Erkennung fehlerhafter Bezahlvorgänge an Selbstbedienungskassen im Einzelhandel

Analyse

Dominik Lewin, Mario Teßmann, Johannes Winkler

4. Juli 2024

1 Anforderungen an Analyseverfahren

Die in Frage kommenden Analyseverfahren sollten einige Anforderungen erfüllen:

- Genauigkeit: Das Analyseverfahren soll in der Lage sein, die Daten korrekt zu klassifizieren. Falsche Klassifizierungen führen zu Gewinneinbußen des Auftraggebers. Bei falsch-positiven Klassifikationen (zu Unrecht beschuldigte Kunden) kommt es zu einer Kundenunzufriedenheit und damit einhergehend zu weiteren noch gravierenderen Gewinneinbußen.
- Robustheit: Das Analyseverfahren soll robust gegenüber Rauschen und Ausreißern sein. Wie schon in der explorativen Datenanalyse angemerkt, ist beides im Datensatz vertreten. Die Analyseverfahren sollten dadurch nicht beeinträchtigt werden.
- Umsetzbarkeit: Das Analyseverfahren soll effizient sein und mit angemessenen Rechenzeiten und der vorhandenen Hardware trainiert und ausgeführt werden können.
- Interpretierbarkeit: Auf Wunsch der Auftraggeberin sollte das Verfahren interpretierbar sein, d.h. die Ergebnisse sollen leicht verständlich und erklärbar sein.
- Reproduzierbarkeit: Das Analyseverfahren soll reproduzierbar sein und auf denselben Trainings- und Testdaten immer dieselben Ergebnisse liefern. Insbesondere sollen so die Ergebnisse der Teammitglieder vergleichbar gemacht werden. Zudem sollen die Ergebnisse der unterschiedlichen Modelle vergleichbar sein.

2 Identifikation geeigneter Analyseverfahren

Die Problemstellung wurde bereits in vorausgehenden Phasen ausführlich erläutert. Kurz gesagt ist das Ziel, möglichst viele fehlerhafte Einkäufe anhand von Daten zu erkennen. Bei diesem Problem handelt es sich um ein Klassifikationsproblem. Für einen neuen Einkauf soll eine Klasse *normal (0)* oder *fraud (1)* vorhergesagt werden. Die Herausforderung beim Datensatz liegt darin, dass dieser Rauschen und Ausreißer enthält, mit denen die Analyseverfahren umgehen können müssen, insbesondere weil die Ausreißer teilweise sehr relevant für die Klassifikation sind.

Da es zu diesem Thema keine vergleichbaren Publikationen gibt, werden zunächst die gängigen Klassifikationsalgorithmen des maschinellen Lernens in Betracht gezogen. Diese sollen in den nachfolgenden Unterkapiteln kurz erläutert werden.

2.1 Entscheidungsbaum

Entscheidungsbäume treffen Entscheidungen durch das Aufspalten des Datensatzes in immer kleinere Teilmengen. So sollen die aufgespaltenen Daten immer ähnlicher werden und im besten Fall zu einer einheitlichen Klassifikation führen. Entscheidungsbäume kommen gut mit wenigen Merkmalen aus und sind auch robust gegenüber Rauschen und Ausreißern. Der Nachteil ist allerdings, dass Entscheidungsbäume zur Überanpassung an den Trainingsdatensatz neigen.

2.2 Random Forest

Beim Random Forest handelt es sich um ein Ensemble-Verfahren, bei dem mehrere Entscheidungsbäume kombiniert werden. Ziel ist es, die zuvor für Entscheidungsbäume genannte Wahrscheinlichkeit einer Überanpassung zu minimieren und die Genauigkeit der Klassifikationen zu steigern. Wie auch schon die Entscheidungsbäume ist dieser Algorithmus robust gegenüber Ausreißern und Rauschen und dementsprechend gut für den vorhandenen Datensatz geeignet. Dies ist auch einer der vielversprechendsten Algorithmen, da in der Regel sehr gute Ergebnisse für Klassifikationsprobleme erzielt werden können.

