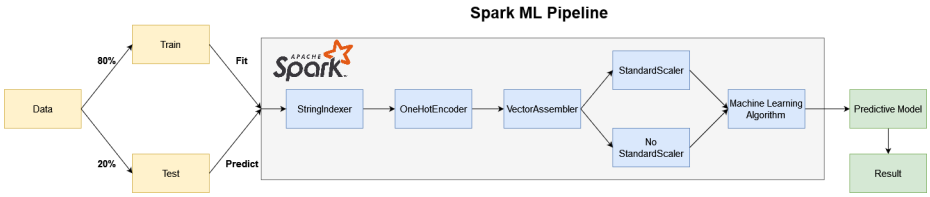
khổng lồ dễ xử lý dữ liệu lớn. Spark còn cung cấp nhiều API dễ dùng dễ giảm đi gánh nặng cho các nhà phát triển trong việc tính toán phân tán hay xử lý bigdata. Hình 2 dưới đây minh họa các thành phần và cấu trúc của Spark.



Hình 2: Hệ thống Apache Spark

4 Thực Nghiệm và Kết Quả

Chúng tôi tiến hành xây dựng một pipeline thực nghiệm các mô hình Machine Learning (ML): Logistic Regression (LR), Decision Tree (DT), Random Forest (RF), Naive Bayes (NB) sử dụng Framework Spark ML (Hình 3).



Hình 3: Spark ML pipeline.

Để tránh tính ngẫu nhiên của kết quả thực nghiệm, chúng tôi thực hiện 5 lần chạy và ghi lại kết quả, sau đó tính kết quả trung bình. Bảng 3 trình bày kết quả so sánh hiệu suất của các bộ hình ML trên tập training và testing khi không sử dụng phương pháp chuẩn hóa dữ liệu và khi sử dụng phương pháp chuẩn hóa dữ liệu là z-score. Vì bộ dữ liệu có xu hướng bị mất cân bằng dữ liệu nên để so sánh hiệu suất của các mô hình chúng tôi dựa vào độ đo F1-macro để đánh giá. Mô hình LR cho kết quả giống nhau khi không áp dụng chuẩn hóa dữ liệu và có áp dụng chuẩn hóa dữ liệu bằng z-score, kết quả trên tập training và testing khá tương đồng nhau (F1-macro train = 36.23% và F1-macro test = 36.25%).

	Scale = None				Scale = StandardScaler			
	Train set		Test set		Train set		Test set	
	Accuracy	F1-macro	Accuracy	F1-macro	Accuracy	F1-macro	Accuracy	F1-macro
LR	63.34	36.23	63.31	36.25	63.34	36.23	63.31	36.25
DT	**67.24**	**49.04**	**64.95**	**45.27**	**66.56**	**47.34**	**64.86**	**44.58**
RF	61.53	25.63	61.49	25.60	61.90	27.15	61.86	27.14
NB	24.60	24.83	24.64	24.86	44.96	38.75	44.73	38.50

Bảng 3: Kết quả trung bình trên 5 lần chạy.

Mô hình DT khi không áp dụng chuẩn hóa dữ liệu cho kết quả cao hơn khi áp dụng chuẩn hóa dữ liệu bằng z-score (cao hơn 1.7% ở tập training và 0.69% ở tập testing), kết quả chênh lệch giữa tập traing và testing khá lớn, từ 2.67% – 3.77%. Mô hình RF khi không áp dụng chuẩn hóa dữ liệu cho kết quả thấp hơn khi áp dụng chuẩn hóa dữ liệu bằng z-score (thấp hơn 1.52% ở tập training và 1.54% ở tập testing), kết quả chênh lệch giữa tập traing và testing là không đáng kể, từ 0.01% – 0.03%. Mặc dù mô hình RF được huấn luyện trên cả 3 nhãn "0", "1" và "2", tuy nhiên mô hình không dự đoán được nhãn "2" của bộ dữ liệu, tức là là kết quả dự đoán của mô hình chỉ có nhãn "0" và "1" ở cả hai tập training và testing. Mô hình NB cho kết quả cao vượt trôi khi áp dụng phương pháp chuẩn hóa dữ liệu là z-score (cao hơn 13.92% ở tập training và 13.64% ở tập testing), kết quả chênh lệch giữa tập training và testing là không đáng kể, từ 0.03% – 0.25%. Tóm lại, mô hình DT không áp dụng chuẩn hóa dữ liệu cho kết quả tốt nhất (F1-macro test = 45.27%), tiếp theo là đến mô hình NB có áp dụng chuẩn hóa dữ liệu bằng z-score (F1-macro test = 38.50) và mô hình RF là mô hình tệ nhất (F1-macro test = 27.14% và không dự đoán được nhãn "2").

