**Chương 2: Tiền xử lý và Phân tích Khám phá Dữ liệu Bộ dữ liệu Phê duyệt Tín dụng**

**2.1. Giới thiệu**

Trong lĩnh vực học máy, hiệu quả của mô hình dự báo phụ thuộc cốt yếu vào chất lượng của dữ liệu đầu vào. Dữ liệu thô, như bộ dữ liệu "Credit Approval" (UCI Repository ID: 27) được sử dụng trong nghiên cứu này, thường tồn tại ở trạng thái không hoàn chỉnh, chứa đựng nhiễu và chưa được cấu trúc tối ưu cho quá trình phân tích mô hình. Do đó, việc thực hiện các bước tiền xử lý dữ liệu một cách hệ thống là yêu cầu tiên quyết và mang tính nền tảng. Chương này trình bày chi tiết quy trình tiền xử lý được áp dụng, bao gồm hai giai đoạn chính: làm sạch dữ liệu (Data Cleaning) và phân tích khám phá dữ liệu thông qua trực quan hóa (Exploratory Data Analysis - EDA). Mục tiêu của chương là chuyển đổi dữ liệu thô thành một dạng thức chuẩn hóa, đáng tin cậy, đồng thời thu được những hiểu biết sâu sắc về đặc tính nội tại của dữ liệu, tạo tiền đề vững chắc cho giai đoạn xây dựng và đánh giá mô hình ở các chương sau.

Bộ dữ liệu bao gồm 15 thuộc tính mô tả (A1-A15) và một thuộc tính mục tiêu (A16), thể hiện quyết định phê duyệt tín dụng ('+' hoặc '-'). Các thuộc tính này có bản chất đa dạng, được phân loại thành:

* **Thuộc tính Định lượng (Quantitative/Continuous):** A2, A3, A8, A11, A14, A15.
* **Thuộc tính Định tính (Qualitative/Categorical):** A1, A4, A5, A6, A7, A9, A10, A12, A13.

**2.2. Làm sạch Dữ liệu (Data Cleaning)**

Giai đoạn làm sạch dữ liệu tập trung vào việc nhận diện và hiệu chỉnh các điểm bất thường, thiếu sót và không nhất quán trong tập dữ liệu gốc. Quá trình này, được thực thi thông qua module clean\_data.py, đảm bảo tính toàn vẹn và độ tin cậy của dữ liệu, bao gồm các tác vụ cụ thể sau:

1. **Xử lý Giá trị Thiếu (Handling Missing Values):** Các giá trị thiếu, thường được mã hóa bằng ký hiệu '?' trong bộ dữ liệu này, là một trở ngại phổ biến. Các chiến lược xử lý riêng biệt được áp dụng:
   * *Đối với thuộc tính định lượng (A2, A3, A8, A11, A14, A15):* Kỹ thuật **thay thế bằng giá trị trung vị (Median Imputation)** được lựa chọn (SimpleImputer(strategy='median')) do tính bền vững trước các giá trị ngoại lệ, giúp bảo toàn hiệu quả hơn hình thái phân phối gốc.
   * *Đối với thuộc tính định tính (A1, A4-A7, A9-A10, A12-A13):* Sau khi chuẩn hóa các ký hiệu thiếu về định dạng np.nan, chiến lược **thay thế bằng giá trị phổ biến nhất (Mode Imputation)** được triển khai (SimpleImputer(strategy='most\_frequent')), phù hợp với bản chất hạng mục của dữ liệu.
2. **Xử lý Dữ liệu Trùng lặp (Handling Duplicate Records):** Sự tồn tại của các bản ghi giống hệt nhau có thể gây sai lệch. Các bản ghi này được xác định bằng phương thức .duplicated() và loại bỏ hoàn toàn khỏi tập dữ liệu, đảm bảo tính nhất quán giữa X và y.
3. **Xử lý Giá trị Ngoại lệ (Handling Outliers in Quantitative Attributes):** Các thuộc tính định lượng có thể chứa các giá trị cực đoan. **Phương pháp Khoảng Tứ phân vị (IQR)** được sử dụng để giảm thiểu tác động này. Thay vì loại bỏ, kỹ thuật **cắt tỉa (clipping)** thông qua .clip() được áp dụng, thay thế các giá trị nằm ngoài phạm vi [, ] bằng giá trị ngưỡng gần nhất.
4. **Mã hóa Biến Mục tiêu (Target Variable Encoding):** Biến mục tiêu A16, với các giá trị hạng mục '+' và '-', được chuyển đổi sang dạng số học (ví dụ: 1 và 0) bằng LabelEncoder để phù hợp với yêu cầu đầu vào của thuật toán.

Kết thúc quy trình làm sạch, tập dữ liệu X\_processed và y\_encoded đã được chuẩn bị, không còn chứa giá trị thiếu, bản ghi trùng lặp, các giá trị ngoại lệ đã được kiểm soát và biến mục tiêu đã được mã hóa, tạo điều kiện thuận lợi cho giai đoạn phân tích tiếp theo.

**2.3. Phân tích Khám phá Dữ liệu thông qua Trực quan hóa (Exploratory Data Analysis via Visualization)**

Sau giai đoạn làm sạch, phân tích khám phá dữ liệu (EDA) thông qua trực quan hóa được tiến hành nhằm thu thập những hiểu biết sâu sắc về đặc tính của các thuộc tính đã qua xử lý. Module visualize\_data.py tạo ra các biểu đồ cung cấp cái nhìn đa chiều về dữ liệu:

1. **Khảo sát Phân phối Dữ liệu Định lượng:**
   * *Biểu đồ Histogram và Ước lượng Mật độ Kernel (KDE):* Hình 2.1 trình bày phân phối của 6 thuộc tính định lượng (A2, A3, A8, A11, A14, A15). Đáng chú ý, hầu hết các thuộc tính, đặc biệt là A15, A14, A11, và A8, thể hiện **độ lệch dương (right-skewed)** rõ rệt, với dữ liệu tập trung ở các giá trị thấp và có đuôi kéo dài về phía giá trị cao. Các thuộc tính A15, A11, và A8 còn có một đỉnh nhọn gần giá trị 0. Thuộc tính A2 có vẻ gần với phân phối chuẩn hơn nhưng vẫn lệch phải.
   * *Hàm ý:* Độ lệch đáng kể này gợi ý sự cần thiết của các kỹ thuật chuẩn hóa dữ liệu (scaling) mạnh mẽ như RobustScaler (ít nhạy cảm với outliers) hoặc các phép biến đổi dữ liệu (ví dụ: log transformation) để cải thiện hiệu suất các mô hình học máy sau này.

* *(Chèn Hình 2.1: Biểu đồ phân phối các đặc trưng số tại đây)*

1. **Tóm tắt Thống kê và Nhận diện Ngoại lệ:**
   * *Biểu đồ Hộp (Box Plot):* Hình 2.2 cung cấp tóm tắt trực quan về phân phối ngũ vị của các thuộc tính định lượng. Biểu đồ này làm nổi bật hai điểm chính:
     + **Sự khác biệt lớn về thang đo (Scale):** Phạm vi giá trị và khoảng tứ phân vị (IQR) của A15 và A14 lớn hơn đáng kể so với các thuộc tính còn lại. Điều này tái khẳng định tầm quan trọng của việc chuẩn hóa dữ liệu để các thuộc tính có đóng góp tương đương trong quá trình mô hình hóa.
     + **Xác nhận Độ lệch và Ảnh hưởng Xử lý Ngoại lệ:** Vị trí của đường trung vị và độ dài của "râu" trên trong hầu hết các hộp (A15, A14, A11, A8, A3) xác nhận tính chất lệch phải. Mặc dù đã áp dụng phương pháp IQR clipping, sự biến động lớn, đặc biệt ở A15 và A14, cho thấy dữ liệu vẫn có độ phân tán cao.

* *(Chèn Hình 2.2: Boxplot các đặc trưng số sau xử lý tại đây)*

1. **Phân tích Mối quan hệ giữa các Thuộc tính Định lượng:**
   * *Ma trận Tương quan và Bản đồ Nhiệt (Correlation Matrix & Heatmap):* Hình 2.3 hiển thị ma trận tương quan Pearson giữa các cặp thuộc tính định lượng. Phân tích bản đồ nhiệt cho thấy:
     + Mối quan hệ tuyến tính **dương** đáng kể nhất là giữa A11 và A15 (r ≈ 0.41), tiếp theo là giữa A8 và A2 (r ≈ 0.36), A8 và A11 (r ≈ 0.33).
     + Mối quan hệ tuyến tính **âm** mạnh nhất, mặc dù vẫn ở mức yếu, là giữa A3 và A14 (r ≈ -0.27).
     + Nhìn chung, các hệ số tương quan chủ yếu ở mức **yếu đến trung bình**. Không có cặp biến nào thể hiện tương quan tuyến tính rất mạnh (|r| > 0.7). Điều này làm giảm bớt lo ngại về hiện tượng **đa cộng tuyến (multicollinearity)** nghiêm trọng giữa các thuộc tính định lượng này, cho thấy mỗi thuộc tính có thể đóng góp thông tin tương đối độc lập cho mô hình.

* *(Chèn Hình 2.3: Ma trận tương quan giữa các đặc trưng tại đây)*

*(Nếu có, thêm phần phân tích biến định tính ở đây, ví dụ: "Phân tích biểu đồ cột cho các biến định tính (không hiển thị ở đây) cho thấy... Đặc biệt, biểu đồ tần suất cho biến mục tiêu A16 (sau mã hóa) chỉ ra rằng lớp '+' xuất hiện nhiều hơn/ít hơn lớp '-', cho thấy sự mất cân bằng/cân bằng tương đối giữa các lớp.")*

**2.4. Kết luận Chương**

Chương này đã trình bày chi tiết quy trình tiền xử lý bộ dữ liệu "Credit Approval", bao gồm các bước làm sạch dữ liệu thiết yếu và phân tích khám phá thông qua trực quan hóa. Quá trình làm sạch đã giải quyết các vấn đề về giá trị thiếu, dữ liệu trùng lặp và kiểm soát ảnh hưởng của các giá trị ngoại lệ, đồng thời chuẩn hóa định dạng biến mục tiêu. Phân tích khám phá dữ liệu, dựa trên các biểu đồ phân phối, box plot và ma trận tương quan, đã cung cấp những hiểu biết quan trọng về đặc điểm của dữ liệu: hầu hết các biến định lượng đều lệch phải, có sự khác biệt đáng kể về thang đo, và mức độ tương quan tuyến tính giữa chúng không quá cao. Những phát hiện này không chỉ xác nhận sự cần thiết của các bước chuẩn hóa dữ liệu tiếp theo mà còn cung cấp nền tảng thông tin vững chắc cho việc lựa chọn, xây dựng và diễn giải các mô hình học máy sẽ được trình bày trong các chương kế tiếp.

