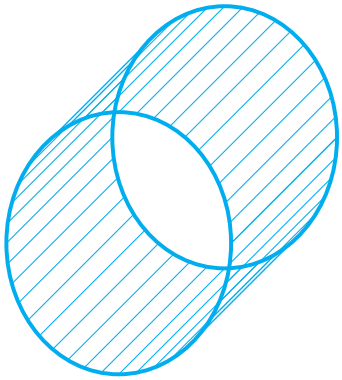
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Khoa Toán - Tin học

o0o



Khoa Toán - Tin học

Fac. of Math. & Computer Science

BÁO CÁO CUỐI KỲ

Môn học	:	Nhận dạng mẫu	
Giảng viên phụ trách	:	TS.Ngô Minh Mẫn	
Nhóm sinh viên thực hiện	:	Trần Huỳnh Nghĩa	– 20110251
		Đỗ Tấn Phát	– 20110270
		Lại Trọng Đức	– 20110150
		Huỳnh Thư Trúc	– 1711290

TP.Hồ Chí Minh - 2022

Tóm tắt nội dung

Được sự phân công của thầy, nhóm em đã thực hiện đề tài 1. Nội dung đề tài gồm 2 yêu cầu như sau:

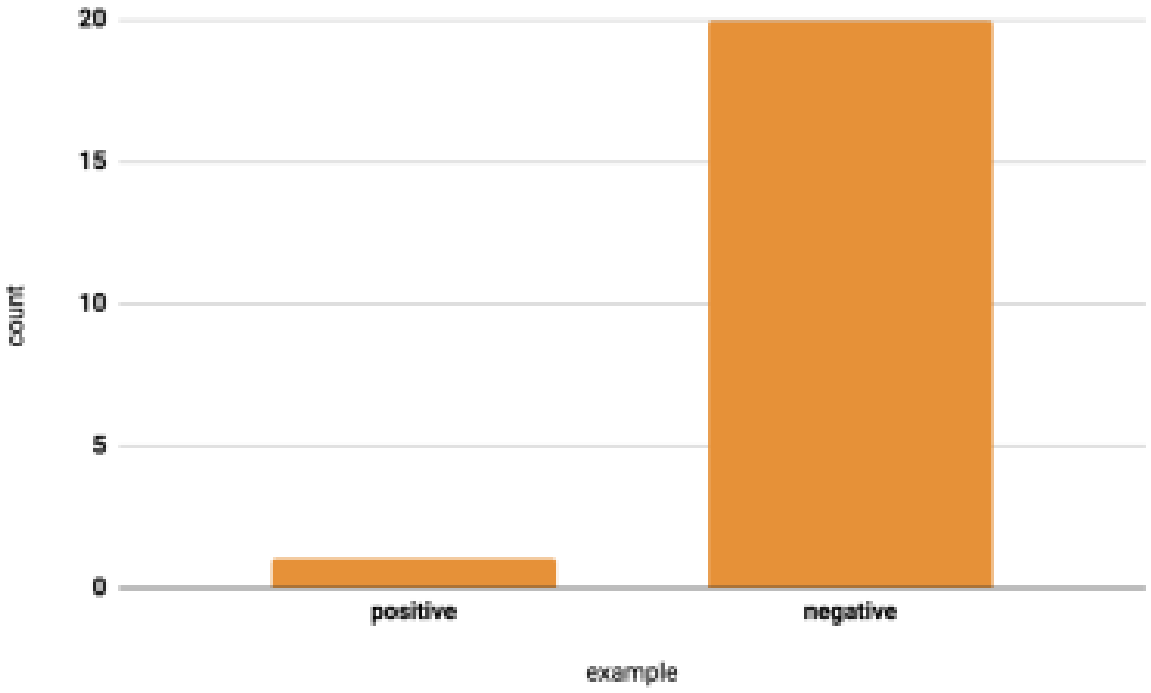
1. Cách xử lý imbalanced data trong bài toán classification khi dùng Bagging.
2. Sử dụng ràng buộc Monotonic (partially) cho Boosting.

Mục lục

1 Cách xử lý imbalanced data trong bài toán classification khi dùng Bagging.

1.1 Imbalanced data là gì?

- Imbalanced data là một trong những hiện tượng phổ biến của bài toán phân loại nhị phân (binary classification) như spam email.
- Ví dụ, dự đoán một email là spam hay không, nếu 99% email không phải là spam (negative class) và chỉ có 1% là spam (positive class) thì tập dữ liệu được coi là imbalanced.
- Việc mất cân bằng dữ liệu nghiêm trọng thường dẫn tới ngộ nhận chất lượng mô hình (thước đo đánh giá mô hình là accuracy có thể đạt được rất cao mà không cần tới mô hình).
- Ngoài ra, mất cân bằng dữ liệu nghiêm trọng thường dẫn tới dự báo kém chính xác trên nhóm thiểu số.