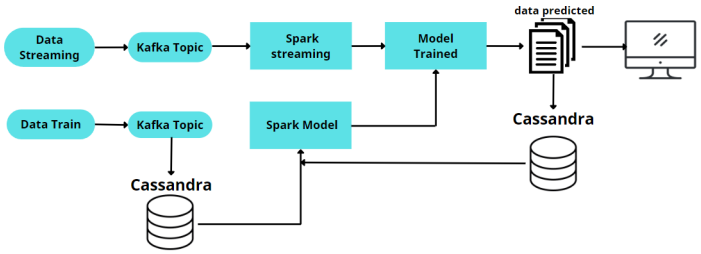
LABEL	**Precision**	**Recall**	**F1-score**
0	68.32	92.52	78.60
1	47.30	22.20	30.22
2	50.08	18.47	26.99

Bảng 4: Kết quả trên mỗi lớp của mô hình tốt nhất trên tập test.

Kết quả chi tiết trên từng nhãn của mô hình DT không áp dụng chuẩn hóa dữ liệu (mô hình tốt nhất) được thể hiện trong Bảng 4. Kết quả giảm dần theo thứ tự nhãn "0", nhãn "1", nhãn "2". Chỉ có một nhãn có F1-score cao trên 70% (nhãn "0" – 78.60%), còn 2 nhãn còn lại có F1-score tương đối thấp (nhãn "1" – 30.22% và nhãn "2" – 26.99%). Kết quả này giải thích sự mất cân bằng dữ liệu khi số lượng nhãn "0" chiếm 61.5% (nhãn "2" chỉ chiếm 12.9% trong bộ dữ liệu Flight Delays Dataset, chi tiết trong hình 1).

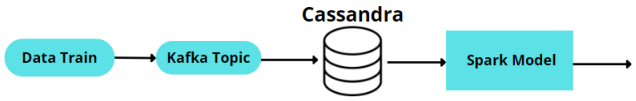
5 Kiến trúc hệ thống

Kiến trúc hệ thống được chia làm ba giai đoạn chính, được thể hiện trong Hình 4.



Hình 4: Kiến trúc tổng quát hệ thống Flight Delay Prediction

5.1 Giai Đoạn Xây Dựng Lựa Chọn Mô Hình

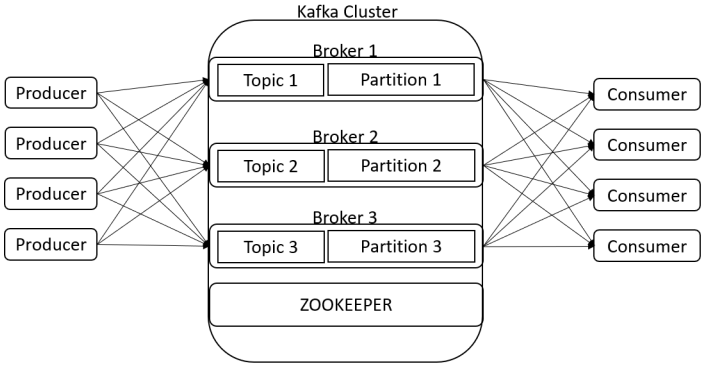


Hình 5: Giai Đoạn Xây Dựng Lựa Chọn Mô Hình

Dữ liệu training được tổng hợp từ nhiều nguồn khác nhau sẽ được Producer của Kafka gửi vào topic Kafka. Sau đó Consumer có vai trò chuyển nhận dữ liệu và truyền dữ liệu trực tiếp vào cơ sở dữ liệu Cassandra để lưu trữ. Tiếp đến dữ liệu được load từ Cassandra để tiến hành huấn luyện các mô hình và chọn ra mô hình tốt nhất để sử dụng làm mô hình chính của hệ thống này. Hình 5 thể hiện đầy đủ dữ liệu ở từng bước nhỏ. Các bước xử lý dữ liệu, thực nghiệm huấn luyện mô hình ở Phần 4 cho thấy mô hình Decision Tree đạt kết quả tốt nhất, nên chúng tôi sẽ sử dụng mô hình Decision Tree cho hệ thống này.

