University of Science
Vietnam National University,
Ho Chi Minh City

Project 2

TEXT CLASSIFICATION

Lớp: 21CNTT
GVHD: Dr. Nguyen Hong Buu Long
GVTH: Dr. Le Thanh Tung, Dr. Luong An Vinh

Mục lục

1. Thông tin chung

1.1 Thành viên nhóm

MSSV	Họ và tên	Email
21127175	Lê Anh Thư	lathu21@clc.fitus.edu.vn
21127191	Nguyễn Nhật Truyền	nntruyen21@clc.fitus.edu.vn
21127592	Nguyễn Minh Đạt	nmdat21@clc.fitus.edu.vn

1.2 Phân công công việc

Nhiệm vụ	Thành viên
Cài đặt thuật toán Decision Tree, Random Forest Trực quan hóa	Lê Anh Thư
Cài đặt thuật toán SVM Viết báo cáo	Nguyễn Minh Đạt
Cài đặt thuật toán CRF Trả lời câu hỏi lý thuyết	Nguyễn Nhật Truyền

1.3 Đánh giá kết quả

Công việc	Hoàn thành
Cài đặt, phân tích thuật toán Decision Tree, Random Forest, SVM, CRF	100%
Khảo sát các tham số cho mỗi thuật toán	100%
Chạy thực nghiệm và đánh giá kết quả	100%
Trả lời câu hỏi phần lý thuyết	100%
Tổng hợp và viết báo cáo	100%

2. Trả lời câu hỏi lý thuyết

2.1 Câu 1

Câu hỏi: What are the differences between supervised, unsupervised and reinforcement learning, and how do they differ in their learning approaches?

Trả lời:

- **Học có giám sát** (Supervised Learning):
  - Mục tiêu:
    Trong học có giám sát, mô hình được đào tạo trên một bộ dữ liệu có nhãn, trong đó mỗi đầu vào được kết hợp với một đầu ra tương ứng. Mục tiêu là để mô hình học cách ánh xạ giữa đầu vào và đầu ra để có thể đưa ra dự đoán chính xác trên dữ liệu mới, chưa thấy trước.
  - Phương pháp học:
    Trong quá trình đào tạo, mô hình được trình bày với các cặp đầu vào-đầu ra, và nó điều chỉnh các tham số của mình để giảm thiểu sự khác biệt giữa các dự đoán của nó và đầu ra thực tế.

- **Học không giám sát** (Unsupervised Learning):
  - Mục tiêu:
    Trong học không giám sát, mô hình được cung cấp dữ liệu không có nhãn và nhiệm vụ là tìm ra các mô hình hoặc cấu trúc trong dữ liệu mà không có sự hướng dẫn cụ thể. Mục tiêu thường là khám phá mối quan hệ ẩn trong dữ liệu.
  - Phương pháp học:
    Mô hình khám phá dữ liệu mà không có mục tiêu cụ thể. Các kỹ thuật phổ biến bao gồm phân cụm (nhóm các điểm dữ liệu tương tự lại với nhau) và giảm chiều (giảm số lượng đặc trưng trong khi giữ nguyên thông tin quan trọng). Thuật toán học xác định mô hình hoặc cấu trúc dựa trên cấu trúc tiềm ẩn của dữ liệu.

- **Học tăng cường** (Reinforcement Learning):
  - Mục tiêu:
    Liên quan đến việc đào tạo các agent để đưa ra các quyết định theo chuỗi trong môi trường nhằm tối đa hóa một tín hiệu thưởng tích tụ. Agent học thông qua những thử nghiệm và sai lầm, nhận phản hồi dưới dạng thưởng hoặc trừng phạt dựa trên hành động của mình.
  - Phương pháp học:
    Quá trình học bao gồm một agent tương tác với môi trường. Agent thực hiện các hành động, và môi trường cung cấp phản hồi dưới dạng thưởng hoặc trừng phạt. Mục tiêu của agent là học một cách thực hiện hành động để tối đa hóa thưởng tích tụ dự kiến theo thời gian. Các kỹ thuật như cân bằng khám phá-khai thác và gán điểm thời gian đóng vai trò quan trọng trong học tăng cường.

2.2 Câu 2

Câu hỏi: When evaluating the performance of a text classification model, what are the common evaluation metrics used, and how can we assess its effectiveness?

Trả lời:

- **Accuracy**:
  - **Công thức**:
    (Number of correct predictions) / (Total number of predictions)
  - **Mô tả**:
    Độ chính xác đo lường sự chính xác tổng thể của mô hình bằng cách so sánh số dự đoán đúng với tổng số dự đoán. Mặc dù cung cấp cái nhìn tổng quan về hiệu suất, độ chính xác có thể không đủ đối với các bộ dữ liệu mất cân đối.
- **Precision**:
  - **Công thức**:
    (True Positives) / (True Positives + False Positives)

- **Mô tả**:
  Chính xác dương đo lường độ chính xác của các dự đoán tích cực. Đây là tỷ lệ của các quan sát tích cực được dự đoán đúng trên tổng số dự đoán tích cực. Chính xác dương hữu ích khi chi phí của các dự đoán sai là cao.

- **Recall**:
  - **Công thức**:
    (True Positives) / (True Positives + False Negatives)
  - **Mô tả**:
    Độ nhạy đo lường khả năng của mô hình bắt kịp tất cả các trường hợp quan trọng của lớp tích cực. Đây là tỷ lệ của các quan sát tích cực được dự đoán đúng trên tổng số tích cực thực tế. Độ nhạy quan trọng khi chi phí của các dự đoán sai âm là cao.

- **F1 Score**:
  - **Công thức**:
    *2 * (Precision * Recall) / (Precision + Recall)*
  - **Mô tả**:

    Điểm F1 là trung bình điều hòa giữa chính xác và độ nhạy. Nó cung cấp một đo lường cân đối có xem xét cả false positives và false negatives. Điểm F1 đặc biệt hữu ích khi phân phối lớp mất cân đối.

2.3 Câu 3

Câu hỏi: In the context of text classification, what are some techniques that can be employed to handle imbalanced datasets and address the challenges associated with class imbalance?

Trả lời:

- **Kỹ thuật Lấy Mẫu Lại (Resampling):**
  - Lấy Mẫu Dưới (Under-sampling):
    Giảm số lượng mẫu trong lớp đa số để phù hợp với lớp thiểu số. Điều này có thể được thực hiện ngẫu nhiên hoặc bằng cách sử dụng các phương pháp tinh tế hơn.

- Lấy Mẫu Quá (Over-sampling):
  Tăng số lượng mẫu trong lớp thiểu số bằng cách sao chép các mẫu hiện có hoặc tạo mẫu tổng hợp bằng các kỹ thuật như SMOTE (Synthetic Minority Over-sampling Technique).
- Một sự kết hợp giữa lấy mẫu quá của lớp thiểu số và lấy mẫu dưới của lớp đa số đôi khi có thể hiệu quả hơn so với việc sử dụng mỗi kỹ thuật một cách riêng lẻ

- **Tăng Cường Dữ Liệu (Data Augmentation):**
  - Tăng Cường Văn Bản (Text Augmentation):
    Tạo các mẫu đào tạo mới bằng cách áp dụng biến đổi cho dữ liệu văn bản hiện có. Điều này có thể bao gồm các kỹ thuật như thay thế từ đồng nghĩa, dịch ngược, hoặc embedding từ.
- **Trọng Số Lớp (Class Weighting):**
  - Gán trọng số cao hơn cho các mẫu trong lớp thiểu số trong quá trình đào tạo mô hình. Điều này trừng phạt việc phân loại sai các mẫu của lớp thiểu số nặng hơn, khuyến khích mô hình chú ý hơn đến lớp thiểu số.
- **Phương Pháp Hợp Nhất (Ensemble Methods):**
  - Sử dụng các phương pháp hợp nhất như bagging và boosting để kết hợp dự đoán của nhiều mô hình. Điều này có thể giúp giảm thiểu ảnh hưởng của các lớp mất cân đối bằng cách sử dụng sức mạnh của các mô hình khác nhau.
- **Học Nhạy Chi Phí (Cost-sensitive Learning):**
  - Sửa đổi thuật toán học để trở thành nhạy chi phí, nơi phân loại sai các mẫu của lớp thiểu số được phạt nặng hơn so với việc phân loại sai các mẫu của lớp đa số.
- **Điều Chỉnh Ngưỡng (Threshold Adjustment):**
  - Điều chỉnh ngưỡng phân loại của mô hình. Theo mặc định, các phân loại thường sử dụng ngưỡng 0.5 cho phân loại nhị phân. Thay đổi ngưỡng này có thể ảnh hưởng đến sự cân đối giữa độ chính xác và độ nhạy.
- **Các Chỉ Số Đánh Giá Khác Nhau:**
  - Thay vì chỉ dựa vào accuracy, sử dụng các chỉ số đánh giá có ích hơn cho các bộ dữ liệu mất cân đối.