Hình 1: Imbalanced data

1.2 Các phương pháp giải quyết dữ liệu mất cân bằng

1.2.1 Thay đổi metric

Như đã giải thích ở mục đầu tiên, khi hiện tượng mất cân bằng dữ liệu nghiêm trọng xảy ra thì việc sử dụng độ chính xác làm thước đo đánh giá mô hình thường không hiệu quả bởi hầu hết chúng đều đạt độ chính xác rất cao. Một mô hình ngẫu nhiên dự báo toàn bộ là nhãn thuộc nhóm đa số cũng sẽ mang lại kết quả gần bằng `100%`. Khi đó ta có thể cân nhắc tới một số metrics thay thế như `precision, recall, f1-score, gini,...`. Các chỉ số này sẽ không quá lớn để dẫn tới ngộ nhận độ chính xác, đồng thời chúng tập trung hơn vào việc đánh giá độ chính xác trên nhóm thiểu số, nhóm mà chúng ta muốn dự báo chính xác hơn so với nhóm đa số.

		Actual		
		Positive	**Negative**	
Predicted	**Positive**	True Positive (TP)	False Positive (FP) Type I error	Precision = TP/(TP+FP)
	Negative	False Negative (FN) Type II error	True Negative (TN)	
		Recall = TP/(TP+FN)	False positive rate (FPR) = FP/(FP+TN)	

Hình 2: Bảng cross table mô tả kết quả thống kê chéo giữa nhãn dự báo và ground truth. Ở đây Positive tương ứng với nhãn 1 (Spam) và Negative tương ứng với nhãn 0 (not Spam).

Từ bảng cross table ta dễ dàng hình dung được ý nghĩa của các chỉ số đó là:

- Precision: Mức độ dự báo chính xác trong những trường hợp được dự báo là Positive.

$$\text{Precision} = \frac{TP}{TP + FP}$$

- Recall: Mức độ dự báo chuẩn xác những trường hợp là Positive trong những trường hợp thực tế là Positive.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- F1-Score: Trung bình điều hòa giữa Precision và Recall. Đây là chỉ số thay thế lý tưởng cho accuracy khi mô hình có tỷ lệ mất cân bằng mẫu cao.

$$F_1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}$$

- Kappa-Score: Là chỉ số đo lường mức độ liên kết tin cậy (inter-rater reliability) cho các categories.
- ini: Đo lường sự bất bình đẳng trong phân phối giữa Positive và Negative được dự báo từ mô hình.
- AUC: Biểu diễn mối quan hệ giữa độ nhạy (sensitivity) và độ đặc hiệu (specificity). Đánh giá khả năng phân loại good và bad được dự báo từ mô hình.

1.2.2 Under sampling

Under sampling là việc ta giảm số lượng các quan sát của nhóm đa số để nó trở nên cân bằng với số quan sát của nhóm thiểu số. Ưu điểm của under sampling là làm cân bằng mẫu một cách nhanh chóng, dễ dàng tiến hành thực hiện mà không cần đến thuật toán giả lập mẫu.

Tuy nhiên nhược điểm của nó là kích thước mẫu sẽ bị giảm đáng kể. Gỉa sử nhóm thiểu số có kích thước là 500, như vậy để tạo ra sự cân bằng mẫu giữa nhóm đa số và thiểu số sẽ cần giảm kích thước mẫu của nhóm đa số từ 10000 về 500. Tổng kích thước tập huấn luyện sau under sampling là 1000 và chiếm gần 1/10 kích thước tập huấn luyện ban đầu. Tập huấn luyện mới khá nhỏ, không đại diện cho phân phối của toàn bộ tập dữ liệu và thường dễ dẫn tới hiện tượng overfitting.

1.2.3 Over sampling

Over sampling là các phương pháp giúp giải quyết hiện tượng mất cân bằng mẫu bằng cách gia tăng kích thước mẫu thuộc nhóm thiểu số bằng các kĩ thuật khác nhau. Có 2 phương pháp chính để thực hiện over sampling đó là:

- Lựa chọn mẫu có tái lập.
- Mô phỏng mẫu mới dựa trên tổng hợp của các mẫu cũ.