Apache Kafka [5] Có thể hiểu đơn giản Kafka là một công cụ để stream dữ liệu phân tán (dữ liệu từ nhiều nguồn khác nhau). Cụ thể thì Kafka là Framework mã nguồn mở, đã được đóng gói hoàn chỉnh, khả năng chịu lỗi cao và là hệ thống nhắn tin nhanh. Chính nhờ sự đáng tin cậy của nó, Kafka đang dần được thay thế cho hệ thống nhắn tin truyền thống. Nó được sử dụng cho các hệ thống nhắn tin thông thường trong các ngữ cảnh khác nhau. Đây là hệ quả khi khả năng

mở rộng theo chiều ngang và chuyển giao dữ liệu đáng tin cậy là những yêu cầu quan trọng nhất. Một vài trường hợp sử dụng Kafka: Stream Processing, Log Aggregation, Website Activity Monitoring, Metrics Collection.



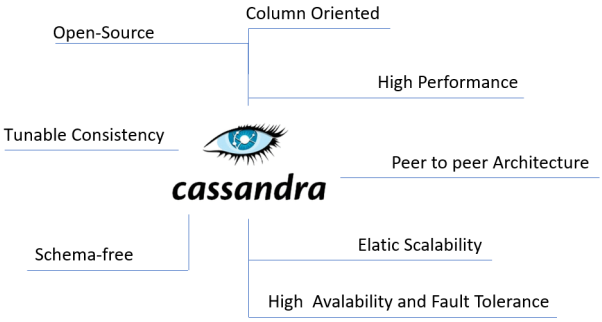
Hình 6: Sơ lượt về Apache kafka

Cấu trúc của Kafka: Dựa vào Hình 6 có thể thấy rằng: Producer có vai trò như một người gửi tin nhắn, tin nhắn sẽ được đưa vào một topic cụ thể. Consumer đóng vai trò là người nhận tin nhắn, tin nhắn để được chuyển từ topic vào customer để thực hiện các yêu cầu tiếp theo từ người dùng Kafka. Topic là một tên nguồn cấp dữ liệu mà nơi đó lưu trữ dữ liệu từ Producer chuyển tới. Partition là các đoạn được các topic chia nhỏ. Broker: Kafka cluster là một tập hợp các server, mỗi một tập hợp này được gọi là 1 broker. ZooKeeper dùng để quản lý và bố trí các Kafka Broker.

Apache Cassandra [4] Nói về phần mềm trung gian NoSQL dẫn đầu là Google BigTable và Amazon Dynamo, nhưng nhiều thứ khác cũng xuất hiện trong thế giới open source. Trong số đó, Apache Cassandra (Cassandra) đang thu hút sự chú ý đặc biệt gần đây.

Cassandra ban đầu được tạo ra bởi Facebook (hiện tại đã đổi tên thành Meta). Sau đó nó đã được tặng cho Apache Foundation vào tháng 2 năm 2010 và được nâng cấp lên thành dự án hàng đầu của Apache. Cassandra là một cơ sở dữ liệu phân tán kết hợp mô hình dữ liệu của Google Bigtable với thiết kế hệ thống phân tán như bản sao của Amazon Dynamo. Hình 7 cho thấy các đặt điểm của Cassandra.