2.3 Naiver Bayes Klassifikator

Der Naive Bayes Klassifikator basiert auf dem Satz von Bayes und nimmt an, dass alle Attribute unabhängig voneinander sind. Da im Datensatz allerdings viele Attribute korrelieren (beispielsweise der Kaufpreis oder die Scan-Dauer mit der Anzahl der Artikeln), ist dieser Algorithmus für den gegebenen Datensatz und die Problemstellung weniger relevant und wird vermutlich zu schlechten Ergebnissen führen.

2.4 Logistische Regression

Auch wenn der Name etwas irreführend ist, handelt es sich hierbei um einen klassischen Klassifikator. Mittels der Sigmoidfunktion werden verschiedene Attribute auf Werte zwischen 0 und 1 abgebildet. Wenn man diese als Wahrscheinlichkeiten interpretiert und die Wahrscheinlichkeit des Eintritts eines Ereignisses ab 0,5 festlegt, kann man die logistische Regression als binären Klassifikator verwenden. Die logistische Regression kann allerdings empfindlich gegenüber Ausreißern sein, da versucht wird, eine lineare Entscheidungsgrenze zu finden, die die beiden Klassen trennt. Extreme Werte haben bei diesem Algorithmus einen relativ großen Einfluss, weshalb dieser weniger geeignet erscheint.

2.5 Support Vector Machine (SVM)

SVMs versuchen eine Hyperebene in den Daten zu erzeugen, um die Daten beider Klassen bestmöglich voneinander zu trennen. Der Algorithmus ist empfindlich gegenüber Ausreißern und Rauschen, da jeder Datenpunkt in die Berechnungen mit einfließt. Dies ist für den vorhandenen Datensatz nicht vielversprechend. Da zudem während der explorativen Datenanalyse keine klare Trennung zwischen den Klassen gefunden werden konnte, wird dieser Algorithmus vermutlich nicht die erwünschten Ergebnisse erzielen können.

2.6 K-Nearest Neighbors (KNN)

Der KNN-Algorithmus macht von der Ähnlichkeit benachbarter Datenpunkte Gebrauch. Für einen neuen Datenpunkt werden die k nächsten Nachbarn betrachtet und genau die Klasse zugewiesen, die am häufigsten vorkommt. Der Algorithmus kann gut mit nicht-linearen Daten umgehen, allerdings ist dieser empfindlich gegenüber Rauschen und Ausreißern. Insbesondere sind die Unterschiede zwischen normalen und fehlerhaften Einkäufen so gering, dass dieser Algorithmus vermutlich ebenfalls keine zufriedenstellenden Ergebnisse erzielen wird.

2.7 Multilayer Perceptron (MLP)

MLPs sind neuronale Netze, bei denen mehrere verdeckte Schichten verwendet werden. So können komplexe Zusammenhänge erlernt werden. Das MLP ist eines der leistungsstärksten Algorithmen, allerdings werden hierfür grundsätzlich sehr viele Daten gebraucht. Der vorhandene Datensatz ist verhältnismäßig klein und durch die vielen Ausreißer könnte es hier zu falschen Klassifikationen kommen.

3 Auswahl der Analyseverfahren

Aufgrund der leichten und effizienten Umsetzung des Trainings von Modellen in Python, insbesondere durch die Bibliothek scikit-learn, werden zur Auswahl der erfolgversprechendsten Analyseverfahren erste Tests durchgeführt. Diese zeigen, dass der Random Forest den einfacheren Algorithmen wie der Logistischen Regression, SVM, KNN oder auch dem normalen Entscheidungsbaum

auf diesem Datensatz bereits ohne Hyperparameter-Tuning überlegen ist. Aufgrund dieses Erfolgs mit dem Random Forest liegt es nahe, auch Boosting-Algorithmen in Betracht zu ziehen. Wie beim Random Forest handelt es sich dabei um einen Algorithmus des Ensemble-Learning. Hier werden zunächst Entscheidungsbäume trainiert und zufällig gewählte Objekte klassifiziert. Für Datenobjekte, die in diesem Schritt falsch klassifiziert werden, werden zusätzliche Bäume trainiert, um diese Fehler zu korrigieren.

