Code-Konzept zur Vorhersage der Kreditwürdigkeit von Joshua Meyer und Nechirvan Zibar

1. Einleitung

1.1 Ziel des Projekts

Das Ziel dieses Projekts ist die Entwicklung einer Machine Learning - Pipeline zur Vorhersage der Betrugswahrscheinlichkeit von Kreditkartentransaktionen. Dies ermöglicht es Banken, fundierte Entscheidungen bei der Betrugserkennung zu treffen, indem eine Transaktion basierend auf historischen Daten als kritisch bzw. unkritisch bewertet wird.

1.2 Relevanz des Use-Cases

Die genaue Vorhersage der Betrugswahrscheinlichkeit ist für Finanzinstitute von entscheidender Bedeutung. Jährlich entstehen durch betrügerische Aktivitäten Milliardenverluste, die nicht nur die Institute selbst, sondern auch die Kunden und die gesamte Wirtschaft beeinträchtigen.

2. Klassenbeschreibung

1. CreditCard Klasse
   - Diese Klasse ist eigenständig und verwaltet nur die Transaktionen.

2. DataProcessor Klasse
   - Diese Klasse verarbeitet die Daten und erstellt die Trainings- und Testsets.

3. ModelTrainer Klasse
   - Diese Klasse trainiert das Modell mit den vom DataProcessor bereitgestellten Daten.

4. ModelManager Klasse
   - Diese Klasse verwaltet das Laden, Speichern und Vorhersagen des Modells.

5. ModelEvaluator Klasse
   - Diese Klasse bewertet das trainierte Modell mit Testdaten.

6. FraudDetectionService Klasse
   - Diese Klasse koordiniert die anderen Klassen, um den gesamten Betrugserkennungsprozess zu implementieren.
   - Sie dient als Abstraktionsschicht für potenzielle APIs, GUIs etc.

Mit Klassendetails:

1. CreditCard Klasse:
   - Attribute:
     - transactions: Liste für Transaktionsdaten.
   - Methoden:

- add_transaction(transaction): Fügt eine Transaktion zur Liste hinzu.
- get_transactions(): Gibt die Liste der Transaktionen zurück.

2. DataProcessor Klasse:
- Attribute:
  - data_path: Pfad zur Datenquelle.
  - x_train, x_test, y_train, y_test: Trainings- und Testdaten für die Modellierung.
  - test_split: Prozentsatz der Daten für den Testsplit.
  - scaler: StandardScaler für die Datenvorverarbeitung.
- Methoden:
  - __init__(data_path, test_split): Initialisiert mit Pfad und Optionen.
  - process_data(): Ladet, teilt und verarbeitet die Daten.
  - load_data(): Lädt die Daten aus der angegebenen Datei.
  - split_data(): Teilt die Daten in Trainings- und Testsets.
  - preprocess_data(): Verarbeitet die Daten mit einem Scaler.

3. ModelTrainer Klasse:
- Attribute:
  - x_train, y_train: Trainingsdaten und -labels.
  - model: Trainiertes Modell, das für die Vorhersage verwendet wird.
- Methoden:
  - __init__(x_train, y_train): Initialisiert mit Trainingsdaten.
  - train_model(n_estimators, random_state): Trainiert das Modell (Random Forest) mit den angegebenen Parametern.

4. ModelManager Klasse:
- Attribute:
  - model_path: Pfad zur gespeicherten Modelldatei.
  - model: Geladenes Modell als InferenceSession.
- Methoden:
  - __init__(model_path): Initialisiert den Manager mit dem Modellpfad.
  - save_model(model, x_train): Speichert das Modell als ONNX-Datei.
  - load_model(): Lädt das Modell aus der ONNX-Datei.
  - get_prediction(data): Macht eine Vorhersage mit dem geladenen Modell.

5. ModelEvaluator Klasse:
- Attribute:
  - model: Trainiertes Modell (InferenceSession).
  - x_test, y_test: Testdaten und -labels für die Modellbewertung.
- Methoden:
  - __init__(model, x_test, y_test): Initialisiert mit Modell und Testdaten.
  - evaluate_model(): Evaluieren des Modells und Ausgabe der Metriken.
  - visualize_confusion_matrix(): Visualisiert die Confusion Matrix der Modellvorhersagen.

6. FraudDetectionService Klasse:
- Attribute:

- model_manager: Instanz von ModelManager für das Laden und Speichern des Modells.
- data_processor: Instanz von DataProcessor für die Datenverarbeitung.
- model_trainer: Instanz von ModelTrainer für das Trainieren des Modells.
- model_evaluator: Instanz von ModelEvaluator für die Evaluation des Modells.
- credit_card: Instanz von CreditCard für die Transaktionsverwaltung.
- Methoden:
- __init__(model_path): Initialisiert den Service mit dem Modellpfad.
- load_model(): Lädt das Modell oder trainiert und speichert es.
- evaluate_model(): Evaluieren des Modells und Ausgabe der Leistungsmetriken.
- add_transaction_and_evaluate(transaction_data): Fügt eine Transaktion hinzu und macht eine Vorhersage.
- get_prediction(data): Gibt eine Vorhersage für die gegebenen Daten zurück.

3. Coding-Guidelines

3.1 PEP 8 Konformität
- Der Code folgt den PEP 8 Richtlinien, um Lesbarkeit und Konsistenz zu gewährleisten.

3.2 Kommentare und Dokumentation
- Jeder wichtige Abschnitt des Codes wird mit Kommentaren versehen, um die Funktionsweise zu erläutern. Wichtig ist auch ‚warum', nicht nur ‚wie'.
- Funktionen und Klassen enthalten Docstrings, die ihre Funktion beschreiben.

3.3 Modularität und Wiederverwendbarkeit
- Der Code ist in modularen Komponenten organisiert, die wiederverwendbar und erweiterbar sind.

3.4 Logging
- Anstelle von Print-Statements wird das Logging-Modul verwendet, um Informationen und Fehler zu protokollieren.

3.5 Single Responsibility Principle (SRP)
- Jede Klasse sollte nur eine einzige Aufgabe haben. Dies reduziert die Kopplung und erhöht die Kohäsion.

5. Aufwandsschätzung

5.1 Vorbereitungsphase (ca. 5 Stunden)
- Datensammlung, Anforderungsanalyse, Design der Klassenstruktur

5.2 Implementierungsphase (ca. 15 Stunden)
- Entwicklung der einzelnen Module und Integration der Pipeline

5.4 Optimierungs- /Testphase (ca. 5 Stunden)
- Performance-Tuning und Code-Refactoring

5.5 Dokumentations- und Präsentationsphase (ca. 5 Stunden)

- Erstellung der Dokumentation und Vorbereitung der Präsentation