**Chương 3: Chuẩn bị Dữ liệu cho Mô hình hóa**

**3.1. Giới thiệu**

Sau khi dữ liệu đã được làm sạch ở Chương 2, bước tiếp theo là chuẩn bị dữ liệu ở định dạng phù hợp và tối ưu cho quá trình huấn luyện các mô hình học máy. Giai đoạn này, được thực hiện trong module preprocessing.py, bao gồm ba tác vụ chính:

1. **Phân chia Dữ liệu (Data Splitting):** Chia bộ dữ liệu thành hai tập riêng biệt: tập huấn luyện (training set) và tập kiểm tra (test set).
2. **Chuyển đổi Dữ liệu Định tính (Categorical Data Conversion):** Biến đổi các thuộc tính định tính sang dạng số học mà các thuật toán có thể hiểu được.
3. **Chuẩn hóa/Co giãn Dữ liệu Định lượng (Numerical Data Scaling):** Đưa các thuộc tính định lượng về cùng một thang đo (scale) để tránh sự thiên vị của mô hình đối với các thuộc tính có giá trị lớn.

Các bước này đóng vai trò quan trọng trong việc đảm bảo mô hình được huấn luyện một cách hiệu quả và được đánh giá một cách khách quan về khả năng tổng quát hóa trên dữ liệu mới.

**3.2. Phân chia Tập dữ liệu Huấn luyện và Kiểm tra (Train-Test Split)**

* **Lý thuyết:** Mục tiêu cốt lõi của việc xây dựng mô hình học máy là tạo ra một mô hình có khả năng **tổng quát hóa (generalize)** tốt, tức là dự đoán chính xác trên những dữ liệu mới mà nó chưa từng thấy trong quá trình huấn luyện. Để đánh giá khả năng này một cách đáng tin cậy, chúng ta cần tách một phần dữ liệu gốc ra làm **tập kiểm tra (test set)**. Mô hình sẽ chỉ được huấn luyện trên phần còn lại, gọi là **tập huấn luyện (training set)**. Sau khi huấn luyện xong, hiệu năng của mô hình sẽ được đánh giá trên tập kiểm tra. Cách làm này giúp tránh việc đánh giá quá lạc quan về hiệu năng của mô hình (do kiểm tra trên chính dữ liệu đã dùng để học). Trong bài toán phân loại, việc phân chia cần đảm bảo tỷ lệ các lớp trong tập huấn luyện và tập kiểm tra tương tự như trong tập dữ liệu gốc. Kỹ thuật này gọi là **phân chia theo tầng (stratified splitting)**, được thực hiện bằng cách đặt tham số stratify=y. Tham số test\_size xác định tỷ lệ dữ liệu dành cho tập kiểm tra (ví dụ: 0.25 nghĩa là 25%). Tham số random\_state đảm bảo rằng việc phân chia là ngẫu nhiên nhưng có thể tái lập được (kết quả chia giống nhau mỗi lần chạy code với cùng random\_state).
* **Thực hiện (trong preprocessing.py):** Hàm train\_test\_split từ thư viện Scikit-learn được sử dụng:
* X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
   X, y, test\_size=0.25, random\_state=42, stratify=y  
  )
* Ở đây, dữ liệu X (đặc trưng đã làm sạch) và y (nhãn đã mã hóa) được chia thành X\_train, y\_train (75% dữ liệu) dùng để huấn luyện và X\_test, y\_test (25% dữ liệu) dùng để đánh giá cuối cùng. Việc sử dụng stratify=y đảm bảo tỷ lệ các lớp phê duyệt tín dụng ('+' và '-') trong hai tập là tương đương nhau. random\_state=42 được dùng để kết quả phân chia nhất quán qua các lần chạy.

**3.3. Chuyển đổi và Chuẩn hóa Đặc trưng (Feature Transformation and Scaling)**

Các mô hình học máy thường yêu cầu đầu vào là dữ liệu số và có thể nhạy cảm với sự khác biệt về thang đo giữa các đặc trưng. Do đó, cần áp dụng các phép biến đổi phù hợp.

1. **Chuyển đổi Đặc trưng Định tính (Categorical Feature Encoding):**
   * **Lý thuyết:** Các thuộc tính định tính (A1, A4, A5, A6, A7, A9, A10, A12, A13) cần được chuyển đổi sang dạng số. Phương pháp **One-Hot Encoding (OHE)** được lựa chọn. OHE tạo ra các cột nhị phân (0 hoặc 1) mới cho mỗi giá trị hạng mục duy nhất trong một cột gốc. Ưu điểm của OHE là nó không tạo ra mối quan hệ thứ bậc giả tạo giữa các hạng mục (điều mà Label Encoding có thể gây ra). Tuy nhiên, nó có thể làm tăng đáng kể số chiều của dữ liệu nếu một thuộc tính có quá nhiều hạng mục duy nhất. Tham số handle\_unknown='ignore' giúp xử lý trường hợp gặp một giá trị hạng mục trong tập test mà không có trong tập train (các cột OHE tương ứng sẽ bằng 0). sparse\_output=False yêu cầu kết quả trả về là mảng NumPy dày đặc thay vì ma trận thưa.
   * **Thực hiện:** OneHotEncoder(handle\_unknown='ignore', sparse\_output=False) được tích hợp vào categorical\_pipeline.
2. **Chuẩn hóa Đặc trưng Định lượng (Numerical Feature Scaling):**
   * **Lý thuyết:** Các thuộc tính định lượng (A2, A3, A8, A11, A14, A15) có thang đo rất khác nhau (như đã thấy trong Hình 2.2 ở Chương 2). Nhiều thuật toán (như Logistic Regression, SVM, KNN, và các thuật toán dựa trên gradient descent) hoạt động hiệu quả hơn hoặc hội tụ nhanh hơn khi các đặc trưng có cùng một phạm vi giá trị. Dựa trên kết quả phân tích khám phá (Hình 2.1 và 2.2) cho thấy sự hiện diện của độ lệch và khả năng có outliers ngay cả sau khi clipping, **RobustScaler** được chọn làm phương pháp chuẩn hóa. RobustScaler sử dụng trung vị (median) và khoảng tứ phân vị (IQR) để co giãn dữ liệu (). Ưu điểm của nó là **ít bị ảnh hưởng bởi các giá trị ngoại lệ** hơn so với các phương pháp như StandardScaler (dùng mean và std) hay MinMaxScaler (dùng min và max).
   * **Thực hiện:** RobustScaler() được tích hợp vào numeric\_pipeline.
3. **Áp dụng Biến đổi một cách Nhất quán:**
   * **Lý thuyết:** Để tránh **rò rỉ dữ liệu (data leakage)**, các tham số cho việc chuyển đổi và chuẩn hóa (ví dụ: các hạng mục để OHE, giá trị median và IQR để scaling) phải được **học (fit) chỉ từ tập huấn luyện (X\_train)**. Sau đó, cùng các tham số đã học đó được áp dụng để **biến đổi (transform)** cả tập huấn luyện (X\_train) và tập kiểm tra (X\_test). Scikit-learn cung cấp các công cụ Pipeline và ColumnTransformer để thực hiện quy trình này một cách hiệu quả và an toàn. Pipeline cho phép kết hợp tuần tự các bước xử lý (ví dụ: imputation rồi scaling), trong khi ColumnTransformer cho phép áp dụng các pipeline khác nhau cho các tập con cột khác nhau (ví dụ: numeric\_pipeline cho cột số, categorical\_pipeline cho cột hạng mục).
   * **Thực hiện:**
     + Các cột số (numeric\_features) và cột hạng mục (categorical\_features) được xác định tự động từ X\_train.
     + numeric\_pipeline (chỉ chứa RobustScaler) và categorical\_pipeline (chỉ chứa OneHotEncoder) được định nghĩa.
     + ColumnTransformer được tạo để áp dụng đúng pipeline cho đúng loại cột.
     + Phương thức .fit\_transform() được gọi trên X\_train: Học các tham số (median, IQR, các categories) từ X\_train và đồng thời biến đổi X\_train.
     + Phương thức .transform() được gọi trên X\_test: Sử dụng các tham số đã học từ X\_train để biến đổi X\_test.
   * # Xác định loại cột  
     numeric\_features = X\_train.select\_dtypes(include=np.number).columns.tolist()  
     categorical\_features = X\_train.select\_dtypes(include='object').columns.tolist()  
       
     # Định nghĩa Pipelines  
     numeric\_pipeline = Pipeline([('scaler', RobustScaler())])  
     categorical\_pipeline = Pipeline([('onehot', OneHotEncoder(handle\_unknown='ignore', sparse\_output=False))])  
       
     # Định nghĩa ColumnTransformer  
     preprocessor = ColumnTransformer(  
      transformers=[  
      ('num', numeric\_pipeline, numeric\_features),  
      ('cat', categorical\_pipeline, categorical\_features)  
      ],  
      remainder='passthrough' # Giữ lại các cột không được chỉ định (nếu có)  
     )  
       
     # Áp dụng  
     X\_train\_processed = preprocessor.fit\_transform(X\_train)  
     X\_test\_processed = preprocessor.transform(X\_test)
     + Code cũng bao gồm khối try-except để lấy tên các đặc trưng mới được tạo ra sau OHE bằng preprocessor.get\_feature\_names\_out() và chuyển đổi kết quả thành DataFrame Pandas để dễ đọc và diễn giải hơn trong các bước sau (ví dụ: khi xem xét feature importance).

**3.4. Kết quả Dữ liệu sau Xử lý**

Kết thúc chương này, chúng ta thu được bốn thành phần dữ liệu đã sẵn sàng cho việc xây dựng và đánh giá mô hình:

* X\_train\_processed: Tập dữ liệu đặc trưng dùng để huấn luyện mô hình. Các đặc trưng định tính đã được mã hóa OHE, và các đặc trưng định lượng đã được chuẩn hóa bằng RobustScaler. Số lượng cột có thể tăng lên do OHE.
* y\_train: Tập nhãn tương ứng với X\_train\_processed.
* X\_test\_processed: Tập dữ liệu đặc trưng dùng để kiểm tra mô hình, đã được xử lý bằng *cùng một bộ biến đổi* đã học từ X\_train.
* y\_test: Tập nhãn tương ứng với X\_test\_processed.

Việc thực hiện cẩn thận các bước phân chia, chuyển đổi và chuẩn hóa này đảm bảo rằng dữ liệu đầu vào cho các mô hình học máy là phù hợp, nhất quán và quá trình đánh giá mô hình sẽ phản ánh đúng khả năng tổng quát hóa của chúng.