Precision, recall, F1 score, ROC-AUC, và PR-AUC thường phù hợp hơn để đánh giá hiệu suất trong bối cảnh lớp mất cân đối.

- **Sử Dụng Mô Hình Phát Hiện Bất Thường (Anomaly Detection Models):**
  - Xem xét lớp thiểu số như là một ngoại lệ và sử dụng các kỹ thuật phát hiện bất thường. Điều này đặc biệt hữu ích khi lớp thiểu số đại diện cho các sự kiện hiếm.
- **Học Chuyển Giao (Transfer Learning):**
  - Tận dụng các mô hình được đào tạo trước đó trên các bộ dữ liệu lớn, cân đối và điều chỉnh chúng trên bộ dữ liệu mất cân đối. Học chuyển giao có thể giúp mô hình tổng quát tốt hơn, ngay cả với dữ liệu hạn chế trong lớp thiểu số.

2.4 Câu 4

Câu hỏi: Support Vector Machines (SVMs) are primarily designed for binary classification. Can you explain the approach used to extend SVMs for handling multi-class classification problems and how it enables them to classify data into multiple classes?

Trả lời:

Máy học Support Vector Machines (SVMs) được thiết kế chủ yếu để phân loại nhị phân, có nghĩa là chúng phân loại các thể hiện vào một trong hai lớp. Tuy nhiên, có các phương pháp mở rộng SVMs để xử lý các vấn đề phân loại đa lớp. Hai chiến lược phổ biến là Phương pháp One-vs-One (OvO) và One-vs-All (OvA).

- **Phương pháp One-vs-One (OvO):**
  - Chiến lược:
    Trong chiến lược OvO, một SVM nhị phân riêng biệt được đào tạo cho mỗi cặp lớp. Nếu có N lớp, điều này dẫn đến N * (N-1) / 2 bộ phân loại. Mỗi SVM nhị phân được đào tạo trên tập dữ liệu con chứa các thể hiện từ chỉ hai lớp.

- Dự đoán:
  Để thực hiện dự đoán cho một thể hiện mới, mỗi bộ phân loại nhị phân bình chọn cho lớp được gán của nó. Lớp có số phiếu bình chọn nhiều nhất sẽ được gán cho thể hiện đó. Phương pháp này giảm bài toán phân loại đa lớp thành một chuỗi các bài toán phân loại nhị phân.

- **Phương pháp One-vs-All (OvA):**
  - Chiến lược:
    Trong chiến lược OvA, một SVM nhị phân được đào tạo cho mỗi lớp so với phần còn lại các lớp. Nếu có N lớp, sẽ có NSVM nhị phân được đào tạo. Mỗi SVM được đào tạo để phân biệt giữa các thể hiện thuộc một lớp và các thể hiện của tất cả các lớp khác.
  - Dự đoán:
    Trong quá trình dự đoán, mỗi bộ phân loại nhị phân tạo ra một điểm đại diện cho lớp được gán của nó. Lớp có điểm đại diện cao nhất sẽ được dự đoán là đầu ra cho thể hiện cụ thể. Phương pháp này cũng giảm bài toán phân loại đa lớp thành một chuỗi các bài toán phân loại nhị phân.
- **Ưu điểm:**
  - Ưu điểm của OvO:

    Ưu điểm của OvO là mỗi bộ phân loại chỉ cần được đào tạo trên một tập con của dữ liệu, làm cho nó hiệu quả về mặt tính toán. Có thể phù hợp hơn khi xử lý một số lớp lớn.

  - Ưu điểm của OvA:

    OvA thường hiệu quả về mặt tính toán hơn so với OvO, đặc biệt là khi số lượng lớp lớn. Nó thường chỉ đòi hỏi ít bộ phân loại nhị phân hơn so với OvO.

2.5 Câu 5

Câu hỏi: How could you define the kernel within the SVM algorithm? Furthermore, could you introduce a few kernel functions offered in scikit-learn and elaborate on the variations among them?

Trả lời:

- Kernel SVM là việc đi tìm một hàm số biến đổi dữ liệu x từ không gian feature ban đầu thành dữ liệu trong một không gian mới bằng hàm số Φ(x). hàm Φ() đơn giản là tạo thêm một chiều dữ liệu mới (một feature mới) là một hàm số của các features đã biết. Hàm số này cần thỏa mãn mục đích của chúng ta: trong không gian mới, dữ liệu giữa hai classes là phân biệt tuyến tính hoặc gần như phần biệt tuyến tính. Khi đó, ta có thể dùng các bộ phân lớp tuyến tính thông thường như PLA, Logistic Regression, hay Hard/Soft Margin SVM.
- Các hàm Φ() thường tạo ra dữ liệu mới có số chiều cao hơn số chiều của dữ liệu ban đầu, thậm chí là vô hạn chiều. Nếu tính toán các hàm này trực tiếp, chắc chắn chúng ta sẽ gặp các vấn đề về bộ nhớ và hiệu năng tính toán. Có một cách tiếp cận là sử dụng các kernel functions mô tả quan hệ giữa hai điểm dữ liệu bất kỳ trong không gian mới, thay vì đi tính toán trực tiếp từng điểm dữ liệu trong không gian mới. Kỹ thuật này được xây dựng dựa trên quan sát về bài toán đối ngẫu của SVM.
- Việc chọn hàm kernel rất quan trọng vì nó có thể ảnh hưởng đến hiệu suất và khả năng khái quát hóa của mô hình SVM của bạn. Hàm hạt nhân phù hợp với cấu trúc cơ bản của dữ liệu có thể giúp SVM tìm ra giải pháp tốt hơn, trong khi hàm hạt nhân không phù hợp với dữ liệu có thể dẫn đến tình trạng khớp quá mức, thiếu khớp hoặc kém hiệu quả tính toán.
- Việc lựa chọn hàm nhân cho mô hình SVM không phải là một khoa học chính xác, nhưng có một số hướng dẫn chung có thể giúp bạn đưa ra quyết định có thông tin. Để bắt đầu, hãy thử nghiệm linear kernel trước. Đây là lựa chọn đơn giản và nhanh chóng, và có thể hoạt động tốt cho nhiều vấn

đề. Nếu dữ liệu của bạn có thể phân tách tuyến tính hoặc có số chiều thấp, linear kernel có thể là đủ. Nếu không, hãy thử nghiệm các hàm nhân khác như polynomial, RBF, hoặc sigmoid để đánh giá và so sánh độ chính xác, tốc độ, và độ phức tạp. Ngoài ra, bạn nên điều chỉnh các tham số của hàm nhân để tối ưu hóa SVM của bạn cho dữ liệu và vấn đề cụ thể của bạn. Sau khi thử nghiệm các hàm nhân khác nhau và điều chỉnh tham số của chúng, bạn có thể chọn ra hàm nhân tốt nhất dựa trên các tiêu chí như độ chính xác, precision, recall, F1-score, sai số trung bình bình phương, chi phí tính toán, khả năng giải thích, và sự ổn định.

- Trong scikit-learn, các hàm kernel đóng vai trò quan trọng trong các thuật toán Support Vector Machines (SVM). Dưới đây là một số hàm kernel và sự biến thể của chúng:
  - **Linear Kernel**:

    Mô tả: Đại diện cho mối quan hệ tuyến tính giữa các đặc trưng đầu vào.

    Biến thể: Không có tham số bổ sung; đây là kernel đơn giản nhất.

  - **Polynomial Kernel**:

    Mô tả: Đưa ra tính phi tuyến bằng cách nâng giá trị tích vô hướng của các đặc trưng đầu vào lên một công suất cụ thể.

    Biến thể: Tham số degree kiểm soát bậc đa thức.

  - **RBF (Radial Basis Function) Kernel**:

    Mô tả: Còn được biết đến là kernel Gaussian, nó xem xét sự tương đồng giữa các điểm dữ liệu trong không gian vô hạn chiều.

    Biến thể: Tham số gamma ảnh hưởng đến hình dạng ranh giới quyết định.

  - **Sigmoid Kernel:**

    Mô tả: Sử dụng hàm siêu phẳng dạng hyperbolic để bắt kịp các mối quan hệ phức tạp.

Biến thể: Các tham số bao gồm gamma và coef0, ảnh hưởng đến hình dạng và độ lệch của kernel.

3. Cài đặt và đánh giá

3.1 Thuật toán Decision Tree

3.1.1 Thông tin lý thuyết

Khái niệm:

Cây Quyết Định là một phương pháp học **có giám sát**, **phi tham số** được sử dụng cho việc **phân loại** và **hồi quy**. Mục tiêu là tạo ra một mô hình dự đoán giá trị của biến mục tiêu bằng cách học các **quy tắc** quyết định được suy ra từ các đặc trưng dữ liệu.