Die vielversprechendsten Algorithmen in Bezug auf den vorhandenen Datensatz und die Problemstellung sind:

- Random Forest,
- Multilayer Perceptron,
- Boosting-Algorithmen.

4 Umgang mit dem unbalancierten Datensatz

Da es sich bei dem Datensatz um einen unausgewogenen (unbalancierten) Datensatz handelt, bei dem die Klasse *fraud* (1) wesentlich seltener vorkommt, als die Klasse *normal* (0), kann es zudem sinnvoll sein, den Datensatz vor dem Training zu manipulieren. Die gängigen Techniken für den Umgang mit unbalancierten Daten werden in den folgenden Unterkapiteln erläutert.

4.1 Undersampling

Es werden Daten der häufig vorkommenden Klasse reduziert, indem Datenzeilen gelöscht werden, bis die Anzahl dem Vorkommen der anderen Klasse entspricht und der Datensatz ausgeglichen ist. Dies ist im vorliegenden Fall nicht sinnvoll, da der Datensatz ohnehin verhältnismäßig klein ist und durch das Undersampling Wissen aus den Daten verloren gehen kann.

4.2 Oversampling

Es werden künstliche Daten erzeugt, um die Anzahl der Daten der selten vorkommenden Klasse zu erhöhen. Dieses Vorgehen kann für den vorliegenden Datensatz sinnvoll sein, da keine Daten verloren gehen. Allerdings besteht auch das Risiko, die Daten zu verfälschen und nicht vorhandene Zusammenhänge zu erzeugen.

Für die Umsetzung des Oversamplings ist in Python bereits eine Bibliothek imbalanced-learn implementiert, die verschiedene Möglichkeiten des Oversamplings ermöglichen. Hier soll mit *SMOTE* exemplarisch eine dieser Möglichkeiten getestet werden. *SMOTE* erzeugt dabei künstliche Daten, indem Muster in den Daten gesucht werden und möglichst ähnliche Datenpunkte hinzugefügt werden.

4.3 Class Weight

Einige Algorithmen besitzen die Möglichkeit, anzugeben, dass der Datensatz unausgeglichen ist. Auf diese Weise wird beim Training automatisch berücksichtigt, dass eine Klasse in der Minderheit ist. Sofern die Algorithmen eine entsprechende Implementierung besitzen, wird diese auch getestet.

5 Anwendung von Analyseverfahren

Die Analyseverfahren werden in Python in Jupyter Notebooks angewendet. Dabei werden insbesondere auf vorhandene Bibliotheken wie scikit-learn zurückgegriffen.

Damit alle Teammitglieder während des Trainings unterschiedlicher Modelle vergleichbare Ergebnisse erzielen können, ist es wichtig, den vorhandenen Datensatz in einen festen Trainings- und Testdatensatz aufzuteilen. Dies erfolgt dabei mit der Funktion *train_test_split* der scikit-learn-Bibliothek. Dabei wird die Größe der Testdaten auf 20% festgelegt. Zudem wird der *random_state = 42* verwendet, um eine Reproduzierbarkeit der Trainings- und Testdaten zu gewährleisten. Da der Datensatz unausgeglichen ist und die Klasse *fraud (1)* deutlich unterrepräsentiert ist, ist es wichtig, die Daten so aufzuteilen, dass der Anteil der fraud-Fälle im Trainings- und Testdatensatz annähernd gleich ist. Dies erfolgt durch den Parameter *stratify*.

Je nach Algorithmus kann es sinnvoll sein, die Daten vor dem Training zu standardisieren. Dies kann mittels scikit-learn durchgeführt, indem der *StandardScaler* verwendet wird. Allerdings kommen Algorithmen wie der Random Forest auch ohne eine Standardisierung aus.