**Chương 4: Mô hình hóa và Huấn luyện Thuật toán Học máy**

**4.1. Giới thiệu**

Sau khi hoàn tất các bước tiền xử lý và chuẩn bị dữ liệu ở chương trước, giai đoạn tiếp theo là áp dụng các thuật toán học máy để xây dựng mô hình dự báo cho bài toán phân loại phê duyệt tín dụng. Dựa trên nguyên tắc học có giám sát (Supervised Learning), mục tiêu là học một hàm ánh xạ từ các đặc trưng đầu vào (X) tới nhãn lớp mục tiêu (y - chấp thuận/từ chối). Module train\_models.py thực hiện huấn luyện và đánh giá sơ bộ một tập hợp đa dạng các thuật toán phân loại, bao gồm cả mô hình phân biệt (Discriminative Models) và mô hình sinh (Generative Models). Chương này sẽ đi sâu vào cơ sở lý thuyết, cơ chế hoạt động, và việc lựa chọn các siêu tham số chính cho từng thuật toán được triển khai trong code.

**4.2. Hồi quy Logistic (Logistic Regression)**

* **Khái niệm:** Hồi quy Logistic (LR) là một thuật toán **phân loại có giám sát** nền tảng, đặc biệt hiệu quả cho các bài toán phân loại nhị phân. Nó thuộc nhóm **mô hình phân biệt**, tập trung vào việc mô hình hóa trực tiếp ranh giới quyết định giữa các lớp. Mục tiêu của LR không phải là dự đoán một giá trị liên tục mà là ước lượng **xác suất** một mẫu dữ liệu thuộc về lớp mục tiêu dương (thường ký hiệu là lớp "1").
* **Cơ chế hoạt động:**
  1. Mô hình tính toán một tổ hợp tuyến tính của các đặc trưng đầu vào, , trong đó là vector trọng số và là vector đặc trưng.
  2. Giá trị này sau đó được biến đổi bằng **hàm Sigmoid (hay hàm Logistic)**: . Hàm Sigmoid nén giá trị vào khoảng (0, 1), cho phép diễn giải kết quả như là xác suất .
  3. Quá trình huấn luyện bao gồm việc tìm bộ trọng số tối ưu bằng cách cực tiểu hóa một hàm mất mát phù hợp, thường là hàm **Cross-Entropy** (còn gọi là Log Loss), đo lường sự khác biệt giữa phân phối xác suất dự đoán và phân phối xác suất thực tế (nhãn 0 hoặc 1). Việc cực tiểu hóa Cross-Entropy tương đương với việc cực đại hóa hàm **Log-Likelihood** của dữ liệu.
  4. Thuật toán tối ưu hóa **Gradient Descent** (hoặc các biến thể như 'liblinear') được sử dụng để điều chỉnh lặp đi lặp lại các trọng số theo hướng làm giảm hàm mất mát.
* **Các siêu tham số chính (trong code models):**
  + C=0.05: Là tham số **nghịch đảo của cường độ điều chuẩn (Inverse of Regularization Strength)**. Điều chuẩn (thường là L1 hoặc L2) được thêm vào hàm mất mát nhằm ngăn chặn hiện tượng overfitting bằng cách giới hạn độ lớn của các trọng số . Giá trị C nhỏ (như 0.05 trong cài đặt này) biểu thị mức độ điều chuẩn mạnh hơn, khuyến khích mô hình đơn giản hơn, có khả năng giảm phương sai nhưng có thể tăng độ chệch.
  + max\_iter=2000: Xác định số vòng lặp tối đa cho phép thuật toán tối ưu hóa hội tụ đến nghiệm.
  + solver='liblinear': Chỉ định thuật toán tối ưu hóa cụ thể. 'liblinear' là một lựa chọn tốt cho các bài toán có kích thước vừa và nhỏ, hỗ trợ cả điều chuẩn L1 và L2.
  + class\_weight='balanced': Tự động điều chỉnh trọng số của các mẫu thuộc các lớp khác nhau trong hàm mất mát, tỷ lệ nghịch với tần suất xuất hiện của chúng. Tham số này rất quan trọng để giải quyết vấn đề **dữ liệu mất cân bằng (imbalanced data)**, tránh việc mô hình học thiên vị về phía lớp đa số.
* **Ưu điểm:**
  + Cung cấp đầu ra là xác suất, giúp đánh giá độ tin cậy của dự đoán và thiết lập ngưỡng quyết định linh hoạt.
  + Mô hình tương đối đơn giản, dễ diễn giải (thông qua các trọng số ).
  + Hiệu quả về mặt tính toán.
* **Nhược điểm:**
  + Giả định mối quan hệ tuyến tính giữa các đặc trưng và log-odds của xác suất, có thể không mô hình hóa tốt các mối quan hệ phi tuyến phức tạp.

**4.3. Cây Quyết định (Decision Tree)**

* **Khái niệm:** Cây quyết định (DT) là một **mô hình logic** mạnh mẽ trong học có giám sát, có thể sử dụng cho cả bài toán phân loại và hồi quy. Nó biểu diễn các quy tắc quyết định dưới dạng một cấu trúc cây phân cấp, giúp mô hình trở nên **minh bạch và dễ diễn giải**. DT thuộc nhóm **mô hình phân biệt**.
* **Cơ chế hoạt động:** Thuật toán xây dựng cây hoạt động bằng cách phân chia lặp đi lặp lại tập dữ liệu thành các tập con ngày càng nhỏ và đồng nhất hơn về lớp mục tiêu. Tại mỗi nút trong cây (bắt đầu từ nút gốc), thuật toán thực hiện các bước sau:
  1. Lựa chọn một thuộc tính và một ngưỡng chia (đối với thuộc tính liên tục) hoặc một tập hợp giá trị (đối với thuộc tính hạng mục) để tạo ra sự phân tách "tốt nhất" cho dữ liệu tại nút đó.
  2. Tiêu chí để đánh giá "tốt nhất" thường dựa trên việc đo lường và tối ưu hóa **độ thuần khiết (purity)** của các nút con sau khi chia. Các thước đo độ thuần khiết phổ biến bao gồm **Entropy** (và **Information Gain** - độ lợi thông tin, là sự giảm entropy) hoặc **Gini Index**. Mục tiêu là tìm phép chia mang lại các nút con có độ thuần khiết cao nhất (chứa chủ yếu các mẫu thuộc một lớp).
  3. Quá trình này được lặp lại một cách đệ quy trên các nút con cho đến khi một điều kiện dừng được thỏa mãn (ví dụ: nút đạt độ thuần khiết tuyệt đối, hết thuộc tính để chia, đạt độ sâu tối đa được quy định, hoặc số lượng mẫu trong nút quá nhỏ).
* **Các siêu tham số chính (trong code models):**
  + max\_depth=6: Giới hạn **độ sâu tối đa** mà cây có thể phát triển. Đây là một phương pháp điều chuẩn hiệu quả để **ngăn chặn overfitting**, tránh việc cây trở nên quá phức tạp và chỉ khớp tốt với dữ liệu huấn luyện mà không tổng quát hóa được. Giá trị 6 cho thấy một sự giới hạn tương đối về độ phức tạp.
  + min\_samples\_split=30: Quy định số lượng mẫu **tối thiểu** phải có trong một nút để nút đó được xem xét chia tiếp. Việc tăng giá trị này giúp ngăn cây học các quy tắc dựa trên số lượng mẫu quá nhỏ, làm tăng khả năng tổng quát hóa.
  + min\_samples\_leaf=15: Quy định số lượng mẫu **tối thiểu** phải hiện diện ở mỗi nút lá (nút cuối cùng không chia nữa). Điều này đảm bảo rằng mỗi dự đoán cuối cùng của cây được dựa trên một cơ sở dữ liệu đủ lớn, giúp làm mịn ranh giới quyết định và giảm overfitting.
  + class\_weight='balanced': Giải quyết vấn đề mất cân bằng lớp bằng cách gán trọng số khác nhau cho các lớp trong quá trình tính toán độ thuần khiết, đảm bảo các lớp thiểu số không bị bỏ qua.
* **Ưu điểm:**
  + Mô hình rất dễ hiểu, dễ diễn giải và có thể trực quan hóa.
  + Có khả năng xử lý cả dữ liệu số và dữ liệu hạng mục.
  + Không yêu cầu chuẩn hóa dữ liệu một cách nghiêm ngặt.
  + Có thể tự động nắm bắt các tương tác phi tuyến giữa các đặc trưng.
* **Nhược điểm:**
  + Rất dễ bị **overfitting** nếu không có các biện pháp điều chuẩn (như giới hạn độ sâu, số mẫu tối thiểu, hoặc cắt tỉa - pruning).
  + Mô hình có thể **không ổn định**, nghĩa là những thay đổi nhỏ trong dữ liệu huấn luyện có thể dẫn đến cấu trúc cây khác biệt đáng kể.

**4.4. Rừng Ngẫu nhiên (Random Forest)**

* **Khái niệm:** Rừng Ngẫu nhiên (RF) là một thuật toán **ensemble (tập hợp)** thuộc nhóm **mô hình phân biệt**, nổi tiếng về độ chính xác và khả năng chống overfitting. Nó cải thiện hiệu năng của một Cây Quyết định đơn lẻ bằng cách xây dựng nhiều cây trên các tập con dữ liệu khác nhau và kết hợp dự đoán của chúng. Kỹ thuật cốt lõi là **Bagging (Bootstrap Aggregating)** kết hợp với việc lựa chọn đặc trưng ngẫu nhiên.
* **Cơ chế hoạt động:**
  1. **Bootstrap Sampling:** Tạo ra n\_estimators tập dữ liệu huấn luyện mới bằng cách lấy mẫu ngẫu nhiên có hoàn lại từ tập huấn luyện gốc.
  2. **Random Feature Selection & Tree Building:** Với mỗi tập dữ liệu bootstrap, xây dựng một Cây Quyết định. Tuy nhiên, tại mỗi nút trong quá trình xây dựng cây, thay vì xem xét tất cả các đặc trưng, thuật toán chỉ chọn ngẫu nhiên một tập con các đặc trưng để tìm điểm chia tốt nhất. Điều này làm tăng sự đa dạng giữa các cây.
  3. **Aggregation:** Khi dự đoán cho một mẫu mới, mẫu đó được đưa qua tất cả các cây trong rừng. Kết quả phân loại cuối cùng được xác định bằng cách lấy **phiếu bầu đa số (majority voting)** từ dự đoán của từng cây.
* **Các siêu tham số chính (trong code models):**
  + n\_estimators=200: Số lượng cây quyết định được tạo ra trong rừng. Một số lượng lớn hơn thường dẫn đến mô hình ổn định và chính xác hơn, nhưng cũng làm tăng thời gian huấn luyện.
  + max\_depth=15, min\_samples\_split=15, min\_samples\_leaf=5: Các tham số này kiểm soát cấu trúc của từng cây riêng lẻ trong rừng. Chúng vẫn cần thiết để giới hạn độ phức tạp của các cây thành viên, mặc dù kỹ thuật ensemble đã giúp giảm overfitting tổng thể.
  + bootstrap=True: Kích hoạt việc lấy mẫu bootstrap, nền tảng của phương pháp Bagging.
  + class\_weight='balanced': Xử lý mất cân bằng lớp trong quá trình huấn luyện từng cây.
  + n\_jobs=-1: Cho phép sử dụng tất cả các lõi CPU có sẵn để huấn luyện các cây một cách song song, giúp giảm đáng kể thời gian huấn luyện.
* **Ưu điểm:**
  + Thường cho độ chính xác dự đoán rất cao và khả năng tổng quát hóa tốt.
  + Khả năng chống overfitting tốt hơn đáng kể so với một cây quyết định đơn lẻ.
  + Hoạt động hiệu quả với dữ liệu có số chiều cao và lượng mẫu lớn.
  + Cung cấp một thước đo hữu ích về **tầm quan trọng của đặc trưng (feature importance)**.
* **Nhược điểm:**
  + Mô hình là một "hộp đen", **kém diễn giải** hơn nhiều so với một cây quyết định đơn lẻ.
  + Yêu cầu nhiều tài nguyên tính toán (CPU, bộ nhớ) hơn để huấn luyện.