Lớp **DecisionTreeClassifier** thuộc thư viện **scikit-learn**:

- DecisionTreeClassifier là một lớp có khả năng thực hiện **phân loại đa lớp** (multi-class classification) trên một bộ dữ liệu.
- DecisionTreeClassifier nhận hai mảng làm đầu vào:
  - Mảng X, có kích thước (n_samples, n_features) chứa các mẫu huấn luyện
  - Mảng Y, có kích thước (n_samples,), chứa các số nguyên là nhãn cho các mẫu huấn luyện.

Danh sách tham số:

Tên tham số	
criterion	**Ý nghĩa**: Đánh giá tầm quan trọng của các đặc trưng **Giá trị**: ● “gini” (default) ● “log_loss” ● “entropy”
splitter	**Ý nghĩa**: Quyết định đặc trưng nào và ngưỡng (threshold) nào được sử dụng **Giá trị**: ● “best”: Đối với tất cả các đặc trưng, chọn điểm **tốt nhất** để chia nhánh, sau đó chọn đặc trưng

	tốt nhất làm quyết định cuối cùng. ● “random”: Đối với tất cả các đặc trưng, chọn **ngẫu nhiên** một điểm để chia nhánh, sau đó chọn đặc trưng tốt nhất làm quyết định cuối cùng.
max_depth	**Ý nghĩa**: Độ sâu tối đa của cây **Giá trị**: int (default=None) ● None: Các node được mở rộng cho đến khi tất cả các node lá đều thuần khiết.
min_samples_split	**Ý nghĩa**: Số lượng mẫu tối thiểu cần thiết để chia một node giữa **Giá trị**: ● int: min_samples_split=min_samples_split ● float: min_samples_split=ceil(min_samples_split * n_samples)
min_samples_leaf	**Ý nghĩa**: Số lượng mẫu tối thiểu cần phải có ở một node lá **Giá trị**: ● int: min_samples_leaf=min_samples_leaf ● float: min_samples_leaf=ceil(min_samples_leaf * n_samples)
min_weight_fraction_leaf	**Ý nghĩa**: Tỷ lệ tối thiểu của tổng trọng số (của tất cả các mẫu đầu vào) yêu cầu để trở thành một nút lá. Các mẫu có trọng số bằng nhau khi không cung cấp sample_weight. **Giá trị**: float (default=0)
max_features	**Ý nghĩa**: Số lượng các đặc trưng được xem xét khi tìm kiếm phân chia tốt nhất trong một cây quyết định.

	Giá trị: ● int: max_features=max_features ● float: max_features=max(1, int(max_features * n_features_in_)) ● sqrt: max_features=sqrt(n_features) ● log2: max_features=log2(n_features) ● None: max_features=n_features
random_state	**Ý nghĩa**: Điều khiển tính ngẫu nhiên của bộ ước lượng. Các đặc trưng luôn bị hoán đổi một cách ngẫu nhiên ở mỗi lần chia, ngay cả khi splitter được thiết lập thành "best". Khi max_features < n_features, thuật toán sẽ chọn max_features một cách ngẫu nhiên ở mỗi lần chia trước khi tìm ra phân chia tốt nhất trong số chúng. **Giá trị**: int (default=None)
max_leaf_nodes	**Ý nghĩa**: Xây dựng một cây với max_leaf_nodes theo phong cách best-first. Các nút tốt nhất được xác định dựa trên sự giảm tương đối trong độ không thuần khiết. Nếu None, thì không giới hạn số lượng nút lá. **Giá trị**: int (default=None)
min_impurity_decrease	**Ý nghĩa**: Một node sẽ được chia nếu việc chia này dẫn đến giảm độ không thuần khiết (impurity) lớn hơn hoặc bằng giá trị min_impurity_decrease. **Giá trị**: float (default=0)
class_weight	**Ý nghĩa**: Trọng số được liên kết với các lớp dưới dạng {class_label: weight}. Nếu None, tất cả các lớp được cho là có trọng số là một. **Giá trị**: dict/list of dict/”balanced” (default=None) ● “balanced”: xem mỗi lớp là quan trọng như

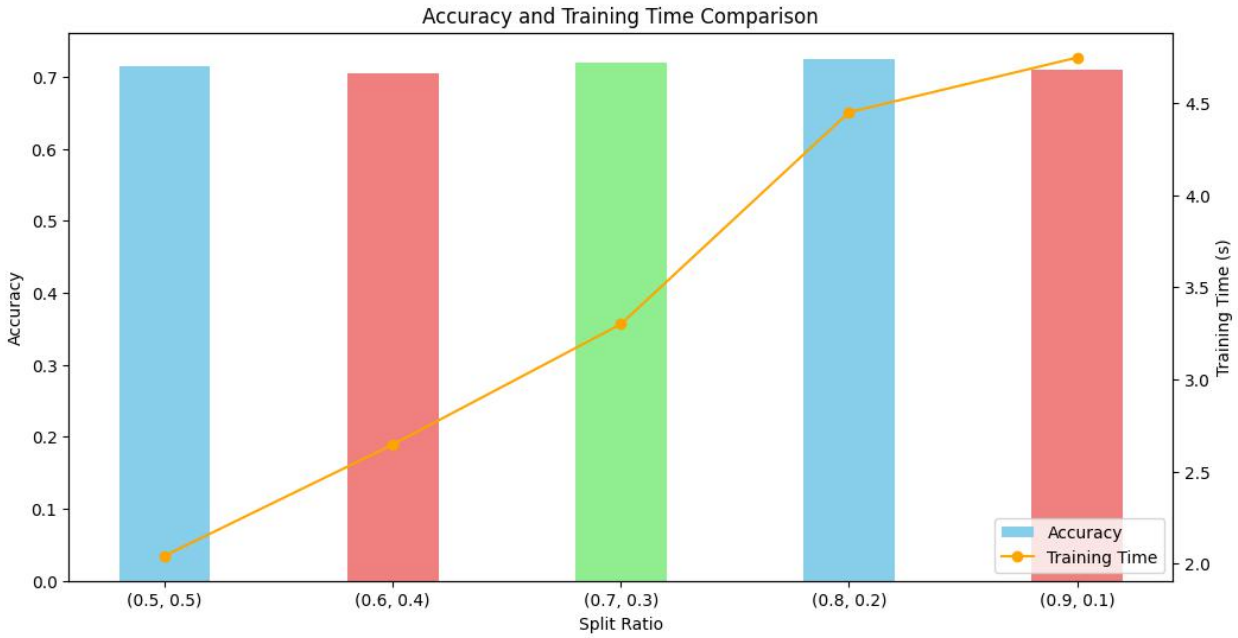
	nhau
ccp_alpha	**Ý nghĩa**: Tham số phức tạp được sử dụng để cắt tỉa theo Minimal Cost-Complexity Pruning. Cây con có độ phức tạp chi phí lớn nhất nhưng nhỏ hơn ccp_alpha sẽ được chọn. Mặc định, không có cắt tỉa được thực hiện. **Giá trị**: float (default=0)

3.1.2 Kết quả thực nghiệm

Khảo sát: **Tỉ lệ phân chia train/test**

Với max_features=5000 và criterion=”gini":

split_ratio	accuracy	training_time
5/5	0.714	2.042
6/4	0.704	2.646
7/3	0.719	3.299
8/2	0.725	4.448
9/1	0.709	4.745



Nhận xét:

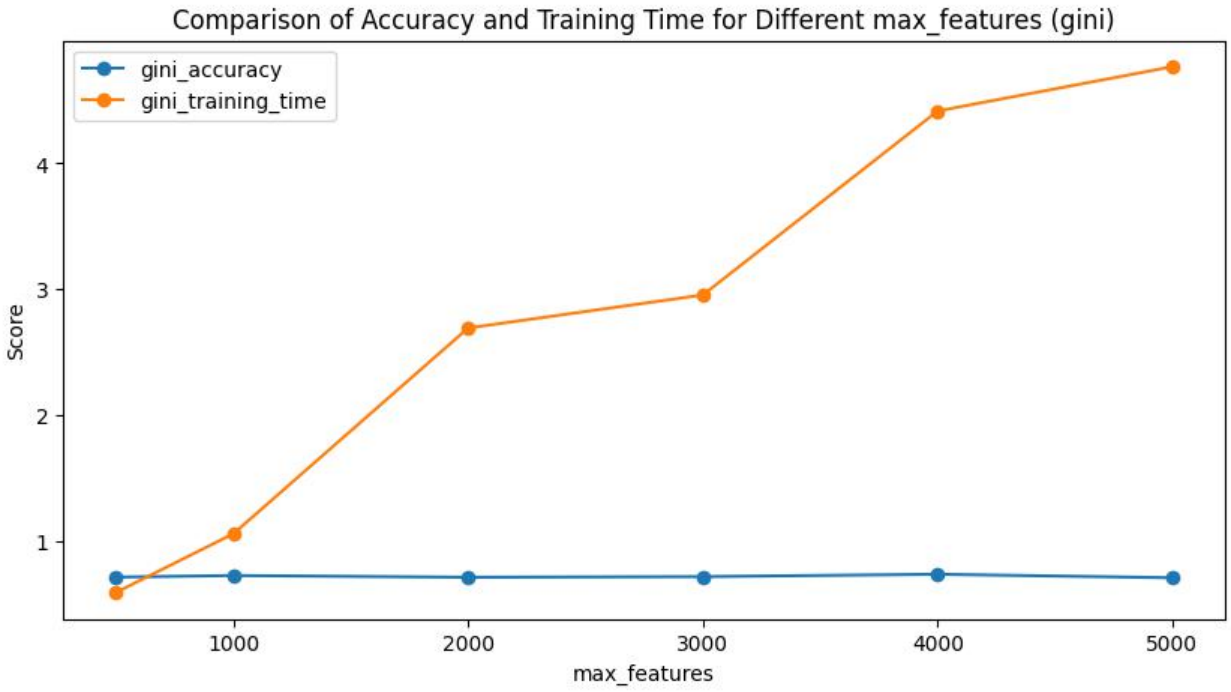
Đối với **Accuracy**:

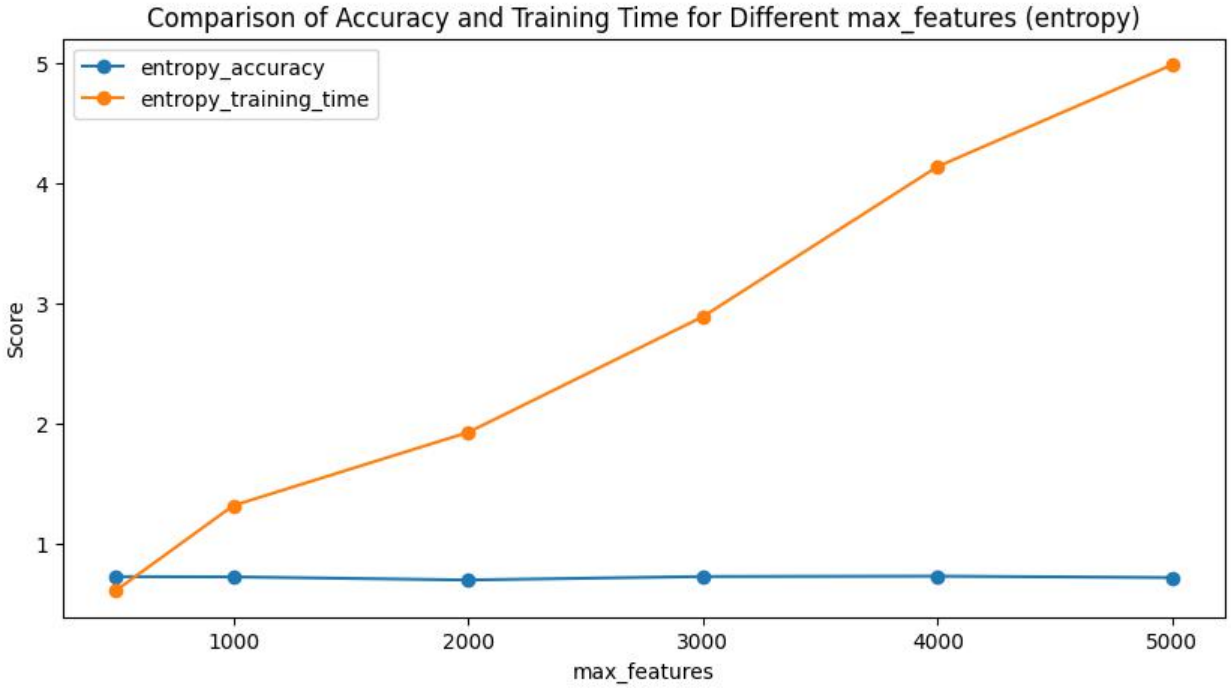
- Accuracy gần như không đổi (hay có thể nói là không phụ thuộc) vào tỉ lệ phân chia train/test

Đối với **Training time**:

- Thời gian huấn luyện tăng tỉ lệ thuận với phần trăm dữ liệu huấn luyện. Hay nói cách khác, training data càng nhiều, training time càng lâu.

Khảo sát: **Tham số max_features và criterion**





max_features	criterion	accuracy	training_time
500	gini	0.714	0.592
	entropy	0.723	0.607
1000	gini	0.727	1.058
	entropy	0.721	1.315
2000	gini	0.713	2.691
	entropy	0.695	1.924
3000	gini	0.718	2.953
	entropy	0.724	2.887
4000	gini	0.737	4.414
	entropy	0.727	4.138
5000	gini	0.709	4.765
	entropy	0.715	4.985

Nhận xét:

Đối với tham số **max_features**:

- Thời gian huấn luyện (training_time) tăng tỉ lệ thuận với max_features
- Độ chính xác gần như không đổi (hay có thể nói là không phụ thuộc) giá trị của max_features

Đối với tham số **criterion**:

- 3 trường hợp gini cần **nhiều** thời gian huấn luyện hơn

entropy: khi max_features nhận các giá trị 2000, 3000, 4000

- 3 trường hợp gini cần **ít** thời gian huấn luyện hơn entropy: khi max_features nhận các giá trị 500, 1000, 5000
  → Nhìn chung, với cùng một giá trị max_features, cả gini và entropy có **thời gian huấn luyện** gần như bằng nhau.
- 3 trường hợp gini đạt độ chính xác **cao** hơn entropy: khi max_features nhận các giá trị 1000, 2000, 4000
- 3 trường hợp gini đạt độ chính xác **thấp** hơn entropy: khi max_features nhận các giá trị 500, 3000, 5000
  → Nhìn chung, với cùng một giá trị max_features, cả gini và entropy có **độ chính xác** gần như bằng nhau.

3.2 Thuật toán Random Forest

3.2.1 Thông tin lý thuyết

Khái niệm: Nói một cách đơn giản, "Random Forest" là một bộ **phân loại** chứa nhiều cây quyết định trên các tập con khác nhau của một bộ dữ liệu đã cho và lấy trung bình để tăng cường độ chính xác dự đoán của bộ dữ liệu đó. Thay vì dựa vào một cây quyết định duy nhất, "random forest" tổng hợp kết quả từ mỗi cây và dựa vào phiếu đa số của các dự đoán để đưa ra kết quả cuối cùng.

Lớp **RandomForestClassifier** thuộc thư viện **scikit-learn**: Bên cạnh các tham số tương tự lớp DecisionTreeClassifier**,** dưới đây là danh sách tham số của riêng lớp RandomForestClassifier:

Tên tham số	
n_estimators	**Ý nghĩa**: Số lượng cây trong rừng. **Giá trị**: int (default=100)
bootstrap	**Ý nghĩa:** Xác định xem liệu mẫu bootstrap có được sử dụng khi xây dựng cây hay không. Nếu False, toàn bộ tập dữ liệu được sử dụng để xây dựng mỗi cây.