Um genauere Ergebnisse zu erhalten und die Wahrscheinlichkeit einer Überanpassung zu verringern, wird die Kreuzvalidierung eingesetzt. Dadurch kann besser verglichen werden, wie das trainierte Modell mit neuen Daten umgeht. Dabei wird die Funktion *StratifiedKFold* aus scikit-learn verwendet, die ebenfalls den *random_state = 42* verwendet, um eine Reproduzierbarkeit zu gewährleisten. Es werden 5 Folds verwendet, wodurch für jedes Modell 5 Ergebnisse verglichen werden können.

Da vorab nicht bekannt ist, welche Algorithmen sich für den Datensatz am besten eignen, werden mehrere Analyseverfahren berücksichtigt (vgl. Abschnitt 3), deren Anwendung in den nachfolgenden Unterkapiteln genauer dargestellt wird.

5.1 Random Forest

Mit Hilfe der Python-Bibliothek scikit-learn kann der Random Forest auf einfache Weise angewendet werden. Der Algorithmus ist bereits als *RandomForestClassifier* in der Bibliothek implementiert und wird auf dem aufbereiteten Datensatz trainiert.

Dabei bietet die scikit-learn-Bibliothek die Möglichkeit, sich die Wichtigkeit der einzelnen Attribute mittels des Attributs *feature_importance* grafisch anzeigen zu lassen. Auf diese Weise kann man schrittweise unwichtige Merkmale aus dem Datensatz entfernen, sofern sich das Ergebnis, hier gemessen am Gewinn, nicht weiter verschlechtert.

In einem zweiten Schritt wird das Hyperparameter-Tuning mittels der scikit-learn-Funktion *GridSeachCV* durchgeführt. Mit Hilfe dieser Funktion kann man für die verschiedenen Hyperparameter (*n_estimators, max_depth, min_sample_split* usw.) verschiedene Parameterwerte ausprobieren und sich die beste Kombination ausgeben lassen. Die getesteten Parametereinstellungen können Abbildung 1 entnommen werden. Dabei konnte eine deutliche Optimierung erreicht werden.

```
param_grid = {
    "n_estimators": [100, 200],
    "max_depth": [10,20],
    "min_samples_split": [2, 5,10],
    "min_samples_leaf": [1, 2, 4],
    "max_features": ["log2", "sqrt"],
    "bootstrap": [True, False]
}
```

Abbildung 1: Getestete Parameterkombinationen für den Random Forest.

Da es sich um einen unausgewogenen Datensatz handelt, haben wir auch den Parameter *class_weight* auf *balanced* bzw. *balanced_subsample* gesetzt. Dies führte allerdings zu keiner Verbesserung. Auch die Anwendung von *SMOTE* brachte lediglich eine drastische Verschlechterung, weshalb nachfolgend auch keine weiteren Versuche mehr unternommen wurden, den Datensatz auszubalancieren.

5.2 Multilayer Perceptron (MLP)

Es wurden verschieden große und verschieden tiefe neuronalen Netze ausprobiert. Die Performance dieser verschiedenen Netze schwankt stark, allerdings ohne die Ergebnisse des Random Forest Klassifikators zu übertreffen. Die Nachteile der neuronalen Netze wie höhere Komplexität und fehlende Interpretierbarkeit werden somit nicht durch bessere Ergebnisse aufgewogen. Damit fiel die Entscheidung darauf, mehr Zeit in das Testen verschiedener Arten von Algorithmen, die auf Entscheidungsbäumen basieren, zu investieren.

5.3 Boosting Algorithmen

Wir haben mit Yggdrasil Decision Forests (YDF), XBG, lightGBM und CatBoost vier state-of-the-art Boosting Algorithmen ausprobiert, die sowohl methodisch, von der Rechengeschwindigkeit, als auch von den Parametern sehr ähnlich sind. So kann man beispielsweise den gleichen Code verwenden, um sie anzusprechen, da die Parameter die gleichen Namen haben (z.B. 'feature_importance') und man lediglich den Klassifkator austauschen muss. Das Hyperparameter-Tuning hat nur einen geringen Effekt. Lediglich bei CatBoost konnte durch die Anpassung der Lerngeschwindigkeit

