ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

ШКОЛА ИНФОРМАТИКИ, ФИЗИКИ И ТЕХНОЛОГИЙ

Руководство Разработчика

для приложения "Fake Reviews Detector

Разработчики, telegram:

Феськов Герман Дмитриевич, @gerashark

Рогов Данил Вадимович, @danrogov

Группа: БКОТСИС241С

Руководитель:

Ролич Алексей Юрьевич

Санкт-Петербург 2025

Оглавление

0.1 Модуль `data_loader.py`

Импортируемый Модуль `data_loader.py` отвечает за загрузку и подготовку исходного набора данных. Состоит из двух ключевых функций:

- `download_kaggle_dataset(image: str) -> pathlib.Path`

  Скачивает весь архив датасета с Kaggle по идентификатору `<owner>/<dataset-name>` и распаковывает его в локальную папку-кэш. Возвращает путь к директории, где находятся файлы.

- `load_raw_dataset(config_path: str) -> pandas.DataFrame`

  Читает YAML-конфигурацию по пути `config_path`, извлекает из неё параметры:

  - `image` — идентификатор для скачивания;

0.2 Модуль `preprocessing.py`

Импортируемый Модуль `preprocessing.py` отвечает за очистку и нормализацию текстовых данных перед обучением и предсказанием. Включает следующие функции:

- `clean_text(text: str, p_cfg: dict) -> str`

  Очищает одну строку отзыва согласно параметрам конфигурации `p_cfg`:

  - приведение к нижнему регистру (`lowercase`),
  - удаление пунктуации (`remove_punctuation`),
  - удаление стоп-слов (`remove_stopwords`),
  - стемминг (`stemming`),
  - лемматизация (`lemmatization`).

- `rename_and_map(df: pd.DataFrame, cfg: dict) -> pd.DataFrame`

  Переименовывает исходные колонки в `review` и `label`, а затем мапит значения меток (`good`/`bad`) в числовой формат (1/0).

- `clean_reviews(df: pd.DataFrame, p_cfg: dict) -> pd.DataFrame`

  Применяет `clean_text` ко всем строкам колонки `review` и возвращает обновлённый DataFrame.

- `create_processed_csv(raw_csv_path: str, config: dict, output_csv_path: str) -> pd.DataFrame`

  Обрабатывает сырые данные файла `raw_csv_path`, с параметрами, заданными в `config` результат записывается в файл `output_csv_path`. Пример `config`:

```
preprocessing:
  lowercase: true # приведение к нижнему регистру
  remove_punctuation: true # удаление лишних знаков
  remove_stopwords: true # удаление лишних слов
  stemming: false # обрезание слов
  lemmatization: true # приведение слов к начальной форме
  max_features: 10000 # максимальное кол-во учитываемых токе
   нов
  ngram_range: [1, 2] # диапазон n-грамм

```

0.3 Модуль `train.py`

Немпортируемый модуль `train.py` отвечает за построение векторизатора, обучение модели и сохранение артефактов. Содержит следующие ключевые функции:

- `split_data(df: pd.DataFrame, tr_cfg: dict) -> (X_train, X_test, y_train, y_test)`

  Делит DataFrame на тренировочный и тестовый наборы по параметрам из `tr_cfg`:

  - `test_size` — доля тестовой выборки,
  - `random_state` — зерно генератора,
  - `stratify` — стратификация по меткам.

- `build_vectorizer(v_cfg: dict) -> Vectorizer`

  Создаёт `TfidfVectorizer` или `CountVectorizer` согласно настройкам:

  - `type` — `tfidf` или `count`,
  - `max_features` — максимальное число признаков,
  - `ngram_range` — диапазон n-грамм.

- `build_model(m_cfg: dict) -> Classifier`

  Инициализирует классификатор на основе `m_cfg`:

  - **Logistic Regression** (`logistic_regression`): классический логистический регрессор, быстро обучается и даёт интерпретируемые веса признаков.
  - **Support Vector Machine** (`svm`): `SVC` с ядром `rbf` по умолчанию, хорошо работает с высокоразмерными данными.
  - **Random Forest** (`random_forest`): `RandomForestClassifier`, устойчив к переобучению, предоставляет оценку важности признаков.
  - **Naive Bayes** (`naive_bayes`): `MultinomialNB`, эффективен на текстовых данных с частотами токенов.

- `train_model(cfg: dict) -> None`

  Полный цикл обучения:

  1. Загрузка очищенного CSV по пути `cfg["dataset_path"]`.

2. Предобработка: заполнение пустых `review`, удаление пропусков, приведение `label` к типу `int`.
3. Вызов `split_data`, `build_vectorizer`, обучение векторизатора на `X_train`.
4. Векторизация `X_test` и обучение модели на `X_train_vec, y_train`.
5. Оценка качества: `accuracy_score` и `classification_report`.
6. Сохранение векторизатора и модели в пути `cfg["vectorizer_path"]` и `cfg["model_path"]`.