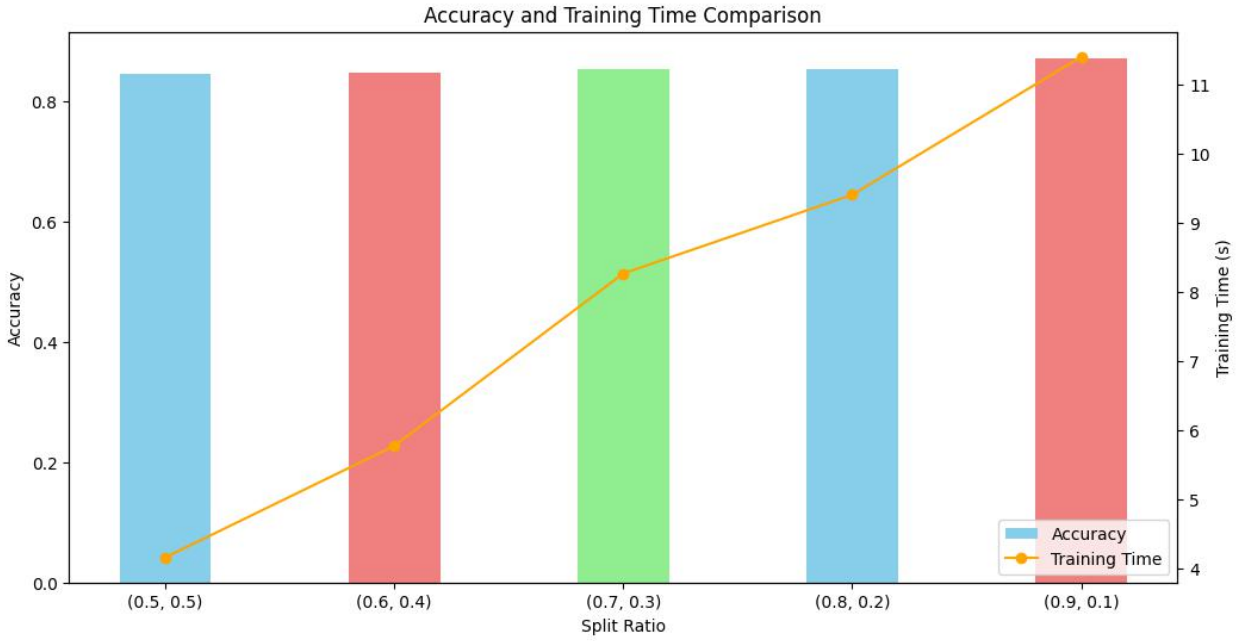
	Giá trị: bool (default=True)
oob_score	**Ý nghĩa:** Có sử dụng mẫu **out-of-bag** để ước tính điểm số tổng quát (generalization score) hay không. (Đối với mỗi cây, chỉ một phần dữ liệu được chọn để xây dựng cây. Các mẫu dữ liệu còn lại được gọi là mẫu out-of-bag. Những mẫu này có thể được sử dụng để tính toán độ chính xác thử nghiệm) **Giá trị**: bool/callable (default=False)
n_jobs	**Ý nghĩa:** Số lượng công việc chạy song song. Các hoạt động như fit, predict, decision_path và apply được thực hiện song song trên các cây. **Giá trị**: int (default=None)
verbose	**Ý nghĩa:** Điều khiển mức độ chi tiết trong các thông báo đầu ra trong quá trình khớp và dự đoán. **Giá trị**: int (default=0)
warm_start	**Ý nghĩa:** Khi được đặt thành True, tái sử dụng kết quả của lần gọi trước đến phương thức fit và thêm các bộ ước lượng khác vào bộ tổ hợp, ngược lại, chỉ khớp một rừng hoàn toàn mới. Xem Thuật ngữ và Khớp các bộ ước lượng yếu bổ sung để biết chi tiết. **Giá trị**: bool (default=False)

3.2.2 Kết quả thực nghiệm

Khảo sát: **Tỉ lệ phân chia train/test**

Với n_estimators=100 và max_features=”sqrt":

split_ratio	accuracy	training_time
5/5	0.844	4.163
6/4	0.846	5.775
7/3	0.852	8.264
8/2	0.852	9.401
9/1	0.871	11.393



Nhận xét:

Đối với **Accuracy**:

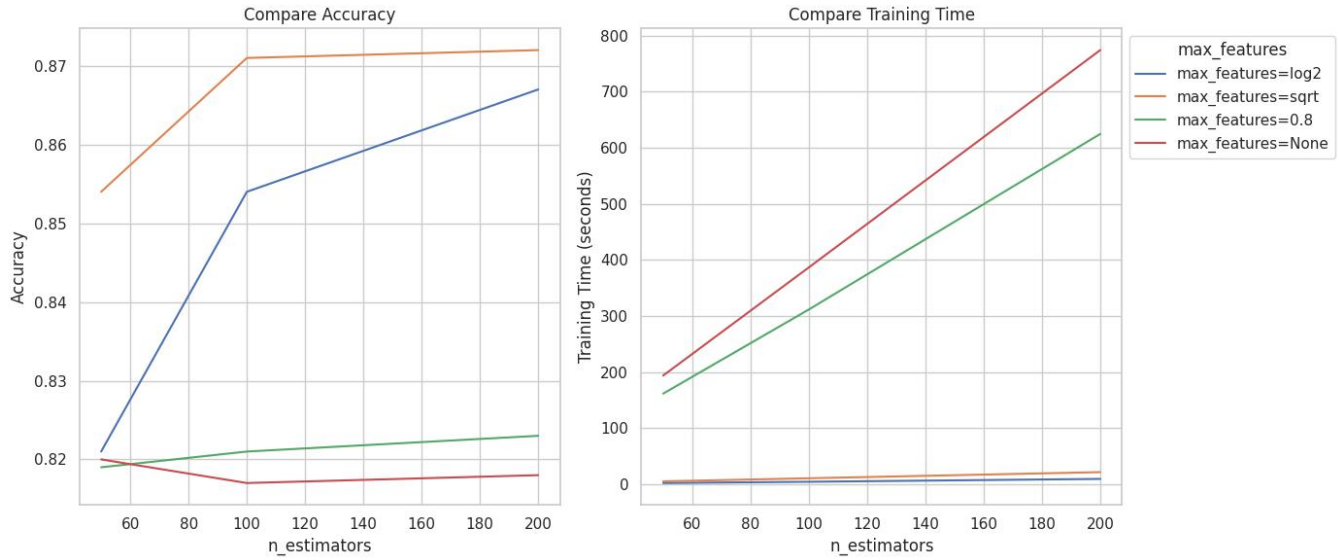
- Độ chính xác tăng tỉ lệ thuận với phần trăm dữ liệu huấn luyện, mặc dù khoảng tăng thêm là không lớn (accuracy tăng xấp xỉ 3% từ 84,4% lên 87,1% khi training data tăng từ 50% lên 90%)

Đối với **Training time**:

- Thời gian huấn luyện tăng tỉ lệ thuận với phần trăm dữ liệu huấn luyện. Hay nói cách khác, training data càng nhiều, training time càng lâu.

Khảo sát: **Tham số n_estimators và max_features**

n_estimators	max_features	accuracy	training_time
50	log2	0.821	2.39
	sqrt	0.854	5.44
	0.8	0.819	161.84
	None	0.820	194.08
100	log2	0.854	4.64
	sqrt	0.871	10.86
	0.8	0.821	311.87
	None	0.817	386.93
200	log2	0.867	9.73
	sqrt	0.872	21.82
	0.8	0.823	624.85
	None	0.818	774.32



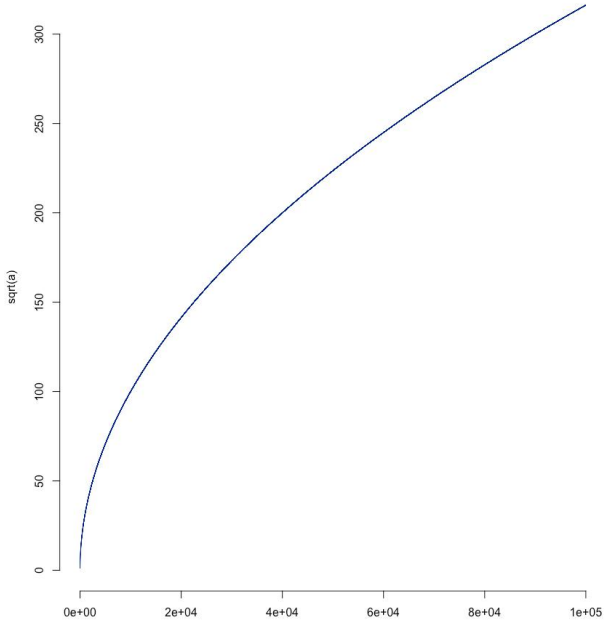
Nhận xét:

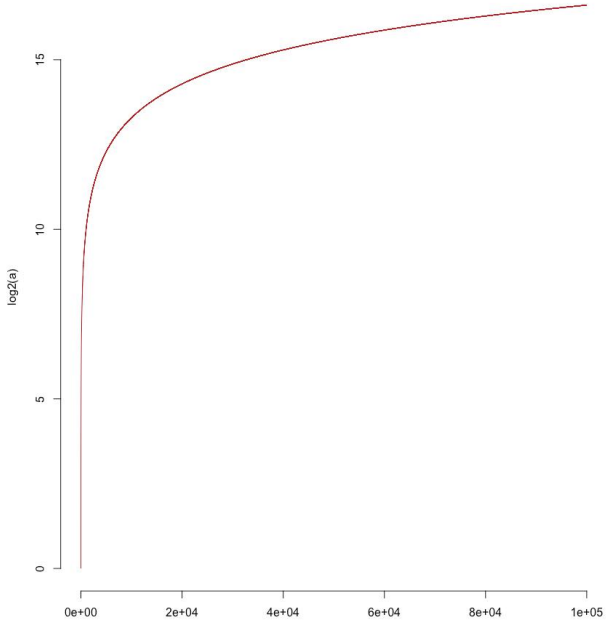
Đối với **max_features**:

- “sqrt” cho kết quả tốt nhất, kế đến là “log2”. Với trường hợp None (max_features=n_features), thuật toán cho kết quả thấp nhất.
  - Giải thích: hàm “sqrt” tăng đều, do đó, số lượng đặc trưng được chọn có tỉ lệ tương đối phù hợp so với tổng số lượng đặc trưng.
  - Trong khi đó, việc gán cứng max_features=n_features mang lại hiệu quả kém do hoàn toàn phụ thuộc vào số lượng đặc trưng của bộ dữ liệu (chẳng hạn như khi n_features quá lớn, thuật toán trở lên tốn chi phí và không hiệu quả)

Đối với **n_estimators**:

- Nhìn chung, giá trị n_estimators tăng tỉ lệ thuận với độ chính xác và thời gian huấn luyện.





