Report: Implement CookieBlock Consent Classifier

▼ Feature Extraction

1. Dataset: 06_Training_Data/tranco_05May_20210510_201615.json
2. Trích xuất các feature từ training data cookies:

```
python prepare_training_data.py tranco_05May_20210510_201615.
```

3. Việc trích xuất tính năng được tiến hành theo 3 giai đoạn (chi tiết trong file CookieBlock-Consent-Classifier/feature_extraction/features.json):
   - Các features được trích xuất một lần trên mỗi cookie
   - Các features được trích xuất một lần mỗi lần cập nhật
   - Các features xuất phát từ sự khác biệt trong bản cập nhật
4. Sau 10-15p ta thu được kết quả là một ma trận thưa thớt dưới định dạng nhị phân trong folder processed_features/ bao gồm:

```
processed_features/feature_map_<timestamp>.txt
processed_features/processed_<timestamp>.sparse
processed_features/processed_<timestamp>.sparse.feature_names
processed_features/processed_<timestamp>.sparse.labels
processed_features/processed_<timestamp>.sparse.weights
```

- `processed_features/feature_map_<timestamp>.txt` ghi lại các feature được đặt tại các chỉ mục trong feature array và có thể được sử dụng với XGBoost để xuất thông tin về các feature quan trọng nhất

- `processed_<timestamp>.sparse` chứa các vectơ dữ liệu huấn luyện.
- `processed_<timestamp>.sparse.labels` chứa các nhãn sự thật cơ bản, theo cùng thứ tự với các vectơ dữ liệu huấn luyện.
- `processed_<timestamp>.sparse.weights` chứa trọng số mẫu đầu vào.
  Chống lại sự mất cân bằng trong tập dữ liệu trong quá trình đào tạo bộ phân loại bằng cách sử dụng các trọng số này.
- `processed_<timestamp>.sparse.feature_names` chứa tên tính năng mà con người có thể đọc được cho mỗi chỉ mục.

▼ Giải thích code file: prepare_training_data.py

1. `logger = logging.getLogger("feature-extract")`: Tạo một logger với tên "feature-extract". Logger này sẽ được sử dụng để ghi lại các thông tin liên quan đến quá trình trích xuất đặc trưng.
2. `featuremap_path = "feature_extraction/features.json"`: Đường dẫn đến file JSON chứa thông tin về cách chuyển đổi cookies thành vector đặc trưng.
3. `def setup_logger() -> None:`: Đây là một hàm để thiết lập logger. Nó đặt mức độ ghi log là DEBUG, thêm một handler để ghi log ra màn hình console với mức độ INFO, và một handler khác để ghi log ra file với mức độ DEBUG.
4. `def main() -> int:`: Đây là hàm chính của tập lệnh. Nó trích xuất các vector đặc trưng từ dữ liệu cookie JSON được cung cấp và trả về một mã thoát (0 nếu thành công, 1 nếu có lỗi).
   - Đầu tiên, nó thiết lập logger.
   - Sau đó, nó tải dữ liệu đầu vào và kiểm tra lỗi. Nếu dữ liệu không tồn tại hoặc không phải là file JSON hợp lệ, nó sẽ ghi log lỗi và trả về mã thoát tương ứng.
   - Nếu dữ liệu được tải thành công, nó sẽ ghi log thông tin và thêm dữ liệu vào `input_data`.
   - Cuối cùng, nó kiểm tra định dạng đầu ra được yêu cầu và ghi log thông tin tương ứng. Nếu định dạng không được hỗ trợ, nó sẽ ghi log lỗi và trả về mã thoát 3.