(*learning-rate*) eine deutliche Verbesserung erzielt werden. CatBoost zeigt sowohl bei der Kreuzvalidierung als auch auf dem Testdatenatz die besten Ergebnisse. Ein Ensemble-Learner aus diesen drei Boosting-Algorithmen hat keine weitere Verbesserung bewirkt. Es sei noch darauf hingewiesen, dass wir als Optimierungskriterium AUC gewählt haben. Da das übliche Maß, die *accuracy*, für unausgeglichene Datesätze verzerrt sein kann, trägt die Wahl des Optimierungskritrerium AUC (area under the curve) dem Rechnung. Kurz gesagt wird bei AUC die Anzahl der False-Positives (FP) und der False-Negatives (FN) optimiert, während bei *accuracy* die Anzahl der True-Positives (TP) und True-Negatives (TN) maximiert wird, was in einem unausgewogenen Datensatz jedoch durch die weit größere Klasse verzerrt sein kann.

5.4 Sonstige Versuche

Wie zuvor beschrieben, wurden aufgrund der einfachen Umsetzung in Python eine Reihe von Algorithmen ohne Hyperparameter-Optimierung und Feature Engineering getestet, deren Ergebnisse wir hier nicht näher dargestellt haben, da deren Leistungen, im Vergleich mit den oben beschriebenen Algorithmen, deutlich schlechter ausfielen. Der Vollständigkeit halber seien nachfolgend die getesten Algorithmen genannt: Einfacher Entscheidungsbaum, K-Nearest-Neighbors, Support Vector Machine, Logistische Regression, Naiver Bayes Klassifikator, AdaBoost Klassifikator, Cart-Tree, mehrere neuronale Netze (von simplen feed-forward Netzen bis zu wide, deep, and cross Netzen). Des Weiteren wurde das Azure Automatisiertes maschinelles Lernen – AutoML Framework ausprobiert, was aber ebenfalls keine Verbesserung zeigte.

6 Evaluation

6.1 Bestimmung der Bewertungskriterien

Grundsätzlich gibt es für Klassifikatoren verschiedene Qualitätskriterien. Diese lassen sich am einfachsten anhand einer Konfusionsmatrix, wie sie in Tabelle 1 dargestellt ist, erläutern.

		Prediction	
		Negative	**Positive**
Truth	**Negative**	TN	FP
	Positive	FN	TP

Tabelle 1: Konfusionsmatrix

6.1.1 Genauigkeit (accuracy)

Das übliche Qualitätskriterium für einen binären Klassifikator ist die Genauigkeit (*accuracy*), welches die Anzahl der richtig klassifizierten Datenpunkte (TP + TN) durch die Anzahl aller Datenpunkte dividiert:

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN} \tag{1}$$

Da es sich aber um einen unausgewogenen Datensatz handelt und wir uns, wie es meistens der Fall ist, mit der richtigen Klassifikation der kleineren Klasse (*fraud (1)*) beschäftigen, ist eine möglichst hohe *accuracy* nicht unsere Zielsetzung. In einem unausgewogenen Datensatz gilt diese als verzerrt, da sie durch die richtige Klassifikation der größeren Klasse, die uns aber nur sekundär interessiert, bestimmt wird. Insbesondere würde ein Klassifikator, der immer die Klasse *normal (0)* vorhersagt, bereits eine hohe Genauigkeit erzielen.

6.1.2 Eigenes Bewertungskriterium

Die Bewertungskriterien wurden durch die Auftraggeberin in Form von monetären Werten vorgegeben. Wenn man in der Konfusionsmatrix in Tabelle 1 die regulären Einkäufe ($label = 0$) als negativ und die fehlerhaften Transaktionen ($label = 1$) als positiv definiert, so möchte die Auftraggeberin

die Anzahl der korrekt identifizierten fehlerhaften Transaktionen (TP) maximieren bei gleichzeitig möglichst geringer Fehlerzahl (FN und FP). Jede korrekt klassifizierte fehlerhafte Transaktion (TP) hat für die Auftraggeberin dabei einen Wert von 5 Euro.