3.3 Thuật toán Support Vector Machine

3.3.1 Thông tin lý thuyết

- "Support Vector Machines" (SVM) là một tập hợp các phương pháp học **có giám sát** được sử dụng cho việc phân loại, hồi quy và phát hiện các điểm ngoại lệ (outliers).
- Một bộ phân loại SVM tuyến tính đơn giản hoạt động bằng cách tạo ra một đường thẳng giữa hai lớp, tất cả các điểm dữ liệu ở một bên của đường thẳng sẽ thuộc về một nhóm và các điểm dữ liệu ở phía còn lại của đường thẳng sẽ thuộc về một nhóm khác. Điều này có nghĩa là có thể có **vô số** đường thẳng để lựa chọn.
- Điều làm cho thuật toán SVM tuyến tính tốt hơn một số thuật toán khác, như k-nearest neighbors, là nó chọn đường thẳng **tốt nhất** để phân loại các điểm dữ liệu. Nó chọn đường thẳng tách biệt dữ liệu và cách xa nhất có thể từ các điểm dữ liệu gần nhất.

Danh sách tham số:

Tên tham số	
C	**Ý nghĩa**: mức độ quan trọng tương đối của loss term so với regularization term. Trong đó: ● Loss term: đảm bảo rằng các trọng số phân loại đúng các điểm dữ liệu huấn luyện. ● Regularization term: có lợi cho các trọng số "đơn giản", chống lại hiện tượng overfitting **Giá trị**: float
kernel	**Ý nghĩa**: Loại nhân (kernel) được sử dụng trong thuật toán **Giá trị**: ● “linear”: hạt nhân tuyến tính ● “poly”: hạt nhân đa thức ● “rbf” hoặc “gaussian”: hạt nhân Radial basis function (RBF) ● “sigmoid”: hạt nhân sigmoid
degree	**Ý nghĩa**: Số mũ của hàm hạt nhân đa thức ('poly'). Bị bỏ qua bởi tất cả các loại hạt nhân khác. **Giá trị**: int (default=3)
gamma	**Ý nghĩa**: Hệ số hạt nhân cho 'rbf', 'poly' và 'sigmoid'. • linear: $\langle x, x' \rangle$. • polynomial: $(\gamma \langle x, x' \rangle + r)^d$, where d is specified by parameter `degree`, r by `coef0`. • rbf: $\exp(-\gamma \|x - x'\|^2)$, where γ is specified by parameter `gamma`, must be greater than 0. • sigmoid $\tanh(\gamma \langle x, x' \rangle + r)$, where r is specified by `coef0`. **Giá trị**: ● float ● “scale": gamma = 1 / (n_features * X.var()) ● “auto”: gamma = 1 / n_features

coef0	**Ý nghĩa**: Hệ số tự do trong hàm hạt nhân. Chỉ có ý nghĩa trong 'poly' và 'sigmoid'. **Giá trị**: float (default=0)
shrinking	**Ý nghĩa**: Cờ báo có nên sử dụng shrinking heuristic, một phương pháp nhằm tăng tốc vấn đề tối ưu hóa không. **Giá trị**: bool (default=True)
probability	**Ý nghĩa**: Cờ báo có cho phép ước lượng xác suất hay không. **Giá trị**: bool (default=False)
tol	**Ý nghĩa**: yêu cầu dừng việc tìm kiếm giá trị tối thiểu (hoặc tối đa) khi đạt được một độ chính xác nhất định **Giá trị**: float (default=1e-3)
cache_size	**Ý nghĩa**: kích thước bộ nhớ cache (đơn vị: MB) **Giá trị**: float (default=200)
class_weight	**Ý nghĩa**: Thiết lập tham số C của lớp i thành class_weight[i]*C cho SVC **Giá trị**: ● dict ● “balanced": sử dụng các giá trị của y để tự động điều chỉnh trọng số theo tỉ lệ nghịch đảo với tần suất lớp trong dữ liệu đầu vào theo công thức: n_samples / (n_classes * np.bincount(y)) ● default=None
verbose	**Ý nghĩa**: Cờ báo có bật đầu ra chi tiết hay không.

	Giá trị: bool (default=False)
max_iter	**Ý nghĩa**: Chặn cứng về số lần lặp trong giải thuật **Giá trị**: int ● default=-1 (không có giới hạn.)
decision_function_shape	**Ý nghĩa**: Cờ báo cho biết liệu có trả về hàm quyết định một-vs-nhiều hay là hàm quyết định một-vs-một **Giá trị**: ● “ovo": hàm quyết định một-vs-một ● “ovr": hàm quyết định một-vs-nhiều (default)
break_ties	**Ý nghĩa**: ● Nếu break_ties là True, decision_function_shape='ovr', và số lớp > 2, predict sẽ phân giải xung đột dựa trên các giá trị độ tin cậy của decision_function ● Nếu không, lớp đầu tiên trong số các lớp có điểm số bằng nhau sẽ được trả về **Giá trị**: bool (default=False)
random_state	**Ý nghĩa**: kiểm soát quá trình tạo số ngẫu nhiên giả khi trộn dữ liệu cho các ước lượng xác suất **Giá trị**: int (default=None)

3.3.2 Kết quả thực nghiệm

Khảo sát: **Tỉ lệ phân chia train/test**

Với C=1 và kernel=’sigmoid’

split_ratio	accuracy	training_time
5/5	0.866	11.692

6/4	0.869	16.103
7/3	0.867	20.960
8/2	0.874	26.458
9/1	0.887	32.194

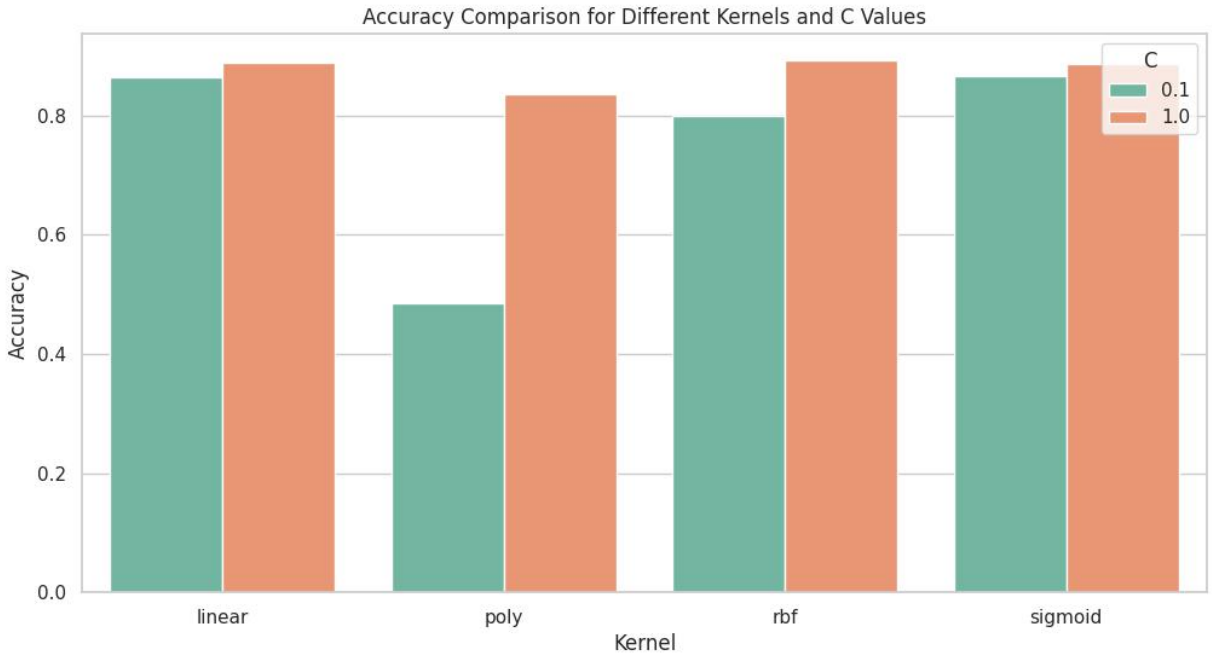
Nhận xét:

- Phần trăm dữ liệu huấn luyện tăng tỉ lệ thuận với thời gian huấn luyện. Hay nói cách khác, dữ liệu huấn luyện càng nhiều, thời gian huấn luyện càng lâu.
- Nhìn chung, độ chính xác tăng nhẹ khi phần trăm dữ liệu huấn luyện tăng lên, tuy nhiên sự gia tăng là không đáng kể (độ chính xác tăng từ 86,6% lên 88,7% khi phần trăm dữ liệu huấn luyện tăng từ 50% lên 90%)