1.2.4 SMOTE (Synthetic Minority Over-sampling)

SMOTE là một kỹ thuật sinh mẫu trong đó các mẫu tổng hợp được tạo ra cho lớp thiểu số. Thuật toán này giúp khắc phục vấn đề overfitting do lấy mẫu quá mức ngẫu nhiên. Nó tập trung vào không gian đặc trưng để tạo ra các trường hợp mới với sự trợ giúp của phép nội suy giữa các trường hợp tích cực nằm với nhau.

SMOTE hoạt động bằng cách chọn ngẫu nhiên một điểm từ lớp thiểu số và tính k-hàng xóm gần nhất cho điểm đó.

Các điểm tổng hợp là các điểm được chèn vào giữa điểm được chỉ định và các điểm lân cận của nó. Thuật toán SMOTE được thực hiện theo các bước sau:

1. Chọn một vectơ đầu vào từ lớp thiểu số.
2. Tìm `k` lân cận gần nhất của nó (`k` hàng xóm là đầu vào của phương thức `SMOTE()`).
3. Chọn một trong các điểm lân cận này và chèn một điểm tổng hợp vào đâu đó trên đường nối điểm đang xem xét và điểm lân cận đã chọn của nó.
4. Lặp lại quá trình cho đến khi dữ liệu được cân bằng.

1.3 Bagging là gì?

- Bagging (Bootstrap Aggregation)là một phương pháp ensemble learning cho phép tạo ra nhiều mô hình dự báo khác nhau từ tập dữ liệu khác nhau và sau đó trung bình hoặc gộp chúng để có một dự báo cuối cùng.
- Bagging sử dụng cơ chế Bootstrap để tạo ra các tập dữ liệu mẫu khác nhau từ tập dữ liệu ban đầu và sử dụng chúng để huấn luyện nhiều mô hình dự báo. Mỗi mô hình được huấn luyện trên một tập dữ liệu mẫu khác nhau và sau đó được gộp lại để có một dự báo cuối cùng.
- Bagging được sử dụng để giảm variance của mô hình và tăng độ chính xác của dự báo.

Lược đồ thuật toán Bagging Bagging($\mathcal{A}$, D, B): B là số lần bootstrap

1. Lấy mẫu có hoàn lại từ D để tạo ra bộ dữ liệu mới $D_1, D_2, \ldots, D_B$ (có cùng kích thước).
2. Áp dụng thuật toán$\mathcal{A}$ để huấn luyện B mô hình
$$h_i(x) = \mathcal{A}(D_i)$$
3. Kết hợp
   - Phân lớp: $h(x) = \arg\max_y \sum_{i=1}^{B} \mathbb{I}(h_i(x = y))$
   - Hồi quy: $h(x) = \frac{1}{B} \sum_{i=1}^{B} h_i(x)$

1.4 Bài toán áp dụng

Bài toán phát hiện gian lận thẻ tín dụng (Credit Card Fraud Detection). Dữ liệu được lấy từ trang Kaggle: https://www.kaggle.com/mlg-ulb/creditcardfraud.

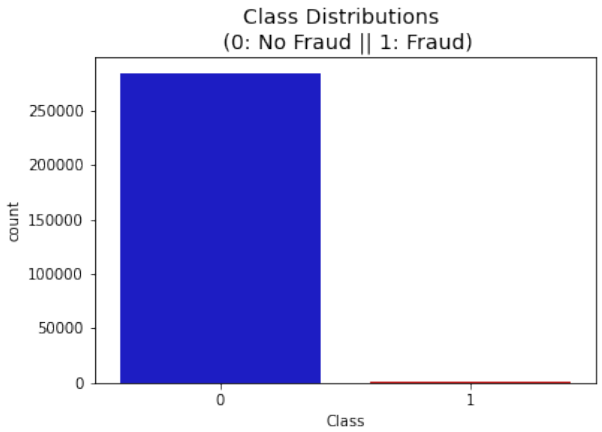
1.4.1 Thu thập dữ liệu

Bộ dữ liệu được lấy từ trang Kaggle. Tổng số giao dịch có trong tập dữ liệu là 284807. Ngoại trừ "Time" và "Amount", tất cả các tính năng được chuyển đổi bằng PCA vì lý do bảo mật.