Als Besonderheit kommt hinzu, dass die beiden möglichen Fehler nicht gleich von der Wertigkeit sind. Eine nicht erkannte fehlerhafte Transaktion (FN) hat einen Wert von -5 Euro. Ein zu Unrecht als fehlerhaft klassifizierter Einkauf (FP) hat dagegen einen Wert von -25 Euro, da Kunden verärgert sein könnten, dass sie des Betrugs verdächtigt wurden. Damit lassen sich anhand des laut Gleichung 2 berechneten Euro-Werts verschiedene Klassifikationsalgorithmen messen.

$$Wert = 5 * TP - 25 * FP - 5 * FN. \tag{2}$$

6.2 Mehrwertschätzung

Tabelle 2 zeigt das Ergebnis des CatBoost Algorithmus für den zuvor bestimmten Testdatensatz (das Modell wurde auf 80% der Daten trainiert). Innerhalb des Testdatensatzes gibt es 908 (550 FN + 358 TN) fehlerhafte Transaktionen. Ohne einen Algorithmus würde der Verlust der Auftraggeberin demnach bei -4540 Euro liegen (-5 Euro * 908). Mit unserem Algorithmus reduziert sich dieser Verlust auf -1235 Euro (5 Euro * 358 - 5 Euro * 550 - 25 Euro * 11), was einem Mehrwert von 3305 Euro entspricht (4540 Euro - 1235 Euro).

Anhand dieser Beispielrechnung lässt sich erklären, wie man anhand der Bewertungskriterien aus Gleichung 2 auch den Mehrwert des jeweiligen Algorithmus berechnen kann.

		Prediction	
		Negative	**Positive**
Truth	**Negative**	20004	11
	Positive	550	358

Tabelle 2: Konfusionsmatrix für CatBoost auf dem Testdatensatz.

6.3 Überprüfung der Umsetzbarkeit und Benchmarking

Der Algorithmus CatBoost liefert, wie bereits zuvor beschrieben, die besten Ergebnisse auf dem vorliegenden Datensatz. Der Algorithmus gehört zur Klasse der Boosting Algorithmen, die bereits 1997 mit AdaBoost etabliert wurden und gut erforscht sind. Das Python Paket für den Algorithmus ist frei verfügbar und wird von der Firma Yandex gepflegt und weiterentwickelt. Eine gute Dokumentation der Schnittstellen und Parameter ist vorhanden und steht unter `https://catboost.ai/` zur Verfügung. Das Trainieren des Algorithmus auf neuen Daten ist auf einem „normalen“ Laptop in unter drei Minuten möglich. Eine Klassifikation eines Einkaufs durch das trainierte Modell erfolgt augenblicklich. Wir können somit keine eventuellen Probleme bei der Umsetzbarkeit des Modells erkennen.

Das Benchmarking des Algorithmus hat gegen den Random Forest Klassifikator stattgefunden. Dieser hat sich zunächst als der beste Algorithmus aus dem scikit-learn-Universum (KNN, SVM, Logistic Regression) herauskristallisiert. Die Ergebnisse von CatBoost auf dem vorliegenden Datensatz konnten die des Random Forest Klassifikators aber nochmals deutlich verbessern. Aufgrund der geringeren Trainingszeit im Vergleich zu Random Forest und der vergleichbaren Interpretierbarkeit mittels der Visualisierung des Einflusses der einzelnen Attribute über *feature_ importance*, sprechen sämtliche Entscheidungskriterien für die Wahl von CatBoost.

6.4 Ergebnisevaluation

Bei der Evaluation haben wir uns ausschließlich auf das von der Auftraggeberin vorgegebene Bewertungskriterium (vgl. Gleichung 2) gestützt. Das heißt, ein Algorithmus a, dessen Klassifikationsergebnis einen höheren Wert erzielt als Algorithmus b, gilt als der bessere Algorithmus in diesem Kontext.