Khảo sát: **Tham số C và kernel**

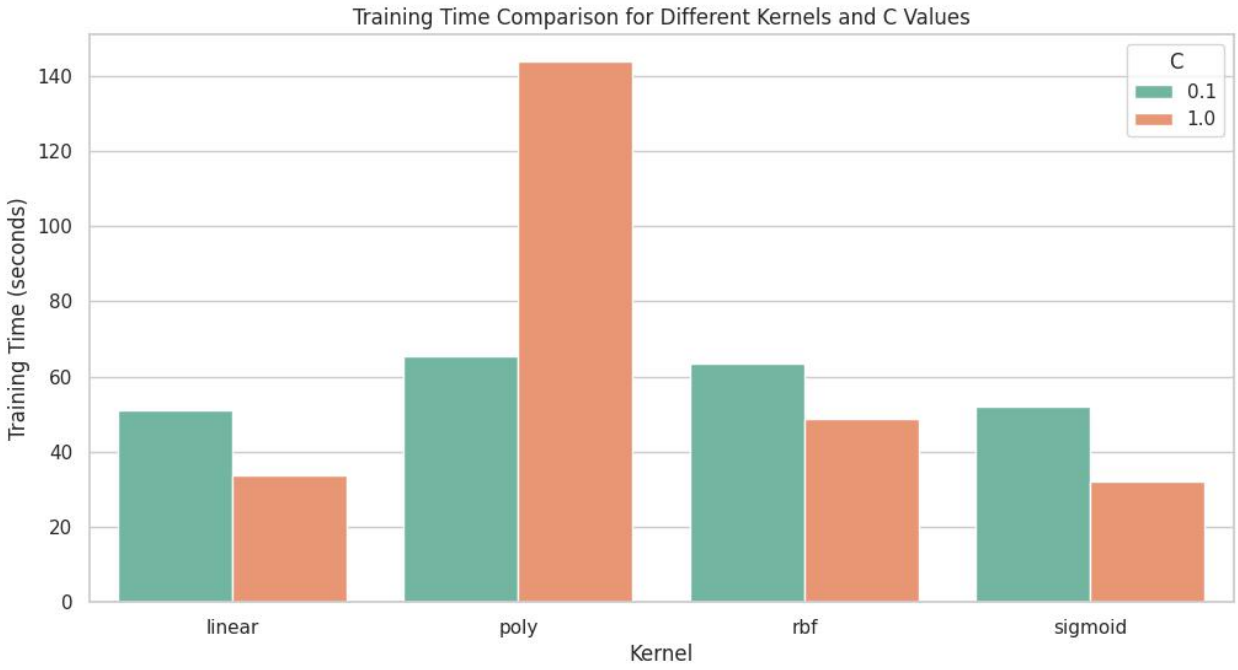
C	kernel	accuracy	training_time
0.1	linear	0.865	51.13
	poly	0.485	65.55
	rbf	0.800	63.50
	sigmoid	0.867	52.09
1	linear	0.889	33.66
	poly	0.836	143.76
	rbf	0.892	48.74

	sigmoid	0.887	32.05



Nhận xét:

- Đối với tham số **C**
  - Với C=1, thuật toán có độ chính xác cao hơn so với C=0.1
  - Điều này có thể lý giải bởi giá trị của C càng cao, thuật toán càng chú trọng phân loại đúng các điểm dữ liệu huấn luyện.
- Đối với tham số **kernel**
  - Cả 3 hàm kernel “linear", “rbf" và “sigmoid" có độ chính xác gần như bằng nhau
  - Ngoại lệ: Khi kernel=”poly, C=0.1 có độ chính xác thấp hơn đáng kể so với C=1.



Nhận xét:

- Đối với tham số **C**:
  - Nhìn chung, C=0.1 cần thời gian huấn luyện lâu hơn C=1, trừ trường hợp kernel=”poly"
- Đối với tham số **kernel**:
  - Thời gian huấn luyện gần như giống nhau đối với kernel “linear”, “rbf" và “sigmoid"

3.4 Thuật toán CRF

3.4.1 Thông tin lý thuyết

Khái niệm:

Conditional Random Fields (CRFs) là một loại phương pháp mô hình thống kê thường được áp dụng trong nhận diện mẫu và máy học, được sử dụng cho việc dự đoán có cấu trúc. Trong khi một bộ phân loại dự đoán một nhãn cho một mẫu duy nhất mà không xem xét các mẫu "lân cận", CRF có thể xem xét ngữ cảnh.

CRFs là thuật toán xác suất có điều kiện. Sự khác biệt ở mô hình này là sự phân phối xác suất theo điều kiện P(y/x), và xác suất này sẽ cố gắng đưa về xác suất bình thường: P(y, x).

Mô hình hồi quy Logistic, SVM, CRFs là những thuật toán phân loại có điều kiện. Naive Bayes, HMMs là những thuật toán phân

loại sinh mẫu. CRFs được sử dụng cho việc gắn nhãn NREs và POS.

Trong thuật toán CRFs, đầu vào là tập hợp các thuộc tính (dạng số thực) từ tập dữ liệu đầu vào theo một quy tắc. Trọng số của biểu thức với các thuộc tính đầu vào cùng các nhãn đã được gắn thẻ trước đó và task sẽ được dùng để dự đoán cho việc nhãn gắn hiện tại. Ta sẽ ước lượng trọng số sao cho chỉ số likelihood của nhãn trong bộ dữ liệu train là cực đại.

Hàm mục tiêu trong thuật toán sẽ xác định nhãn cho mỗi từ trong câu. Hãy tưởng tượng với một ví dụ sau: chữ cái đầu tiên của từ có phải là chữ cái viết hoa không?, các tiền tố và hậu tố của từ có dạng như thế nào?, …

Trong CRFs, chúng ta cũng xây dựng dự đoán nhãn từ hiện tại theo nhãn của các từ trước đó. Các trọng số của mô hình sẽ được hợp lý nhất.

sklearn_crfsuite.CRF() là một lớp được cung cấp bởi thư viện scikit-learn cho Python, cụ thể trong module sklearn_crfsuite.

- **Khởi tạo:**
  Khi tạo một thể hiện của sklearn_crfsuite.CRF(), bạn có thể cung cấp các tham số khác nhau để cấu hình hành vi của mô hình CRF. Các tham số này bao gồm các hệ số regularization L1 và L2, số lần lặp tối đa trong quá trình đào tạo và thuật toán sử dụng.

- **Huấn luyện:**

  Sau khi tạo thể hiện CRF, bạn có thể đào tạo nó trên bộ dữ liệu có nhãn của mình bằng cách sử dụng phương thức fit. Điều này bao gồm việc cung cấp các chuỗi đầu vào cùng với các nhãn tương ứng. CRF sau đó sẽ học cách thực hiện dự đoán dựa trên ngữ cảnh và mối quan hệ trong các chuỗi.

- **Dự đoán:**

  Sau khi được đào tạo, bạn có thể sử dụng CRF đã đào tạo để thực hiện dự đoán trên các chuỗi mới bằng cách sử dụng phương thức predict. Mô hình xem xét thông tin ngữ cảnh

trong các chuỗi đầu vào để đưa ra dự đoán cho mỗi phần tử trong chuỗi.

Danh sách siêu tham số:

Tên tham số	
algorithm	Xác định thuật toán sử dụng trong quá trình tối ưu hóa. Ví dụ: 'lbfgs' (Limited-memory Broyden-Fletcher-Goldfarb-Shanno), 'l2sgd' (Stochastic Gradient Descent với L2 regularization),...
c1 và c2	Các hệ số regularization L1 và L2. c1 áp dụng cho trọng số của các tính năng (features), trong khi c2 áp dụng cho trọng số của các chuyển trạng thái (state transitions). Những giá trị nhỏ giúp mô hình tránh overfitting.
max_iterations	Số lần lặp tối đa trong quá trình huấn luyện. Nếu quá trình huấn luyện không hội tụ trước khi đạt tới số lần lặp tối đa, quá trình sẽ dừng lại.
all_possible_transitions	Nếu được đặt là True, cho phép tất cả các chuyển trạng thái có thể trong dữ liệu đào tạo. Nếu là False, chỉ các chuyển trạng thái xuất hiện trong dữ liệu đào tạo sẽ được xem xét.
feature_minfreq	Số lần xuất hiện tối thiểu của một tính năng để nó được xem xét trong quá trình đào tạo. Tính năng có tần suất thấp hơn sẽ được loại bỏ.
verbose	Nếu được đặt là True, hiển thị thông tin về quá trình huấn luyện, giúp theo dõi tiến trình.
delta	Điều chỉnh sự hội tụ của thuật toán. Giá trị nhỏ hơn có thể dẫn đến hội tụ nhanh hơn, nhưng cũng có thể tăng thời gian tính toán.