**4.5. Tăng cường Gradient (Gradient Boosting)**

* **Khái niệm:** Tăng cường Gradient (GB) cũng là một kỹ thuật **ensemble** mạnh mẽ, nhưng thuộc họ **Boosting**. Khác với Random Forest xây dựng cây song song, GB xây dựng các mô hình (thường là các cây quyết định nông, gọi là "weak learners") một cách **tuần tự**. Mỗi mô hình mới được huấn luyện để sửa chữa những sai sót (phần dư - residuals) của mô hình tổng hợp trước đó.
* **Cơ chế hoạt động:**
  1. Bắt đầu với một mô hình cơ sở đơn giản (ví dụ: dự đoán giá trị trung bình hoặc log-odds trung bình).
  2. Lặp đi lặp lại qua n\_estimators bước: a. Tính toán phần dư (gradient của hàm mất mát) dựa trên dự đoán của mô hình hiện tại. b. Huấn luyện một cây quyết định yếu mới để khớp với phần dư này. c. Cập nhật mô hình tổng hợp bằng cách thêm cây mới vào, thường được co lại bởi một **tỷ lệ học (learning rate)** nhỏ.
  3. Dự đoán cuối cùng là tổng hợp kết quả từ tất cả các cây yếu đã xây dựng.
* **Các siêu tham số chính (trong code models):**
  + n\_estimators=150: Số lượng cây yếu (hay số bước tăng cường) được thực hiện.
  + learning\_rate=0.1: Kiểm soát mức độ đóng góp của mỗi cây mới vào mô hình tổng thể. Giá trị nhỏ hơn đòi hỏi nhiều n\_estimators hơn nhưng thường giúp mô hình tổng quát hóa tốt hơn bằng cách thực hiện các bước cập nhật nhỏ hơn.
  + max\_depth=8, min\_samples\_split=15, min\_samples\_leaf=5: Giới hạn độ phức tạp của từng cây yếu. Trong GB, việc giữ các cây cơ sở tương đối đơn giản là rất quan trọng để tránh overfitting.
  + subsample=0.8: Kích hoạt **Stochastic Gradient Boosting**. Ở mỗi bước, cây yếu chỉ được huấn luyện trên một phần (ở đây là 80%) dữ liệu huấn luyện được lấy mẫu ngẫu nhiên. Điều này giúp tăng tính ngẫu nhiên, giảm phương sai và chống overfitting.
* **Ưu điểm:**
  + Thường đạt được hiệu năng dự đoán hàng đầu trên nhiều loại dữ liệu có cấu trúc.
  + Linh hoạt, có thể tùy chỉnh với nhiều hàm mất mát khác nhau.
* **Nhược điểm:**
  + Việc huấn luyện diễn ra tuần tự nên khó song song hóa hơn Random Forest, có thể tốn nhiều thời gian hơn.
  + Rất nhạy cảm với việc lựa chọn siêu tham số và dễ bị overfitting nếu không được tinh chỉnh cẩn thận.

**4.6. K Hàng xóm Gần nhất (K-Nearest Neighbors - KNN)**

* **Khái niệm:** KNN là một thuật toán học **dựa trên thực thể (instance-based)** hoặc **học lười (lazy learning)** đơn giản nhưng hiệu quả. Nó thuộc nhóm **mô hình phân biệt**. Thay vì học một hàm phân loại tường minh, KNN đưa ra dự đoán dựa trên các điểm dữ liệu huấn luyện gần nhất với điểm cần dự đoán.
* **Cơ chế hoạt động:**
  1. **Giai đoạn "huấn luyện":** Đơn giản là lưu trữ toàn bộ tập dữ liệu huấn luyện.
  2. **Giai đoạn dự đoán:** Đối với một điểm dữ liệu mới **x**: a. Tính toán khoảng cách từ **x** đến mọi điểm trong tập huấn luyện bằng một **hàm khoảng cách (Distance Function)** được chọn trước (ví dụ: Euclidean, Manhattan, Minkowski). b. Xác định **K** điểm huấn luyện có khoảng cách nhỏ nhất (tức là gần nhất) với **x**. c. Phân loại **x** dựa trên **lớp chiếm đa số (majority vote)** trong số K hàng xóm này.
* **Các siêu tham số chính (trong code models):**
  + n\_neighbors=7: Số lượng hàng xóm **(K)** được xem xét để đưa ra quyết định. Đây là siêu tham số quan trọng nhất, ảnh hưởng đến sự cân bằng giữa độ chệch và phương sai của mô hình.
  + weights='distance': Áp dụng cơ chế **trọng số theo khoảng cách**. Các hàng xóm gần hơn sẽ có ảnh hưởng lớn hơn đến kết quả bỏ phiếu so với các hàng xóm xa hơn. Lựa chọn khác là 'uniform', nơi tất cả K hàng xóm có trọng số bằng nhau.
  + p=2: Chỉ định sử dụng **khoảng cách Euclidean** khi tính toán sự tương đồng giữa các điểm.
  + n\_jobs=-1: Tận dụng khả năng xử lý song song để tăng tốc độ tìm kiếm hàng xóm, đặc biệt hữu ích với tập dữ liệu lớn.
* **Ưu điểm:**
  + Thuật toán đơn giản, dễ hiểu và dễ triển khai.
  + Phi tham số, không đưa ra giả định mạnh về phân phối dữ liệu.
  + Có khả năng học các ranh giới quyết định phức tạp và phi tuyến.
* **Nhược điểm:**
  + Chi phí tính toán trong giai đoạn dự đoán cao (phải so sánh với tất cả các điểm huấn luyện), làm cho nó **chậm với các tập dữ liệu lớn**.
  + **Rất nhạy cảm với thang đo của các đặc trưng**. Do đó, việc **chuẩn hóa dữ liệu (scaling)** là bước bắt buộc trước khi áp dụng KNN (đã thực hiện trong preprocessing.py bằng RobustScaler).
  + Bị ảnh hưởng bởi "lời nguyền số chiều" (curse of dimensionality) và các đặc trưng không liên quan.
  + **Không tạo ra một mô hình tường minh**, khó diễn giải cách nó đưa ra quyết định.
  + Yêu cầu lưu trữ toàn bộ dữ liệu huấn luyện, tốn bộ nhớ.

**4.7. Naive Bayes (Gaussian Naive Bayes)**

* **Khái niệm:** Naive Bayes (NB) là một họ các bộ phân loại xác suất dựa trên **Định lý Bayes**. Nó được gọi là "ngây thơ" (naive) vì đưa ra một giả định mạnh mẽ nhưng thường hiệu quả trong thực tế: các đặc trưng là **độc lập có điều kiện** với nhau khi biết lớp mục tiêu. NB thuộc nhóm **mô hình sinh (Generative Model)**, vì nó mô hình hóa phân phối xác suất của dữ liệu trong từng lớp.
* **Cơ chế hoạt động:**
  1. Sử dụng Định lý Bayes để tính xác suất hậu nghiệm của một lớp khi biết các đặc trưng : . Do không phụ thuộc vào lớp, việc phân loại tương đương với tìm lớp cực đại hóa .
  2. Áp dụng giả định độc lập ngây thơ: .
  3. Mô hình học các tham số từ dữ liệu huấn luyện: xác suất tiên nghiệm của mỗi lớp (thường là tần suất xuất hiện của lớp) và xác suất có điều kiện cho từng đặc trưng trong mỗi lớp .
  4. **Gaussian Naive Bayes (GaussianNB):** Là biến thể của NB được sử dụng khi các đặc trưng đầu vào là liên tục. Nó giả định rằng giá trị của mỗi đặc trưng liên tục , khi biết lớp , tuân theo một **phân phối chuẩn (Gaussian)**. Do đó, mô hình cần ước lượng tham số trung bình () và phương sai () của phân phối Gaussian này cho mỗi đặc trưng và mỗi lớp từ dữ liệu huấn luyện.
* **Các siêu tham số chính (trong code models):**
  + var\_smoothing=1e-9: Một giá trị nhỏ được cộng vào phương sai của các đặc trưng trong quá trình tính toán. Tham số này giúp **ổn định số học**, tránh các vấn đề có thể xảy ra khi một đặc trưng có phương sai bằng không trong một lớp nào đó (ví dụ, tất cả các mẫu của lớp đó có cùng giá trị cho đặc trưng đó).
* **Ưu điểm:**
  + Huấn luyện và dự đoán rất nhanh, hiệu quả về mặt tính toán.
  + Yêu cầu tương đối ít dữ liệu huấn luyện để ước lượng tham số.
  + Hoạt động tốt với dữ liệu có số chiều rất cao.
  + Thường cho kết quả tốt ngay cả khi giả định độc lập không hoàn toàn đúng.
* **Nhược điểm:**
  + Giả định độc lập giữa các đặc trưng là một hạn chế lớn, vì trong nhiều bài toán thực tế, các đặc trưng thường có mối liên hệ với nhau.
  + Đối với GaussianNB, giả định về phân phối chuẩn của dữ liệu liên tục có thể không chính xác, đặc biệt nếu dữ liệu bị lệch nhiều.

