**Code-Konzept zur Vorhersage der Kreditwürdigkeit**

**1. Einleitung**

**1.1 Ziel des Projekts**

**Das Ziel dieses Projekts ist die Entwicklung einer Machine Learning - Pipeline zur Vorhersage der Kreditwürdigkeit von Kunden. Dies ermöglicht es Banken, fundierte Entscheidungen bei der Kreditvergabe zu treffen, indem das Ausfallrisiko eines Kredits basierend auf historischen Daten bewertet wird.**

**1.2 Relevanz des Use-Cases**

**Die genaue Vorhersage der Kreditwürdigkeit ist für Finanzinstitute von entscheidender Bedeutung, um Verluste zu minimieren und profitable Geschäftsbeziehungen zu fördern. Ein datengetriebenes Modell bietet eine objektive Grundlage für Kreditentscheidungen.**

**2. Klassenbeschreibung**

**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.**

**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.**

**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.**

**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.**

**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.**

**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.**

**Diese Klassen interagieren miteinander, indem sie Daten laden, trainieren, vorverarbeiten, Modelle speichern und laden, Modelle evaluieren und Vorhersagen treffen. Der FraudDetectionService koordiniert die Arbeit der anderen Klassen, um das Gesamtsystem für die Betrugserkennung bereitzustellen.**

**3. Coding-Guidelines**

**3.1 PEP 8 Konformität**

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

**- Beispielsweise wird für die Benennung von Variablen und Funktionen das snake\_case verwendet.**

**3.2 Kommentare und Dokumentation**

**- Jeder wichtige Abschnitt des Codes wird mit Kommentaren versehen, um die Funktionsweise zu erläutern.**

**- 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.**

**5. Aufwandsschätzung und Effiziente Implementierung**

**5.1 Vorbereitungsphase (ca. 5 Stunden)**

**- Datensammlung, Anforderungsanalyse, Design der Klassenstruktur**

**5.2 Implementierungsphase (ca. 20 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**

**6. Zusammenfassung und Ausblick**

**Dieses Projekt zielt darauf ab, eine Machine Learning - Pipeline zur Vorhersage der Kreditwürdigkeit zu entwickeln. Es werden verschiedene Klassen für die Datenverarbeitung, Modelltraining und -bewertung verwendet. Durch die Einhaltung von Coding-Guidelines und einer klaren Struktur wird eine effiziente Implementierung und Wartung des Codes gewährleistet. Zukünftige Erweiterungen könnten zusätzliche Datenquellen und fortschrittlichere Modelle umfassen.**