1.4.2 Tiền xử lý dữ liệu

Tiền xử lý dữ liệu là một bước không thể thiếu trong học máy vì chất lượng dữ liệu và thông tin thu được từ dữ liệu ảnh hưởng trực tiếp đến khả năng học hỏi của mô hình học máy; do đó, việc xử lý trước dữ liệu trước khi đào tạo một mô hình học máy là khá quan trọng.
Tiền xử lý dữ liệu bao gồm các bước sau:

- Xóa các cột không cần thiết
- Handling Missing Data: Thiếu dữ liệu có thể gây ra lỗi trong mô hình Machine Learning. Không có dữ liệu nào bị thiếu trong tập dữ liệu này.
- Data split: Tập dữ liệu được chia thành 2 phần là tập huấn luyện và tập kiểm tra. 80% dữ liệu được sử dụng để huấn luyện và 20% còn lại để kiểm tra.
- Feature scaling: Để đảm bảo tính đồng nhất của các tính năng, chúng ta cần chuẩn hóa các tính năng. Để chuẩn hóa các tính năng, chúng ta sử dụng StandardScaler hoặc RobustScaler. Trong bài này, chúng tôi sử dụng RobustScaler.
- Imbalanced data: Dữ liệu không cân bằng là một vấn đề phổ biến trong các bài toán phân loại. Trong bài toán này, chúng tôi có 492 giao dịch gian lận và 284315 giao dịch bình thường. Điều này có nghĩa là dữ liệu không cân bằng. Để giải quyết vấn đề này.



Hình 3: Imbalanced data.

Hình 4: Balanced data.

1.4.3 Mô hình hóa

Ta chạy mô hình với Decision Tree Classifier và Bagging Decision Tree Classifier.

- Decision Tree Classifier

```
dctree_model = DecisionTreeClassifier(random_state=0)
dctree_model.fit(X_train, y_train)
pred= dctree_model.predict(X_test)
print ("ROC AUC Score=",roc_auc_score(y_test, pred))

print ("Classification Report:")
print (classification_report(y_test, pred))
```

```
ROC AUC Score= 0.9212377832393509
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.90      0.95     56864
           1       0.02      0.94      0.03        98

    accuracy                           0.90     56962
   macro avg       0.51      0.92      0.49     56962
weighted avg       1.00      0.90      0.95     56962
```

Hình 5: Decision Tree Classifier.

- Bagging Decision Tree Classifier

Bagging Classifier using Decision Tree

```
[ ]  bagging_clf = BaggingClassifier(DecisionTreeClassifier(random_state = 0),
                                     n_estimators = 1000, bootstrap = True,
                                     max_samples = 0.85, n_jobs = -1, oob_score = True)

     bagging_clf.fit(X_train_rus, y_train_rus )
     pred = bagging_clf.predict(X_test)
     print ("ROC AUC Score=",roc_auc_score(y_test, pred))
     print ("Classification Report:")
     print (classification_report(y_test, pred))
```

```
ROC AUC Score= 0.9504390353496492
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     56864
           1       0.04      0.94      0.08        98

    accuracy                           0.96     56962
   macro avg       0.52      0.95      0.53     56962
weighted avg       1.00      0.96      0.98     56962
```

Hình 6: Bagging Decision Tree Classifier.

1.4.4 Kết luận

Mô hình Bagging Decision Tree Classifier cho độ chính xác cao hơn với 96% so với 90% của Decision Tree Classifier . Điều này có thể được giải thích bằng cách nói rằng mô hình này sử dụng nhiều cây quyết định để đưa ra dự đoán, có nghĩa là nó sẽ có nhiều quyết định và sẽ có nhiều lựa chọn để đưa ra dự đoán giúp mô hình tránh các lỗi do các cây quyết định đơn lẻ.

2 Sử dụng ràng buộc Monotonic(partially) cho Boosting

Áp dụng ràng buộc Monotonic (partially) trong thuật toán Boosting trong bài toán dự đoán doanh thu có thể giúp cho việc dự báo trở nên chính xác hơn và dễ hiểu hơn. Vì trong nhiều trường hợp, quan hệ giữa các biến đầu vào (ví dụ: số lượng khách hàng, chi phí quảng cáo) và doanh thu là tăng dần hoặc giảm dần, áp dụng ràng buộc Monotonic sẽ giúp cho mô hình dự báo chỉ tăng hoặc giảm tương ứng với sự thay đổi của các biến đầu vào.

Điều này có thể giúp cho việc dự báo trở nên chính xác hơn vì mô hình sẽ không gặp phải các vấn đề như dự báo giảm khi các biến đầu vào tăng hoặc ngược lại.

Tuy nhiên, chúng tôi cần lưu ý rằng áp dụng ràng buộc Monotonic (partially) có thể giảm hiệu suất của mô hình vì nó hạn chế sự tùy chỉnh của mô hình. Nên trong một số trường hợp, việc sử dụng ràng buộc Monotonic có thể không cần thiết hoặc không hiệu quả.