**4.8. Máy Vector Hỗ trợ (Support Vector Machine - SVM)**

* **Khái niệm:** SVM là một thuật toán phân loại thuộc nhóm **mô hình phân biệt**, tìm cách xác định một **siêu phẳng (hyperplane)** trong không gian đặc trưng sao cho nó phân tách tốt nhất các lớp dữ liệu. "Tốt nhất" ở đây thường có nghĩa là siêu phẳng có **lề (margin)** - khoảng cách đến các điểm dữ liệu gần nhất của mỗi lớp - là lớn nhất.
* **Cơ chế hoạt động:**
  1. Trong trường hợp dữ liệu phân tách tuyến tính, SVM tìm siêu phẳng phân cách với lề cực đại. Các điểm dữ liệu nằm trên hoặc gần lề nhất được gọi là **vector hỗ trợ (support vectors)**, vì chúng xác định vị trí của siêu phẳng.
  2. Khi dữ liệu không phân tách tuyến tính, SVM sử dụng **thủ thuật hạt nhân (kernel trick)**. Dữ liệu được ánh xạ một cách ngầm định vào một không gian có số chiều cao hơn, nơi hy vọng rằng chúng sẽ trở nên phân tách tuyến tính. Các hàm kernel phổ biến bao gồm tuyến tính, đa thức, và **hàm cơ sở bán kính (Radial Basis Function - RBF)** - một lựa chọn mạnh mẽ và linh hoạt cho các bài toán phi tuyến.
  3. Quá trình tìm siêu phẳng tối ưu là một bài toán tối ưu hóa lồi.
* **Các siêu tham số chính (trong code models):**
  + C=0.1: Tham số **điều chuẩn (Regularization Parameter)**. Nó kiểm soát sự đánh đổi giữa hai mục tiêu: (1) tối đa hóa độ rộng của lề và (2) giảm thiểu số lượng điểm bị phân loại sai hoặc nằm trong lề (trong trường hợp soft-margin SVM). Giá trị C nhỏ (như 0.1) cho phép một lề "mềm" hơn, chấp nhận nhiều lỗi phân loại hơn để đổi lấy lề rộng hơn, giúp tăng khả năng tổng quát hóa và giảm overfitting.
  + kernel='rbf': Chỉ định sử dụng hàm kernel RBF, cho phép SVM học các ranh giới quyết định phi tuyến phức tạp.
  + gamma='scale': Tham số của kernel RBF, xác định "phạm vi ảnh hưởng" của một vector hỗ trợ. 'scale' là một chiến lược tự động điều chỉnh gamma dựa trên phương sai của dữ liệu và số lượng đặc trưng, thường cho kết quả khởi đầu tốt. Giá trị gamma ảnh hưởng đến độ phức tạp của ranh giới quyết định.
  + probability=True: Bật cờ này để cho phép SVM tính toán ước lượng xác suất cho các dự đoán sau khi mô hình đã được huấn luyện (thường thông qua hiệu chuẩn Platt). Việc này làm tăng thời gian tính toán.
  + class\_weight='balanced': Xử lý mất cân bằng lớp bằng cách gán trọng số khác nhau cho các lớp trong bài toán tối ưu hóa.
  + max\_iter=1000: Giới hạn số vòng lặp tối đa cho thuật toán tối ưu hóa (solver) của SVM. Điều này có thể ngăn chặn thời gian huấn luyện quá dài đối với các tập dữ liệu khó hội tụ.
* **Ưu điểm:**
  + Rất hiệu quả trong không gian có số chiều cao.
  + Vẫn hiệu quả khi số chiều lớn hơn số lượng mẫu.
  + Tối ưu hóa dựa trên việc xác định các vector hỗ trợ, giúp tiết kiệm bộ nhớ trong giai đoạn dự đoán.
  + Linh hoạt với việc sử dụng các hàm kernel khác nhau.
* **Nhược điểm:**
  + Thời gian huấn luyện có thể trở nên rất lớn đối với các tập dữ liệu có kích thước lớn, đặc biệt khi sử dụng các kernel phi tuyến.
  + Hiệu năng **rất nhạy cảm** với việc lựa chọn hàm kernel và các giá trị siêu tham số (C, gamma), đòi hỏi quá trình tinh chỉnh (tuning) cẩn thận.
  + Mô hình SVM, đặc biệt với kernel phi tuyến, thường được coi là **khó diễn giải** ("hộp đen").

**4.9. Kết luận Chương**

Chương này đã cung cấp một cái nhìn tổng quan và phân tích chi tiết về 7 thuật toán học máy được áp dụng trong nghiên cứu này: Hồi quy Logistic, Cây Quyết định, Rừng Ngẫu nhiên, Tăng cường Gradient, K Hàng xóm Gần nhất, Naive Bayes (Gaussian), và Máy Vector Hỗ trợ. Các thuật toán này đại diện cho nhiều phương pháp tiếp cận khác nhau trong lĩnh vực học máy có giám sát, từ các mô hình tuyến tính, dựa trên quy tắc logic, ensemble, dựa trên khoảng cách, xác suất, cho đến các mô hình tối ưu hóa biên. Việc lựa chọn và cấu hình các siêu tham số ban đầu, như đã trình bày cho từng thuật toán trong module train\_models.py (ví dụ: C, max\_depth, n\_estimators, learning\_rate, n\_neighbors, kernel, gamma, class\_weight, var\_smoothing), là bước khởi đầu quan trọng, phản ánh sự cân nhắc giữa độ phức tạp mô hình, khả năng tổng quát hóa và các đặc thù của dữ liệu như sự mất cân bằng lớp. Hiệu năng thực nghiệm của các mô hình này trên tập dữ liệu kiểm tra sẽ được trình bày và phân tích sâu hơn trong chương kế tiếp, nhằm mục đích lựa chọn ra mô hình phù hợp và hiệu quả nhất cho bài toán phê duyệt tín dụng đang xem xét.

**Chương 5: Đánh giá và So sánh Hiệu năng Mô hình**

**5.1. Giới thiệu**

Sau khi huấn luyện các thuật toán học máy đa dạng trên tập dữ liệu huấn luyện (X\_train\_processed, y\_train) như đã trình bày ở Chương 4, bước tiếp theo và cực kỳ quan trọng là đánh giá hiệu năng của chúng một cách khách quan. Mục tiêu của chương này là đo lường khả năng tổng quát hóa của từng mô hình trên tập dữ liệu kiểm tra (X\_test\_processed, y\_test) - là dữ liệu mà mô hình chưa từng gặp trong quá trình huấn luyện. Việc đánh giá này sử dụng các độ đo (metrics) định lượng phù hợp cho bài toán phân loại nhị phân (phê duyệt hoặc từ chối tín dụng), cho phép so sánh hiệu quả giữa các phương pháp và đưa ra kết luận về mô hình phù hợp nhất cho bài toán. Quá trình này tuân theo nguyên tắc đánh giá mô hình trong quy trình học có giám sát (Supervised Learning).

**5.2. Phương pháp và Độ đo Đánh giá (Evaluation Metrics)**

Để đánh giá toàn diện hiệu năng của các mô hình phân loại, các độ đo sau đây đã được sử dụng (thực thi trong module evaluate.py):

1. **Accuracy:**
   * *Định nghĩa:* Tỷ lệ phần trăm số điểm dữ liệu được mô hình phân loại đúng trên tổng số điểm dữ liệu trong tập kiểm tra.
   * *Công thức:* Accuracy = (True Positives + True Negatives) / (True Positives + True Negatives + False Positives + False Negatives)
   * *Ý nghĩa:* Cung cấp một cái nhìn tổng quan về hiệu suất tổng thể. Trong bộ dữ liệu này, với tỷ lệ lớp tương đối cân bằng (307 lớp 0 và 383 lớp 1), Accuracy là một chỉ số tham khảo hữu ích.
2. **Precision:**
   * *Định nghĩa:* Trong số các điểm được mô hình dự đoán là thuộc lớp tích cực (ví dụ: lớp 1 - phê duyệt), tỷ lệ bao nhiêu điểm thực sự thuộc lớp đó.
   * *Công thức (cho lớp 1):* Precision = True Positives / (True Positives + False Positives)
   * *Ý nghĩa:* Đánh giá mức độ "đáng tin" khi mô hình dự đoán một điểm là tích cực. Precision cao có nghĩa là mô hình ít mắc lỗi **False Positive**. Trong bài toán tín dụng, False Positive có thể tương ứng với rủi ro phê duyệt một khoản vay không phù hợp.
3. **Recall (Sensitivity):**
   * *Định nghĩa:* Trong số tất cả các điểm thực sự thuộc lớp tích cực, tỷ lệ bao nhiêu điểm được mô hình dự đoán đúng là thuộc lớp đó.
   * *Công thức (cho lớp 1):* Recall = True Positives / (True Positives + False Negatives)
   * *Ý nghĩa:* Đánh giá khả năng mô hình "phát hiện" được tất cả các trường hợp tích cực. Recall cao có nghĩa là mô hình ít mắc lỗi **False Negative**. Trong bài toán tín dụng, False Negative có thể tương ứng với việc bỏ lỡ cơ hội kinh doanh khi từ chối khách hàng đủ điều kiện.
4. **F1-Score:**
   * *Định nghĩa:* Là trung bình điều hòa (harmonic mean) của Precision và Recall. Nó cung cấp một thước đo cân bằng duy nhất giữa hai chỉ số này.
   * *Công thức:* F1-Score = 2 \* (Precision \* Recall) / (Precision + Recall)
   * *Ý nghĩa:* F1-Score hữu ích khi cần cân bằng giữa việc giảm thiểu False Positives và False Negatives. Giá trị F1-Score cao cho thấy sự cân bằng tốt giữa Precision và Recall. Trong kết quả báo cáo, F1-Score được tính theo average='weighted', nghĩa là lấy trung bình F1-Score của từng lớp, có trọng số theo số lượng mẫu của lớp đó.
5. **Classification Report:**
   * *Nội dung:* Cung cấp giá trị Precision, Recall, F1-Score và Support (số lượng mẫu thực tế) cho *từng lớp riêng biệt* (lớp 0 và lớp 1). Điều này cho phép phân tích sâu hơn hiệu năng của mô hình đối với từng loại dự đoán. Báo cáo cũng bao gồm các giá trị trung bình (macro average, weighted average).
6. **Confusion Matrix:**
   * *Nội dung:* Là một bảng trực quan hóa, hiển thị cụ thể số lượng dự đoán True Positives, True Negatives, False Positives, và False Negatives. Nó giúp hiểu rõ bản chất các loại lỗi mà mô hình đang mắc phải. (Kết quả được trực quan hóa bởi module evaluate.py).
7. **ROC Curve (Receiver Operating Characteristic Curve) và AUC (Area Under the Curve):**
   * *Nội dung:* ROC Curve là đồ thị biểu diễn sự đánh đổi giữa Tỷ lệ Dương tính Thật (True Positive Rate, chính là Recall) và Tỷ lệ Dương tính Giả (False Positive Rate) tại các ngưỡng phân loại xác suất khác nhau. AUC là diện tích nằm dưới đường cong ROC, giá trị nằm trong khoảng [0, 1].
   * *Ý nghĩa:* AUC đo lường khả năng tổng thể của mô hình trong việc phân biệt giữa hai lớp, không phụ thuộc vào ngưỡng phân loại cụ thể. AUC càng gần 1, mô hình phân biệt càng tốt. AUC = 0.5 thể hiện hiệu năng tương đương dự đoán ngẫu nhiên. (Kết quả được trực quan hóa bởi evaluate.py cho các mô hình hỗ trợ predict\_proba).
8. **Cross-Validation Accuracy (Kết quả từ pha huấn luyện):**
   * *Ý nghĩa:* Giá trị Accuracy trung bình và độ lệch chuẩn (ví dụ: 0.8626 ± 0.0265 cho Logistic Regression) thu được từ kỹ thuật kiểm định chéo (Cross-Validation) 5-fold trên tập huấn luyện. Chỉ số này cung cấp một ước lượng về độ ổn định và khả năng tổng quát hóa dự kiến của mô hình *trước khi* thực hiện đánh giá cuối cùng trên tập kiểm tra. Độ lệch chuẩn thấp cho thấy hiệu năng ổn định qua các tập dữ liệu con khác nhau.