5. `feature_processor = CookieFeatureProcessor(featuremap_path, skip_cmp_cookies=True)`: Khởi tạo một đối tượng `CookieFeatureProcessor` với đường dẫn đến file JSON chứa thông tin về cách chuyển đổi cookies thành vector đặc trưng. `skip_cmp_cookies=True` nghĩa là bỏ qua các cookie từ các công ty so sánh.
6. `feature_processor.print_feature_info()`: In ra số lượng đặc trưng sẽ được trích xuất từ mỗi mục dữ liệu đào tạo.
7. `feature_processor.extract_features_with_labels(input_data)`: Trích xuất các vector đặc trưng từ `input_data`.
8. `dump_path: str = "./processed_features/"`: Đường dẫn đến thư mục sẽ chứa các file đặc trưng đã xử lý.
9. `os.makedirs(dump_path, exist_ok=True)`: Tạo thư mục `dump_path` nếu nó chưa tồn tại.
10. `if cargs["--out"]: basefn = cargs["--out"] else: basefn = f"processed_{now_str}"`: Nếu tùy chọn `-out` được cung cấp, sử dụng nó làm tên cơ sở cho file đầu ra. Nếu không, sử dụng chuỗi "processed_" kèm theo thời gian hiện tại.
11. `if cargs["--format"] == "xgb": ... elif cargs["--format"] == "sparse": ... elif cargs["--format"] == "libsvm": ... elif cargs["--format"] == "debug": ... else: raise ValueError(...)`: Dựa vào định dạng đầu ra được yêu cầu, lưu các vector đặc trưng đã xử lý vào file tương ứng.
12. `if not cargs["--format"] == "debug": ...`: Nếu định dạng đầu ra không phải là "debug", lưu bản đồ đặc trưng vào một file riêng.
13. `feature_processor.reset_processor()`: Đặt lại trạng thái của `feature_processor` để sẵn sàng cho lần chạy tiếp theo.
14. `return 0`: Trả về 0 để chỉ ra rằng tập lệnh đã hoàn thành thành công.
15. `if __name__ == "__main__": exit(main())`: Nếu tập lệnh này được chạy như một tập lệnh chính (không phải được import), thì gọi hàm `main()` và thoát với mã thoát mà `main()` trả về.

▼ XGBoost Classifier

▼ 1. Train XGBoost

```
python train_xgb.py ../processed_features/processed_<timest
```

- Câu lệnh trên sẽ chạy một vòng duy nhất của quá trình phân chia đào tạo/kiểm tra `80/20`, đào tạo mô hình trên 80% dữ liệu và xác thực nó trên phần còn lại.
- Quá trình sẽ chạy trong khoảng 10-15 phút trên VM. Sau khi hoàn thành, nó sẽ xuất ra một phạm vi lớn của điểm xác thực thành đầu ra tiêu chuẩn và vào tệp nhật ký, được lưu trữ tại `train_xgb.log`.
- Vì phương sai thấp nên ngay cả kết quả đầu ra của một lần cũng phải nhất quán và có thể so sánh được với kết quả trong paper. Lưu ý rằng trong trường hợp một mảng được liệt kê, các lớp cookie luôn theo thứ tự `[necessary, functional, analytics, advertising].`
- Nó cũng xuất ra confusion matrix sau mỗi lần chạy:

```
2023-12-21-09:34:48 :: classifier :: INFO :: Predicted labe
2023-12-21-09:34:48:: classifier :: INFO :: Confusion Matri
[[ 9402  1109   628   430]
[  416  2900   220   206]
[  515   556 15871   751]
[  394   948   920 20088]]
```

- Các hàng ở đây thể hiện các nhãn thực, trong khi các cột thể hiện các dự đoán.
- Giống như mảng, thứ tự của các lớp là `[necessary, functional, analytics, advertising].`
- Tệp mô hình XGBoost sẽ được ghi vào thư mục:

```
./classifiers/models/xgbmodel_<timestamp>.xgb
```

- Trong khi dữ liệu xác thực sẽ được xuất ra thư mục:

```
./classifiers/xgb_predict_stats/validation_matrix_<timestar
```