Các tính năng ưu việt của Cassandra bao gồm các điểm sau: Thích hợp để sử dụng thực tế, có khả năng chịu lỗi cao, kiến trúc không có SPOF (Single point of failure), mức độ tự do kiểm soát nhất quán, mô hình dữ liệu phong phú, Fast Linear-scale Performance, Cassandra tăng thông lượng vì chúng tạo điều kiện cho chúng ta tăng số lượng nút trong cụm. Do đó, Cassandra duy trì thời gian phản hồi nhanh chóng. Tính khả dụng cao, hỗ trợ các ngôn ngữ khác nhau dưới

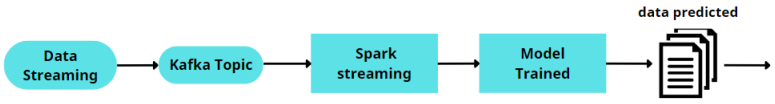


Hình 7: Sơ lượt về Apache Cassandra

dạng client code, dễ dàng nắm bắt trạng thái bên trong của máy chủ bằng JMX (Java Management Extensions).

Vì Bigdata có khối lượng rất lớn nên việc lưu trữ và quản lý dữ liệu trở nên khó khăn, nhờ vào các tính năng vượt trội của Cassandra chúng tôi sử dụng nó để quản lý dữ liệu lớn này. Mục đích của chúng tôi sử dụng Cassandra trong hệ thống này còn vì muốn tận dụng khả năng multi node, khi hai hay nhiều máy đều chứa dữ liệu cần thì có thể truy cập P2P (peer-to-peer) để truy vấn dữ liệu qua lại, từ đó có thể lấy dữ liệu từ nhiều node khác nhau, đó là bản chất của dữ liệu lớn vì nó có khối lượng rất lớn nên khả năng lưu trữ phân tán cao, dữ liệu khó mà tập trung tại 1 node.

5.2 Giai Đoạn Stream Dữ Liệu

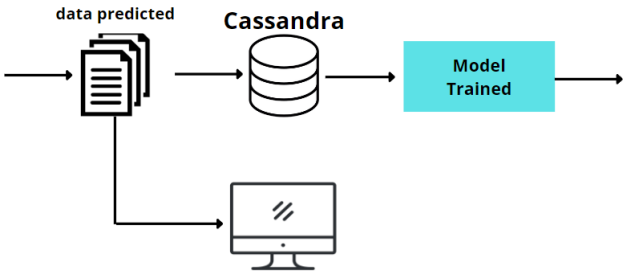


Hình 8: Giai Đoạn Stream Dữ Liệu

Dựa vào hình 8 có thể thấy được rằng việc dùng Spark Streaming để load dữ liệu streaming trực tiếp từ Kafka Topic tương ứng để tiến hành xử lý và dự đoán độ trễ của chuyến bay ngay lập tức. Ở đây chúng tôi sử dụng Kafka kết hợp với Spark Streaming để mô hình cho tốc độ nhận tin và xử lý liên tục.

Spark Streaming giúp xử lý dữ liệu thời gian thực từ nhiều nguồn khác nhau như Kafka, Flume và Amazon Kinesis. Sau khi xử lý, dữ liệu này có thể được đẩy ra hệ thống tệp hoặc cơ sở dữ liệu. Spark Streaming được sử dụng cho dữ liệu thời gian thực có thể là dữ liệu phi cấu trúc như hình ảnh hoặc văn bản.

5.3 Giai Đoạn Cập Nhật Mô Hình



Hình 9: Cập nhật mô hình và hiển thị kết quả

Dữ liệu được dự đoán xong sẽ được lưu lại vào Cassandra để tiến hành cập nhật lại mô hình sau này. Đồng thời dữ liệu dự đoán cũng sẽ được đưa ra màn hình ứng dụng để người dùng có thể nhìn trực quan kết quả. Hình 9 mô phỏng lại kiến trúc ở giai đoạn này.

Chúng tôi sử dụng thư viện Skinter của Python để thiết kế một giao diện thể hiện được kết quả dưới dạng danh sách các chuyến bay khi được dự đoán xong. Giao hiện ở Hình 10 lấy cảm hứng từ màn hình thống báo trạng thái chuyến bay ở sân bay và được chúng tôi mô phỏng lại.