**5.3. Kết quả Đánh giá trên Tập Kiểm tra**

Bảng dưới đây tổng hợp hiệu năng của các mô hình đã huấn luyện khi được đánh giá trên tập kiểm tra (X\_test\_processed, y\_test) gồm 173 mẫu. Các chỉ số chính bao gồm Accuracy, Precision (weighted), Recall (weighted), F1-Score (weighted), thời gian huấn luyện (tính bằng giây trên tập huấn luyện), và kết quả Accuracy trung bình từ Cross-Validation (CV) 5-fold trên tập huấn luyện.

**Bảng 5.1: So sánh hiệu năng các mô hình trên tập kiểm tra**

| Tên Mô hình | Accuracy | Precision | Recall | F1-Score | Thời gian Huấn luyện (s) | CV Accuracy | CV Độ lệch chuẩn |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Logistic Regression | 0.8324 | 0.8346 | 0.8324 | 0.8328 | 0.0034 | 0.8626 | 0.0265 |
| Decision Tree | 0.8439 | 0.8461 | 0.8439 | **0.8443** | 0.0063 | 0.8626 | 0.0192 |
| Random Forest | 0.8439 | 0.8442 | 0.8439 | 0.8440 | 0.3698 | 0.8627 | 0.0223 |
| Gradient Boosting | 0.8439 | 0.8449 | 0.8439 | 0.8442 | 0.5939 | **0.8724** | 0.0266 |
| K-Nearest Neighbors | 0.8092 | 0.8105 | 0.8092 | 0.8076 | 0.0020 | 0.8588 | 0.0350 |
| Naive Bayes | 0.6532 | 0.7085 | 0.6532 | 0.6056 | 0.0023 | 0.6923 | 0.0790 |
| Support Vector Machine | 0.8439 | 0.8449 | 0.8439 | 0.8442 | 0.0824 | 0.8645 | 0.0297 |

*(Lưu ý: F1-Score cao nhất được in đậm. CV Accuracy cao nhất được in đậm.)*

**5.4. Phân tích và So sánh Kết quả**

Từ Bảng 5.1 và các báo cáo Classification Report chi tiết, có thể đưa ra các phân tích và so sánh sau:

1. **Hiệu năng Tổng thể:**
   * Một nhóm gồm bốn thuật toán: **Decision Tree, Random Forest, Gradient Boosting, và Support Vector Machine** thể hiện hiệu năng vượt trội và rất sát nhau trên tập dữ liệu kiểm tra. Các mô hình này đều đạt mức Accuracy và F1-Score xấp xỉ **0.844**.
   * **Logistic Regression** cho kết quả thấp hơn một chút, với F1-Score khoảng 0.833.
   * **K-Nearest Neighbors** có hiệu năng thấp hơn nữa, đạt F1-Score khoảng 0.808.
   * **Naive Bayes** cho thấy hiệu năng **yếu nhất** một cách rõ rệt, với F1-Score chỉ đạt 0.606, cách biệt đáng kể so với các phương pháp còn lại.
2. **Đối chiếu với Kết quả Cross-Validation:**
   * Đối với đa số các mô hình (Logistic Regression, Decision Tree, Random Forest, Gradient Boosting, K-Nearest Neighbors, Support Vector Machine), giá trị Accuracy trên tập kiểm tra thấp hơn một chút so với giá trị Accuracy trung bình thu được từ Cross-Validation trên tập huấn luyện. Ví dụ, Gradient Boosting đạt CV Accuracy cao nhất (0.8724) nhưng trên tập kiểm tra là 0.8439. Sự sụt giảm nhẹ này là hiện tượng phổ biến, cho thấy mô hình có thể đã bị overfitting nhẹ trên tập huấn luyện mà Cross-Validation chưa phản ánh hết, nhưng nhìn chung khả năng tổng quát hóa vẫn ở mức tốt.
   * Sự chênh lệch lớn hơn đối với Naive Bayes (từ 0.6923 xuống 0.6532) càng củng cố kết luận về hiệu năng hạn chế của thuật toán này. Độ lệch chuẩn CV của Naive Bayes cũng cao nhất (0.0790), cho thấy sự thiếu ổn định trong hiệu năng của nó.
3. **Phân tích Chi tiết theo Từng Lớp (Từ Classification Report):**
   * **Các mô hình hiệu năng cao (Decision Tree, Random Forest, Gradient Boosting, Support Vector Machine):** Nhìn chung, các thuật toán này thể hiện sự cân bằng tốt trong việc dự đoán cả hai lớp 0 (từ chối) và lớp 1 (phê duyệt). Chúng đạt được các giá trị Precision và Recall tương đối đồng đều cho cả hai lớp. Ví dụ, Support Vector Machine có precision/recall là 0.81/0.84 cho lớp 0 và 0.87/0.84 cho lớp 1. Điều này cho thấy khả năng nhận diện khá tốt cả hai tình huống.
   * **Logistic Regression:** Có xu hướng đạt Recall cao hơn cho lớp 0 (0.84) và Precision cao hơn cho lớp 1 (0.87), cho thấy nó có thể "bỏ sót" ít trường hợp từ chối đúng hơn, nhưng lại "chắc chắn" hơn khi dự đoán phê duyệt.
   * **K-Nearest Neighbors:** Hiệu năng không đồng đều giữa hai lớp, với Recall thấp cho lớp 0 (0.73) nhưng cao cho lớp 1 (0.88). Điều này ngụ ý mô hình này có xu hướng bỏ sót nhiều trường hợp từ chối đúng, nhưng lại khá tốt trong việc xác định các trường hợp phê duyệt đúng.
   * **Naive Bayes:** Thể hiện sự mất cân bằng nghiêm trọng nhất. **Recall cực thấp cho lớp 0 (0.29)** cho thấy mô hình gần như không thể nhận diện được phần lớn các trường hợp cần từ chối. Ngược lại, Recall rất cao cho lớp 1 (0.95) nhưng đi kèm với Precision thấp (0.62), chỉ ra rằng mô hình này dự đoán "phê duyệt" một cách quá thường xuyên, dẫn đến nhiều dự đoán sai (False Positives). Nguyên nhân có thể nằm ở việc vi phạm mạnh mẽ giả định độc lập hoặc giả định phân phối Gaussian không phù hợp với bản chất lệch của dữ liệu.
4. **Thời gian Huấn luyện:**
   * Logistic Regression, Decision Tree, K-Nearest Neighbors, Naive Bayes có thời gian huấn luyện rất ngắn.
   * Support Vector Machine yêu cầu thời gian nhiều hơn một chút (0.08 giây).
   * Các mô hình ensemble, đặc biệt là **Gradient Boosting (0.59 giây)** và **Random Forest (0.37 giây)**, có thời gian huấn luyện dài nhất do phải xây dựng nhiều mô hình cơ sở. Yếu tố thời gian này cần được xem xét trong các ứng dụng thực tế đòi hỏi cập nhật mô hình thường xuyên.
5. **Đánh giá Sơ bộ Mô hình Tốt nhất:**
   * Dựa trên chỉ số F1-Score (cân bằng giữa Precision và Recall), **Decision Tree** (0.8443) nhỉnh hơn một chút so với Gradient Boosting và Support Vector Machine (cùng 0.8442), và Random Forest (0.8440). Tuy nhiên, sự khác biệt về hiệu năng giữa bốn mô hình này là không đáng kể về mặt thống kê dựa trên một lần chạy duy nhất.
   * Gradient Boosting thể hiện tiềm năng ổn định cao nhất dựa trên kết quả Cross-Validation Accuracy, nhưng lại tốn nhiều thời gian huấn luyện nhất.
   * Decision Tree và Support Vector Machine cung cấp sự cân bằng tốt giữa hiệu năng cao và thời gian huấn luyện hợp lý.
   * Random Forest cũng cho hiệu năng tương đương nhưng với chi phí thời gian huấn luyện cao hơn Decision Tree và Support Vector Machine.

**5.5. Kết luận Chương**

Quá trình đánh giá hiệu năng trên tập dữ liệu kiểm tra độc lập đã cung cấp những cơ sở định lượng vững chắc để so sánh khả năng tổng quát hóa của các thuật toán học máy đã được huấn luyện. Kết quả thực nghiệm cho thấy một nhóm gồm Decision Tree, Random Forest, Gradient Boosting và Support Vector Machine đạt được hiệu suất vượt trội và rất tương đồng, thể hiện qua các chỉ số Accuracy và F1-Score xấp xỉ 0.844. Ngược lại, Logistic Regression cho thấy hiệu năng thấp hơn một chút, trong khi K-Nearest Neighbors và đặc biệt là Naive Bayes tỏ ra kém hiệu quả hơn đáng kể trên bộ dữ liệu cụ thể này.