Пример использования

```python
from fake_reviews_detector.train import train_model
from fake_reviews_detector.utils import load_yaml_config

cfg = load_yaml_config("config/local_dev.yaml")
train_model(cfg)
```

Настройки конфигурации

```yaml
vectorizer:
  type: tfidf           # выбор схемы векторизации
  max_features: 10000   # число признаков
  ngram_range: [1, 2]   # диапазон n-грамм
model:
  type: logistic_regression      # тип модели: logistic_regression, svm, random_forest, naive_bayes
  hyperparameters:
    C: 1.0
    max_iter: 1000
    solver: lbfgs
training:
  test_size: 0.2        # доля тестовой выборки
  random_state: 42      # зерно генератора
  stratify: true        # стратификация по меткам

```

Примеры конфигурации для различных моделей

Для разных типов классификаторов секция `model` в файле `config/local_-dev.yaml` может выглядеть так:

Support Vector Machine (SVM)

```yaml
model:
  type: svm
  hyperparameters:
    C: 1.0          # параметр регуляризации
    kernel: rbf     # ядро: linear, poly, rbf, sigmoid
    gamma: scale    # коэффициент ядра
    max_iter: -1    # без ограничения по итерациям
```

Random Forest

```yaml
model:
  type: random_forest
  hyperparameters:
    n_estimators: 100   # количество деревьев
    max_depth: 10       # максимальная глубина
    random_state: 42    # зерно для воспроизводимости
```

Naive Bayes

```yaml
model:
  type: naive_bayes
  hyperparameters:
    alpha: 1.0          # параметр Лапласовского сглаживания
    fit_prior: true     # учитывать априорное распределение
```

Logistic Regression (для сравнения)

```yaml
model:
  type: logistic_regression
  hyperparameters:
    C: 1.0
    max_iter: 1000
    solver: lbfgs
```

0.4 Модуль `preview.py`

Неимпортируемый Модуль `preview.py` предоставляет функционал быстрого прототипирования и проверки модели на новых отзывах. Основные функции:

- `load_artifacts(model_path: pathlib.Path, vectorizer_path: pathlib.Path) -> Tuple[Vectorizer, Classifier]`

  Загружает обученные артефакты из файлов:

  - `vectorizer.pkl`: объект векторизатора,
  - `model.pkl`: объект обученной модели.

- `clean_text(text: str, p_cfg: dict) -> str`

  (Импортируется из `preprocessing.py`) Очищает текст по тем же правилам, что при обучении.

- `preview(texts: list[str], config_path: str) -> pandas.DataFrame`

  Полный цикл предсказания:

  1. Загрузка конфигурации и артефактов через `load_artifacts`.
  2. Очистка входных отзывов: `clean_text`.
  3. Векторизация очищенных текстов.
  4. Предсказание числовых меток.
  5. Обратное маппирование в оригинальные метки (“good”/“bad”).

  Возвращает `DataFrame` с колонками: `['raw', 'cleaned', 'pred_numeric', 'pred_label']`.

Пример использования

```
from fake_reviews_detector.preview import preview
from fake_reviews_detector.utils import load_yaml_config

cfg = load_yaml_config("config/local_dev.yaml")
samples = ["Great product!", "Terrible experience."]
df_out = preview(samples, "config/local_dev.yaml")
print(df_out)
```

0.5 Архитектура приложения

Приложение организовано как набор модулей, каждый из которых выполняет свою задачу:

- `data_loader.py` — загрузка и чтение исходных CSV-файлов;
- `preprocessing.py` — очистка и нормализация текста (lowercase, punctuation removal, stopwords, stemming, lemmatization);
- `train.py` — разделение данных, построение векторизатора, обучение моделей, оценка качества и сохранение артефактов;
- `preview.py` — быстрый инференс на новых отзывах (загрузка модели и векторизатора, очистка, предсказание);
- `gui.py` — графический интерфейс на базе `Tkinter`;
- `utils.py` — вспомогательные функции (загрузка конфигурации, логирование);
- `main.py` — точка входа, инициализация и запуск GUI.

0.6 Зависимости и используемые библиотеки

В проекте используются следующие основные библиотеки и инструменты:

- **Python 3.11+** — язык разработки.
- **Anaconda** — среда разработки
- **pandas** — работа с табличными данными (DataFrame).
- **numpy** — математические операции и массивы.
- **scikit-learn** — векторизация (TF-IDF/Count), модели машинного обучения, метрики.
- **nltk** — лингвистическая обработка: стоп-слова, стемминг, лемматизация.
- **pyyaml** — чтение YAML-конфигураций.
- **kagglehub** — скачивание датасетов с Kaggle.
- **tkinter** — создание графического интерфейса.
- **joblib** — сериализация/десериализация моделей и векторизаторов.
- **tqdm** — индикаторы прогресса в консоли.
- **matplotlib, seaborn** — (опционально) визуализация результатов анализа.

0.7 Параметры конфигурации

В файле `config/local_dev.yaml` задаются пути и файлы, используемые приложением:

`dataset_path` Путь к CSV-файлу с обработанными данными, например:

```
dataset_path: data/processed/data.csv
```

`model_path` Путь для сохранения обученной модели:

```
model_path: data/model/model.pkl
```

`vectorizer_path` Путь для сохранения объекта векторизатора:

```
vectorizer_path: data/model/vectorizer.pkl
```

`log_file` Файл лога, куда перенаправляются stdout и stderr:

```
log_file: data/log.txt
```