App

Chuyen Bay/Departures Time: 07:13:19

ID	ORIGIN	DEST	CRS_DEP_TIME	prediction
4/13/2022-OH-CLT-CSG-5339	CLT	CSG	20.26	Not Delay
4/14/2022-OH-CLT-CSG-5339	CLT	CSG	20.26	Not Delay
4/15/2022-OH-CLT-CSG-5339	CLT	CSG	20.26	Not Delay
4/16/2022-OH-CLT-CSG-5339	CLT	CSG	20.26	Not Delay
4/17/2022-OH-CLT-CSG-5339	CLT	CSG	20.26	Not Delay
4/18/2022-OH-CLT-CSG-5339	CLT	CSG	20.26	Not Delay
4/19/2022-OH-CLT-CSG-5339	CLT	CSG	20.26	Not Delay
4/20/2022-OH-CLT-CSG-5339	CLT	CSG	20.26	Not Delay
4/21/2022-OH-CLT-CSG-5339	CLT	CSG	20.26	Not Delay

Hình 10: Màn hình kết quả

6 Kết Luận

Trong báo cáo này, chúng tôi đã xây dựng và trình bày bộ dữ liệu Flight Delays Dataset, một bộ dữ liệu mới cho bài toán dự đoán độ trễ chuyến bay bao gồm hai tập dữ liệu là tập training/testing chứa 2,157,737 dòng dữ liệu và 11 thuộc tính là các thông tin chuyến bay và tập dữ liệu streaming giả lập real–time chứa 542,117 dòng dữ liệu và 11 thuộc tính. Với mỗi một thông tin máy bay sẽ được gán một trong ba nhãn "0" (không trễ), "1" (trễ không quá 30 phút) và "2" (trễ hơn 30 phút hoặc hủy chuyến). Hiện tại, chúng tôi đã cài đặt thành công 4 mô hình sử dụng các thuật toán Machine Learning: Logistic Regression, Decision Tree Classifier, Random Forest Classifier, Naive Bayes được hỗ trợ bởi Machine Learning Library (MLlib) là một thư viện máy học của Spark. Kết quả cao nhất mà chúng tôi đạt được là 45.27% F1-macro. Kết quả của chúng tôi đạt được không cao, nó đặt ra một thách thức cho các nhóm nghiên cứu sau về việc cải thiện kết quả cho bài toán. Ngoài ra, chúng tôi đã xây dựng thành công một hệ thống giả lập dự đoán độ trễ của chuyến bay theo thời gian thực, bằng cách sử dụng các framework làm việc với BigData và kết hợp chúng lại thành một hệ thống hoàn chỉnh.

Hướng phát triển trong tương lai: (1) Xử lý mất cân bằng dữ liệu bằng các kỹ thuật oversampling và undersampling [6]. (2) Áp dụng các mô hình Deep Learning như Artificial Neural Network (ANN) [7] trong BigData bằng cách sử dụng framework BigDL [8].

Tài liệu

1. Singh, P., 2019. Logistic Regression. In Machine Learning with PySpark (pp. 65-98). Apress, Berkeley, CA.
2. Dalianis, H., 2018. Evaluation metrics and evaluation. In Clinical text mining (pp. 45-53). Springer, Cham.
3. Zhang, Z., 2018. Artificial neural network. In Multivariate time series analysis in climate and environmental research (pp. 1-35). Springer, Cham.
4. Avinash Lakshman, Prashant Malik, 2010. Cassandra - A Decentralized Structured Storage System
5. Jay Kreps, Neha Narkhede, Jun Rao, 2011. Kafka: a Distributed Messaging System for Log Processing
6. Shelke, M.S., Deshmukh, P.R. and Shandilya, V.K., 2017. A review on imbalanced data handling using undersampling and oversampling technique. Int. J. Recent Trends Eng. Res, 3(4), pp.444-449.
7. Wang, S.C., 2003. Artificial neural network. In Interdisciplinary computing in java programming (pp. 81-100). Springer, Boston, MA.
8. Dai, J.J., Ding, D., Shi, D., Huang, S., Wang, J., Qiu, X., Huang, K., Song, G., Wang, Y., Gong, Q. and Song, J., 2022. BigDL 2.0: Seamless Scaling of AI Pipelines from Laptops to Distributed Cluster. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 21439-21446).