Phân tích chi tiết hiệu năng trên từng lớp cho thấy các mô hình hàng đầu có khả năng duy trì sự cân bằng tốt giữa việc nhận diện các trường hợp phê duyệt và từ chối tín dụng. Với sự khác biệt rất nhỏ về các độ đo chính giữa bốn mô hình tốt nhất (Decision Tree, Random Forest, Gradient Boosting, Support Vector Machine), việc lựa chọn mô hình tối ưu cuối cùng có thể cần cân nhắc thêm các yếu tố phi hiệu năng như:

* **Thời gian huấn luyện:** Decision Tree và Support Vector Machine nhanh hơn đáng kể so với Random Forest và Gradient Boosting.
* **Khả năng diễn giải:** Decision Tree là mô hình dễ diễn giải nhất ("white box"), trong khi các mô hình còn lại (đặc biệt là các mô hình ensemble và Support Vector Machine với kernel phi tuyến) được xem là các mô hình "hộp đen" ("black box"), khó giải thích lý do đằng sau các dự đoán cụ thể. Yêu cầu về tính minh bạch trong ngành tài chính có thể ưu tiên Decision Tree hoặc Logistic Regression (dù hiệu năng thấp hơn một chút).

Mặc dù Decision Tree có chỉ số F1-Score cao nhất trong lần đánh giá này, sự tương đồng về hiệu năng với Random Forest, Gradient Boosting và Support Vector Machine cho thấy tất cả chúng đều là những lựa chọn tiềm năng. Để đưa ra quyết định cuối cùng, các bước tiếp theo có thể bao gồm việc thực hiện tinh chỉnh siêu tham số (Hyperparameter Tuning) cho nhóm mô hình hàng đầu này nhằm khai thác tối đa tiềm năng của chúng và đánh giá lại hiệu năng trên một quy trình nghiêm ngặt hơn (ví dụ: lặp lại quá trình chia train-test hoặc sử dụng nested cross-validation).

**Chương 6: Hiện tượng Overfitting và Các Kỹ thuật Giảm thiểu**

**6.1. Giới thiệu**

Một trong những thách thức trung tâm trong học máy có giám sát là xây dựng các mô hình không chỉ hoạt động tốt trên dữ liệu đã thấy (tập huấn luyện) mà còn có khả năng **tổng quát hóa (generalize)** tốt trên dữ liệu mới, chưa từng gặp (tập kiểm tra hoặc dữ liệu thực tế). Hiện tượng **Overfitting (Quá khớp)** xảy ra khi mô hình học quá chi tiết các đặc điểm, bao gồm cả nhiễu (noise) và các biến động ngẫu nhiên, trong tập dữ liệu huấn luyện. Kết quả là mô hình đạt hiệu năng rất cao trên tập huấn luyện nhưng lại hoạt động kém trên dữ liệu mới. Việc hiểu rõ, nhận biết và áp dụng các kỹ thuật để giảm thiểu Overfitting là cực kỳ quan trọng để xây dựng các mô hình học máy đáng tin cậy và hữu ích, đặc biệt trong các lĩnh vực đòi hỏi độ chính xác cao như phê duyệt tín dụng. Chương này sẽ đi sâu vào hiện tượng Overfitting, các nguyên nhân, hậu quả, các phương pháp giải quyết phổ biến, phân tích cách các phương pháp này đã được tích hợp trong quá trình huấn luyện mô hình (với bộ tham số models), và so sánh hiệu quả thực nghiệm với một bộ tham số được thiết kế để dễ Overfitting hơn (over\_models) dựa trên kết quả thu được.

**6.2. Hiện tượng Overfitting**

* **Định nghĩa:** Overfitting xảy ra khi một mô hình học máy trở nên quá phức tạp so với bản chất của dữ liệu hoặc khi nó "học thuộc lòng" các điểm dữ liệu huấn luyện thay vì nắm bắt xu hướng tổng quát tiềm ẩn. Mô hình này khớp một cách quá chặt chẽ với dữ liệu huấn luyện, bao gồm cả các yếu tố nhiễu hoặc các mẫu hình chỉ xuất hiện ngẫu nhiên trong tập huấn luyện đó.
* **Nguyên nhân:** Mô hình quá phức tạp; Dữ liệu huấn luyện hạn chế; Nhiễu trong dữ liệu; Thời gian huấn luyện quá dài.
* **Hậu quả:** Khả năng tổng quát hóa kém (poor generalization) trên dữ liệu mới.
* **Sự đánh đổi Giữa Độ chệch và Phương sai (Bias-Variance Trade-off):** Overfitting thường tương ứng với phương sai cao (high variance). Các kỹ thuật chống Overfitting thường nhằm mục đích giảm phương sai.

**6.3. Các Phương pháp Giải quyết Overfitting**

Có nhiều kỹ thuật đã được phát triển để giảm thiểu nguy cơ Overfitting:

1. **Điều chuẩn (Regularization):**
   * *Nguyên tắc:* Thêm một thành phần "phạt" (penalty term) vào hàm mất mát của mô hình, nhằm hạn chế độ phức tạp của mô hình. Thành phần phạt này thường dựa trên độ lớn của các tham số (trọng số) của mô hình.
   * *Các dạng phổ biến:*
     + **L2 Regularization (Ridge):** Thêm bình phương chuẩn L2 của vector trọng số vào hàm mất mát (λ∣∣w∣∣22\lambda ||\mathbf{w}||\_2^2). Nó khuyến khích các trọng số nhỏ hơn, làm cho mô hình "mượt" hơn và ít nhạy cảm hơn với các đặc trưng đầu vào riêng lẻ.
     + **L1 Regularization (Lasso):** Thêm chuẩn L1 (giá trị tuyệt đối) của vector trọng số vào hàm mất mát (λ∣∣w∣∣1\lambda ||\mathbf{w}||\_1). L1 có xu hướng đẩy một số trọng số về chính xác bằng 0, có tác dụng như một cơ chế **lựa chọn đặc trưng (feature selection)** tự động.
   * *Ứng dụng:* Thường được sử dụng trong các mô hình tuyến tính (Logistic Regression, Linear Regression) và Support Vector Machine. Tham số C trong Scikit-learn Logistic Regression và SVM thường là *nghịch đảo* của hệ số điều chuẩn λ\lambda (tức là C nhỏ tương ứng điều chuẩn mạnh).
2. **Kiểm soát Độ phức tạp Cây (Tree Complexity Control / Pruning):**
   * *Nguyên tắc:* Áp dụng riêng cho các mô hình dựa trên cây (Decision Tree, Random Forest, Gradient Boosting) để ngăn chúng phát triển quá sâu và phức tạp.
   * *Các kỹ thuật:*
     + **Giới hạn Độ sâu Tối đa (max\_depth):** Ngăn cây phát triển vượt quá một độ sâu nhất định.
     + **Số mẫu Tối thiểu để Chia (min\_samples\_split):** Yêu cầu một nút phải có ít nhất số lượng mẫu này mới được phép chia tiếp.
     + **Số mẫu Tối thiểu ở Lá (min\_samples\_leaf):** Đảm bảo mỗi nút lá (quyết định cuối cùng) được hỗ trợ bởi ít nhất số lượng mẫu này.
     + **Cắt tỉa Sau Huấn luyện (Post-Pruning):** Xây dựng cây đầy đủ trước, sau đó loại bỏ các nhánh không mang lại cải thiện đáng kể trên một tập dữ liệu kiểm định (validation set).
3. **Phương pháp Tập hợp (Ensemble Methods):**
   * *Nguyên tắc:* Kết hợp dự đoán từ nhiều mô hình yếu hơn (base learners) để tạo ra một mô hình tổng hợp mạnh mẽ và ổn định hơn.
   * *Các kỹ thuật chính:*
     + **Bagging (Bootstrap Aggregating):** Huấn luyện nhiều mô hình cơ sở (ví dụ: Decision Trees) trên các tập dữ liệu bootstrap khác nhau (lấy mẫu có hoàn lại) và kết hợp kết quả (ví dụ: majority voting). **Random Forest** là một ví dụ điển hình, nó còn thêm yếu tố ngẫu nhiên khi chọn đặc trưng. Bagging chủ yếu giúp **giảm phương sai**.
     + **Boosting:** Huấn luyện các mô hình cơ sở một cách tuần tự. Mỗi mô hình mới tập trung vào việc sửa lỗi của các mô hình trước đó. **Gradient Boosting** là một ví dụ mạnh mẽ. Boosting thường giúp **giảm độ chệch** nhưng cũng cần các cơ chế (như learning rate, subsampling) để kiểm soát overfitting.
4. **Kiểm định Chéo (Cross-Validation):**
   * *Nguyên tắc:* Phân chia dữ liệu huấn luyện thành nhiều phần (folds). Lần lượt sử dụng một phần làm tập kiểm định (validation) và các phần còn lại để huấn luyện. Lặp lại quá trình này và lấy trung bình kết quả đánh giá.
   * *Ứng dụng chống Overfitting:* Mặc dù chủ yếu dùng để đánh giá mô hình một cách đáng tin cậy hơn, Cross-Validation rất quan trọng trong việc **phát hiện Overfitting** (nếu hiệu năng trên các fold rất khác nhau hoặc thấp hơn nhiều so với training) và đặc biệt là trong **tinh chỉnh siêu tham số (Hyperparameter Tuning)** để chọn ra các giá trị siêu tham số giúp mô hình tổng quát hóa tốt nhất.
5. **Dừng Sớm (Early Stopping):**
   * *Nguyên tắc:* Áp dụng cho các thuật toán huấn luyện lặp đi lặp lại (ví dụ: Gradient Descent, huấn luyện mạng nơ-ron, Gradient Boosting). Theo dõi hiệu năng của mô hình trên một tập kiểm định (validation set) riêng biệt sau mỗi vòng lặp (hoặc một số vòng lặp). Dừng quá trình huấn luyện khi hiệu năng trên tập kiểm định bắt đầu giảm đi (ngay cả khi hiệu năng trên tập huấn luyện vẫn đang cải thiện).
   * *Ứng dụng:* Ngăn chặn mô hình huấn luyện quá lâu và bắt đầu khớp với nhiễu. Việc giới hạn n\_estimators trong Random Forest/Gradient Boosting hoặc max\_iter trong Logistic Regression/Support Vector Machine có thể xem là một dạng dừng sớm đơn giản.
6. **Thu thập Thêm Dữ liệu:** Cách hiệu quả nhất để chống Overfitting thường là cung cấp thêm dữ liệu huấn luyện đa dạng và đại diện cho mô hình học. Dữ liệu nhiều hơn giúp mô hình khó "ghi nhớ" hơn và buộc phải học các quy luật tổng quát.
7. **Lựa chọn/Thiết kế Đặc trưng (Feature Selection/Engineering):** Loại bỏ các đặc trưng không liên quan hoặc tạo ra các đặc trưng mới có ý nghĩa hơn có thể giúp mô hình tập trung vào các tín hiệu quan trọng và giảm khả năng khớp với nhiễu.