3.4.2 Phân tích áp dụng

- CRF thường được sử dụng chủ yếu cho bài toán gán nhãn chuỗi (sequence labeling) như named entity recognition hoặc part-of-speech tagging. Chúng không phải là lựa chọn phổ biến cho bài toán classification thông thường. Nên đối với bài toán classification nên xem xét sử dụng các mô hình khác như SVM, Decision tree, Random forest, logistic regression, hoặc các mô hình học sâu như neural networks.
- Trong khi CRF giữ thông tin về mối quan hệ không gian và thời gian giữa các nhãn, các mô hình classification truyền thống thường tập trung vào đặc điểm đặc trưng độc lập. Điều này đồng nghĩa với việc CRF sẽ không hiệu quả bằng các mô hình khác khi đối mặt với các bài toán classification truyền thống.
- Ngoài ra, xét với bài toán cụ thể trên. Với tập dữ liệu đã pre_processing không phù hợp với mô hình của thư viện sklearn_crfsuite có sẵn (không fit với mô hình có sẵn). Vì:
    - Lấy ví dụ bài toán xác định nhãn POS bằng CRF, chúng ta sẽ xây dựng một bộ từ điển với nhãn được gắn cho mỗi từ trong câu:
        - Chữ cái đầu tiên có phải là chữ viết hoa không (Danh từ riêng có chữ cái đầu tiên viết hoa)?
        - Đó có phải là từ đầu tiên trong câu không?
        - Đó có phải là từ cuối cùng không?
        - Từ này có chứa cả chữ và số không?
        - Từ này có dấu gạch nối ở giữa không (phần này được áp dụng trong tiếng anh, những tính từ dài thường có dấu gạch nối: ví dụ: fast-growing, slow-moving).
        - Các chữ cái đều được viết hoa toàn bộ?
        - Đó là một số?
        - Bốn hậu tố và tiền tố của từ là gì? (Trong tiếng anh, những từ có đuôi “ed” thường là động từ; những từ kết thúc bằng “ous” thường là tính từ…)
    - Các câu hỏi trên sẽ đại diện cho 1 đặc trưng của 1 từ. Như trên chúng ta dễ dàng thấy được miền giá trị của các đặc trưng hoàn toàn rời rạc.

- Với data của bài toán ban đầu là 1 ma trận TF_IDF thì giá trị của nó không rời rạc (hay liên tục), bên cạnh đó chúng ta chỉ có 1 đặc trưng xét tới là TF_IDF nên CRF ở đây chưa thực sự hiệu quả.

- Chúng ta cần phải có 1 quá trình pre_processing để có thể fit được data trên với mô hình CRF có sẵn. Tuy nhiên, qua tìm hiểu thì mô hình CRF không đem lại hiểu quả trong bài toán classification này nên chúng ta sẽ dừng lại và không thực nghiệm trên data hiện tại.

3.5 Kết quả tổng quát

3.5.1 So sánh thuật toán

Với:

- max_features=5000
- criterion=”entropy"
- n_estimators=100
- C=1
- kernel=”rbf"

Bảng so sánh thuật toán:

Tên thuật toán	Training Time	Accuracy
Decision Tree	4.88	0.715
Random Forest	402.95	0.817
SVM	49.60	0.892

Nhận xét:

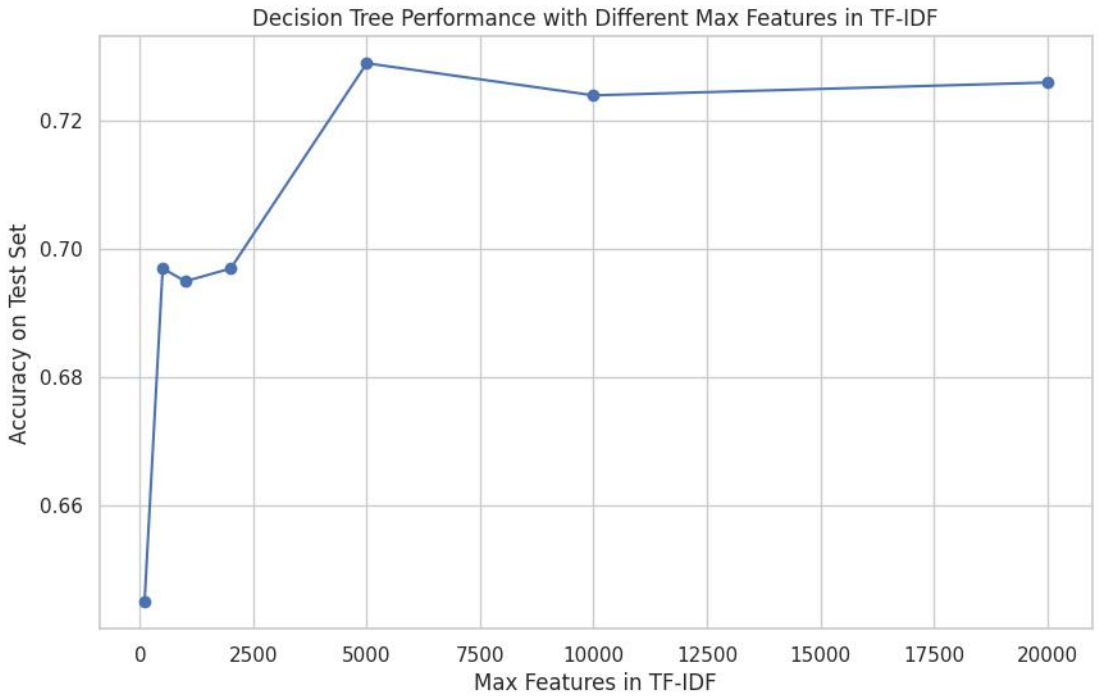
- Đối với Training Time:
  - Random Forest tốn nhiều thời gian chạy nhất (xấp xỉ 10 lần SVM và 100 lần Decision Tree).
  - Điều này có thể lý giải bởi so với Decision Tree (chỉ lấy kết quả từ 1 cây duy nhất) thì Random Forest phải tổng hợp kết quả từ 100 cây (n_estimators=100), do đó cần nhiều hơn gần 100 lần thời gian huấn luyện.

- Đối với Accuracy:
  - Đối với hai thuật toán sử dụng cây quyết định là Decision Tree và Random Forest thì Random Forest mang lại độ chính xác cao hơn (81,7% so với 71,5%).
  - Tuy nhiên, SVM vẫn là thuật toán ưu việt nhất với độ chính xác (89,2%) và chỉ tốn 1/10 thời gian huấn luyện so với Random Forest.

3.5.2 Khảo sát tiêu chí và mô hình

- Tiêu chí: max_features (tf-idf)
- Mô hình: Decision Tree
  - max_features=5000
  - criterion=”entropy"

max_features (tf-idf)	accuracy	execute time
100	0.645	2.14
500	0.697	3.74
1,000	0.695	5.62
2,000	0.697	9.70
5,000	0.729	11.20
10,000	0.724	6.17
20,000	0.726	2.47



Nhận xét:

- Đối với Accuracy:
  - Nhìn chung, max_features (tf-idf) tăng tỉ lệ thuận với Accuracy
  - Giá trị tối ưu của max_features (tf-idf) là 5,000
- Đối với Execute time:
  - Có 2 xu hướng:
    - max_features (tf-idf) từ 100 đến 5,000: max_features (tf-idf) tăng tỉ lệ thuận với execute time
    - max_features (tf-idf) từ 5,000 trở lên: max_features (tf-idf) tăng trong khi execute time giảm
  - Như vậy, max_features (tf-idf)=5,000 mang lại accuracy cao nhất (72,9%), cùng với execute time lớn nhất.
  - max_features (tf-idf)=20,000 có độ chính xác 72,6% (gần bằng 72,9%), tuy nhiên execute time chỉ bằng ⅕ execute time khi max_features (tf-idf)=5,000.

Tài liệu tham khảo

Decision Tree scikit-learn documentation: https://scikit-learn.org/stable/modules/tree.html

Random Forest scikit-learn documentation: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Support Vector Classification scikit-learn documentation: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

CRFsuit scikit-learn: https://sklearn-crfsuite.readthedocs.io/en/latest/index.html

Difference between min_samples_split and min_samples_leaf: https://stackoverflow.com/questions/46480457/difference-between-min-samples-split-and-min-samples-leaf-in-sklearn-decisiontre

What happens when bootstrapping isn't used in sklearn.RandomForestClassifier? https://stats.stackexchange.com/questions/354336/what-happens-when-bootstrapping-isnt-used-in-sklearn-randomforestclassifier

parameter C in SVM & standard to find best parameter: https://stackoverflow.com/questions/12809633/parameter-c-in-svm-standard-to-find-best-parameter

_ The end _