▼ Giải thích code file: train_xgb.py

1. `get_search_params() -> Dict[str, List[Union[None, float, int, str]]]`: Trả về một từ điển chứa các tham số tìm kiếm cho mô hình XGBoost. Mỗi tham số là một danh sách các giá trị có thể.
2. `get_best_params() -> Dict[str, Union[None, float, int, str]]`: Trả về một từ điển chứa các tham số tốt nhất được tìm thấy cho mô hình XGBoost.
3. `save_model(bst) -> None`: Lưu mô hình XGBoost đã được huấn luyện vào một file.
4. `custom_metrics(lweights: np.ndarray, data) -> List[Tuple[str, float]]`: Tính toán và trả về các chỉ số đánh giá tùy chỉnh cho mô hình XGBoost.
5. `analyse_feature_contribs(pred_probabilities: np.ndarray, feature_contributions: np.ndarray, test_labels: np.ndarray, class_names: List[str], eval_path: str) -> None`: Phân tích đóng góp của từng đặc trưng trong việc dự đoán chính xác một lớp cụ thể. Kết quả được lưu vào một file.
6. `output_validation_statistics`: Hàm này nhận vào một mô hình đã được huấn luyện (bst), số lượng cây tối đa để dự đoán (ntree), một biến boolean để quyết định có tính toán đóng góp của các đặc trưng hay không (compute_feat_contribs), tập dữ liệu kiểm thử (X_test), nhãn của tập kiểm thử (y_test), và tên của các lớp (class_names). Hàm này sẽ tính toán và xuất ra một số thống kê từ việc dự đoán trên tập kiểm thử.
7. `simple_train`: Hàm này nhận vào tập dữ liệu huấn luyện (X), nhãn của tập huấn luyện (y), và trọng số của mỗi điểm dữ liệu (weights). Hàm này sẽ huấn luyện mô hình trên toàn bộ tập dữ liệu đã cung cấp và lưu mô hình đã huấn luyện.
8. `split_train`: Hàm này nhận vào tập dữ liệu huấn luyện (X), nhãn của tập huấn luyện (y), trọng số của mỗi điểm dữ liệu (weights), tỷ lệ dữ liệu dùng để huấn luyện (train_fraction), và một biến boolean để quyết định có áp dụng trọng số cho tập kiểm thử hay không (apply_weights_to_validation). Hàm này sẽ chia tập dữ liệu thành hai phần (huấn luyện và kiểm thử), huấn luyện mô hình trên phần dữ liệu huấn luyện, và sau đó xuất ra thống kê từ việc dự đoán trên tập kiểm thử.

9. `split_train_crossvalidate`: Hàm này thực hiện việc huấn luyện và kiểm thử mô hình XGBoost thông qua phương pháp K-Fold Cross Validation. Trong mỗi lần lặp, dữ liệu được chia thành tập huấn luyện và tập kiểm thử dựa trên chỉ số từ KFold. Mô hình được huấn luyện trên tập huấn luyện và sau đó được kiểm thử trên tập kiểm thử. Kết quả kiểm thử được ghi lại và xuất ra.
10. `crossvalidate_train`: Hàm này thực hiện việc huấn luyện mô hình XGBoost thông qua phương pháp Cross Validation. Mô hình được huấn luyện trên toàn bộ tập dữ liệu và kết quả của quá trình Cross Validation được ghi lại và xuất ra.
11. `paramsearch_train`: Hàm này thực hiện việc tìm kiếm tham số tốt nhất cho mô hình XGBoost thông qua phương pháp Grid Search hoặc Random Search. Mô hình được huấn luyện với mỗi tổ hợp tham số và kết quả của quá trình tìm kiếm tham số được ghi lại và xuất ra.
12. `main`: huấn luyện mô hình XGBoost. Nó nhận các đối số dòng lệnh để chỉ định dữ liệu huấn luyện, trọng số lớp, và chế độ huấn luyện.
    - Các chế độ huấn luyện có thể là "train", "split", "cross_validate", "grid_search", hoặc "random_search". Tùy thuộc vào chế độ, nó sẽ gọi các hàm khác nhau để huấn luyện mô hình.
    - Dưới đây là mô tả ngắn gọn về từng chế độ:
        - "train": Huấn luyện mô hình sử dụng hàm `simple_train`.
        - "split": Huấn luyện mô hình sử dụng phương pháp chia tập huấn luyện - kiểm thử với hàm `split_train`.
        - "cross_validate": Huấn luyện mô hình sử dụng phương pháp kiểm định chéo với hàm `split_train_crossvalidate`.
        - "grid_search": Huấn luyện mô hình sử dụng tìm kiếm lưới để tìm các tham số tốt nhất với hàm `paramsearch_train`.
        - "random_search": Huấn luyện mô hình sử dụng tìm kiếm ngẫu nhiên để tìm các tham số tốt nhất với hàm `paramsearch_train`.
    - Hàm trả về một mã trạng thái kiểu số nguyên, với các giá trị khác không cho thấy lỗi.

▼ 2. Training without splitting & cross-validation

Để chạy đào tạo mà không phân tách dữ liệu, chỉ cần thực hiện lệnh sau:

```
python train_xgb.py ../processed_features/processed_<timest
```

Để chạy xác thực chéo, thay vào đó chạy lệnh sau:

```
python train_xgb.py ../processed_features/processed_<timest
```

Thao tác này sẽ chia tập dữ liệu huấn luyện thành 5 phần, tạo ra một đầu ra xác thực cho mỗi phần.