**6.4. Áp dụng Các Phương pháp Tránh Overfitting trong Bộ Tham số models (Thiết lập Kiểm soát)**

Bộ tham số models được định nghĩa trong module train\_models.py (xem Phụ lục hoặc Chương 4) đã tích hợp các biện pháp kiểm soát Overfitting một cách có chủ đích:

* **Logistic Regression:** Sử dụng điều chuẩn L2 tương đối mạnh (C=0.05).
* **Decision Tree:** Giới hạn độ sâu (max\_depth=6), yêu cầu số mẫu tối thiểu đáng kể để chia và tạo lá (min\_samples\_split=30, min\_samples\_leaf=15).
* **Random Forest:** Tận dụng sức mạnh của Bagging (n\_estimators=200, bootstrap=True) và giới hạn độ phức tạp của cây cơ sở (max\_depth=15, min\_samples\_split=15, min\_samples\_leaf=5).
* **Gradient Boosting:** Sử dụng learning rate nhỏ (learning\_rate=0.1), giới hạn độ phức tạp cây cơ sở (max\_depth=8, min\_samples\_split=15, min\_samples\_leaf=5), và áp dụng Stochastic Gradient Boosting (subsample=0.8).
* **K-Nearest Neighbors:** Chọn n\_neighbors=7 (thay vì K=1).
* **Support Vector Machine:** Sử dụng điều chuẩn tương đối mạnh (C=0.1) và giới hạn vòng lặp (max\_iter=1000).
* **Chung:** Sử dụng class\_weight='balanced' giúp mô hình học tốt hơn trên toàn bộ phân phối dữ liệu, cải thiện khả năng tổng quát hóa khi dữ liệu mất cân bằng.

**6.5. So sánh Hiệu quả Thực nghiệm: Thiết lập Kiểm soát (models) và Thiết lập Overfit-prone (over\_models)**

Để đánh giá tác động thực tế của các biện pháp kiểm soát Overfitting, chúng ta tiến hành so sánh hiệu năng giữa hai bộ cấu hình siêu tham số trên cùng tập dữ liệu kiểm tra. Bộ over\_models được thiết kế cố ý để giảm bớt hoặc loại bỏ các biện pháp kiểm soát này (chi tiết cấu hình over\_models xem ở Phụ lục hoặc Chương 4, ví dụ: C=10000, max\_depth=None, min\_samples\_split=2, min\_samples\_leaf=1, n\_neighbors=1, learning\_rate=0.5, subsample=1.0). Bảng 6.1 và 6.2 trình bày kết quả so sánh.

**Bảng 6.1: So sánh Hiệu năng trên Tập Kiểm tra (Test Set Performance)**

| Tên Mô hình | F1-Score (models - Kiểm soát) | F1-Score (over\_models - Overfit-prone) | Thay đổi Hiệu năng |
| --- | --- | --- | --- |
| Logistic Regression | 0.8328 | **0.8386** | +0.0058 |
| Decision Tree | **0.8443** | 0.7810 | -0.0633 |
| Random Forest | 0.8440 | **0.8497** | +0.0057 |
| Gradient Boosting | 0.8442 | 0.8442 | 0.0000 |
| K-Nearest Neighbors | 0.8076 | **0.8141** | +0.0065 |
| Naive Bayes | 0.6056 | 0.6056 | 0.0000 |
| Support Vector Machine | **0.8442** | 0.7926 | -0.0516 |

**Bảng 6.2: So sánh Độ ổn định (CV Accuracy trên Tập Huấn luyện)**

| Tên Mô hình | CV Accuracy (models - Kiểm soát) | CV Accuracy (over\_models - Overfit-prone) | Thay đổi Độ ổn định/Tổng quát hóa Dự kiến |
| --- | --- | --- | --- |
| Logistic Regression | 0.8626 ± 0.0265 | 0.8355 ± 0.0303 | Giảm (-) |
| Decision Tree | 0.8626 ± 0.0192 | 0.8027 ± 0.0157 | Giảm mạnh (--) |
| Random Forest | 0.8627 ± 0.0223 | **0.8762 ± 0.0254** | **Cải thiện (+)** |
| Gradient Boosting | **0.8724 ± 0.0266** | 0.8008 ± 0.0127 | Giảm mạnh (--) |
| K-Nearest Neighbors | 0.8588 ± 0.0350 | 0.8143 ± 0.0165 | Giảm (-) |
| Naive Bayes | 0.6923 ± 0.0790 | 0.6923 ± 0.0790 | Không đổi |
| Support Vector Machine | 0.8645 ± 0.0297 | 0.7641 ± 0.0347 | Giảm mạnh (--) |

**Phân tích Kết quả So sánh:**

1. **Trường hợp Overfitting rõ ràng:**
   * **Decision Tree và Support Vector Machine:** Kết quả thực nghiệm hoàn toàn phù hợp với dự đoán lý thuyết. Việc loại bỏ các giới hạn (over\_models) đã làm **giảm mạnh** cả hiệu năng trên tập kiểm tra (F1-Score giảm ~0.06 và ~0.05 tương ứng) và độ ổn định/khả năng tổng quát hóa ước tính (CV Accuracy giảm mạnh). Điều này chứng tỏ các mô hình này đã học thuộc lòng nhiễu trong tập huấn luyện và mất khả năng dự đoán chính xác trên dữ liệu mới.
   * **Gradient Boosting:** Mặc dù F1-Score trên tập kiểm tra không đổi trong lần chạy này, **CV Accuracy giảm nghiêm trọng** (từ 0.8724 xuống 0.8008). Đây là dấu hiệu mạnh mẽ của Overfitting – mô hình trở nên kém ổn định và hiệu năng tổng quát hóa dự kiến thấp hơn nhiều, ngay cả khi kết quả trên một tập kiểm tra cụ thể có thể chưa phản ánh hết. Cấu hình over\_models rõ ràng đã làm mô hình này kém hiệu quả hơn trong việc tổng quát hóa.
2. **Trường hợp Hiệu năng Kiểm tra Tăng nhẹ nhưng Độ ổn định Giảm:**
   * **Logistic Regression và K-Nearest Neighbors:** Cả hai thuật toán này đều cho thấy F1-Score trên tập kiểm tra **tăng nhẹ** khi sử dụng cấu hình over\_models (giảm điều chuẩn cho LR, K=1 cho KNN). Tuy nhiên, đi kèm với đó là sự **suy giảm đáng kể trong CV Accuracy**. Điều này cho thấy các mô hình over\_models này **kém ổn định hơn** và có thể chỉ hoạt động tốt hơn trên tập kiểm tra cụ thể này do yếu tố may rủi hoặc do cấu hình models ban đầu hơi bị Underfitting. Nhìn chung, cấu hình models vẫn được ưu tiên hơn do độ ổn định cao hơn.
3. **Trường hợp đặc biệt - Random Forest:**
   * Điều thú vị là với Random Forest, việc cho phép các cây cơ sở phức tạp hơn (over\_models) lại dẫn đến **cả F1-Score trên tập kiểm tra và CV Accuracy đều tăng nhẹ**. Điều này làm nổi bật **khả năng chống Overfitting mạnh mẽ vốn có của Random Forest** nhờ vào cơ chế Bagging và chọn đặc trưng ngẫu nhiên. Có thể trong trường hợp này, các cây phức tạp hơn đã giúp nắm bắt tốt hơn các tương tác trong dữ liệu, và cơ chế ensemble vẫn đủ mạnh để ngăn chặn Overfitting đáng kể. Tuy nhiên, cần lưu ý rằng cây phức tạp hơn sẽ tốn nhiều tài nguyên hơn.
4. **Naive Bayes:** Không có thay đổi, phù hợp với kỳ vọng.

**Minh chứng cho Tầm quan trọng của Kiểm soát Overfitting:**

Kết quả thực nghiệm đã cung cấp bằng chứng mạnh mẽ. Ngoại trừ trường hợp đặc biệt của Random Forest (cho thấy sự mạnh mẽ của chính thuật toán), việc nới lỏng hoặc loại bỏ các biện pháp kiểm soát Overfitting trong over\_models nhìn chung đã dẫn đến:

* **Giảm hiệu năng trên tập kiểm tra** (Decision Tree, SVM).
* **Giảm độ ổn định và khả năng tổng quát hóa dự kiến** (thể hiện qua CV Accuracy giảm đối với LR, DT, GB, KNN, SVM).

Điều này khẳng định rằng các kỹ thuật chống Overfitting được áp dụng trong bộ models là **thiết yếu và hiệu quả**. Chúng giúp tạo ra các mô hình không chỉ khớp với dữ liệu huấn luyện mà quan trọng hơn là có thể duy trì hiệu năng tốt khi đối mặt với dữ liệu mới, chưa từng thấy. Sự cân bằng được tìm thấy trong models đã dẫn đến hiệu năng tổng quát hóa tốt hơn so với việc cố gắng tối đa hóa hiệu năng trên tập huấn luyện một cách thiếu kiểm soát.

**6.6. Kết luận Chương**

Overfitting là một hiện tượng phổ biến và là mối quan tâm hàng đầu trong xây dựng mô hình học máy, đặc trưng bởi hiệu năng cao trên dữ liệu huấn luyện nhưng kém trên dữ liệu mới. Chương này đã thảo luận về bản chất của Overfitting, các nguyên nhân, và các phương pháp cốt lõi để giải quyết như Điều chuẩn (Regularization), kiểm soát độ phức tạp mô hình, phương pháp tập hợp (Ensemble Methods), và các kỹ thuật khác.

Quá trình huấn luyện mô hình trong dự án đã tích hợp các biện pháp kiểm soát Overfitting vào cấu hình siêu tham số của bộ models. So sánh thực nghiệm hiệu năng trên tập kiểm tra và kết quả kiểm định chéo giữa bộ models (kiểm soát chặt chẽ hơn) và bộ over\_models (ít kiểm soát/overfit-prone) đã cung cấp bằng chứng định lượng rõ ràng. Ngoại trừ Random Forest cho thấy khả năng chống chịu tốt, việc nới lỏng kiểm soát nhìn chung đã dẫn đến suy giảm đáng kể khả năng tổng quát hóa (thể hiện qua hiệu năng kiểm tra hoặc độ ổn định CV) đối với phần lớn các thuật toán được khảo sát.

Kết quả này khẳng định mạnh mẽ tầm quan trọng của việc áp dụng các kỹ thuật tránh Overfitting. Việc cân bằng giữa độ phức tạp mô hình và khả năng tổng quát hóa, như đã thực hiện trong bộ models, là chìa khóa để xây dựng các mô hình phân loại phê duyệt tín dụng đáng tin cậy và có giá trị thực tiễn cao.