Das Training erfolgte auf 80% der vorliegenden Daten und die eigentliche Evaluation des Algorithmus fand sowohl mittels Kreuzvalidierung als auch anhand der Ergebnisse auf den verbleibenden 20% der Daten statt, die der jeweilige Algorithmus bis zu diesem Zeitpunkt nicht kannte. Durch die

einheitlichen Festlegung des Zufallsgenerators mit *random_state* = 42, ist dieser Datensatz für alle Algorithmen identisch und ihre Ergebnisse somit direkt vergleichbar. Der Testdatensatz enthält zudem eine angemessene Anzahl an fehlerhaften Transaktionen durch das Setzen des Parameters *stratify*.

6.5 Performance-Tests

Unsere Analyse erfolgt ausschließlich in Python, was sich durch seine Open-Source-Charakteristik auszeichnet. Es gibt eine große Entwickler-Community, so dass sowohl die Sprache selbst als auch die von uns verwendeten Pakete kostenlos und auf allen üblichen Computersystemen verfügbar sind. Der von uns vorgeschlagene Algorithmus lässt sich mit Hilfe der Installation des Python-Pakets CatBoost mittels *pip install catboost* auf jedem Betriebssystem verwenden. Die zugrundeliegenden Daten können in jedem Format vorliegen und werden dann mittels weniger Python-Befehle in das sehr übliche Pandas-DataFrame-Format konvertiert. Auf diese Weise können die Daten leicht vom Algorithmus konsumiert werden. In unserem Fall stehen die Daten im CSV-Format zur Verfügung. Das folgende Code-Beispiel (Abbildung 2) zeigt, wie ein Python-Paket eingebunden wird und wie man die CSV-Datei (train.csv") als DataFrame einlesen kann. Unserer Meinung nach ist die Anwendung von CatBoost mit diesem set-up auf jeglicher Computer-Hardware sichergestellt.

```
# import package
import pandas as pd
# read data set
df = pd.read_csv("train.csv")
```

Abbildung 2: Einbindung eines Python-Pakets und Einlesen eines CSV-Files.

7 Analyseergebnisse

Die Ergebnisse des Modells entsprechen für manche Attribute den Erwartungen. So hatten wir zum Beispiel bei der explorativen Datenanalyse festgestellt, dass der Tag und die Stunde des Einkaufs einen Einfluss auf die Wahrscheinlichkeit einer fehlerhaften Transaktion haben (höhere Wahrscheinlichkeit zwischen 16:00 und 20:00 Uhr und an den Tagen von Donnerstag bis Samstag). Dies spiegelt sich in den Ergebnissen wider, da sowohl der Tag als auch die Stunde einen signifikanten Erklärungsbeitrag liefern.

Unter den Attributen, die den höchsten Erklärungbeitrag liefern, befinden sich für uns ausschließlich Variablen mit denen wir intuitiv oder aufgrund der vorhergehenden Analyse gerechnet haben, wie zum Beispiel *item_per_EUR*, *hour* oder *cash_epoch*, wobei letzteres Attribut die Einführung der Bargeldzahlung anzeigt.

Für andere Attribute wie zum Beispiel *convenience* oder *dry* konnte im Vorfeld nicht dieselbe Erwartung formuliert werden. Zur Veranschaulichung ist in Abbildung 3 die Wichtigkeit aller Attribute dargestellt.

Wie schon genannt, war CatBoost der beste getestete Algorithmus. Der final für das Training verwendete Datensatz enthält die nachfolgend aufgeführten Attribute:

- grand_total
- n_items
- total_checkout_time
- line_voids
- alcohol
- fruit and vegetables
- snack
- dry
- convenience
- bakery

- household
- energy_drink
- weekday
- hour
- payment_medium_card
- payment_medium_cash
- label
- cash_epoch
- item_per_time
- item_per_EUR
- void_per_time

Als beste Hyperparameter-Einstellung für CatBoost ergab sich:

- learning_rate: 0.03
- objective: Logloss
- random_state: 42
- depth: 6
- boosting_type: Ordered
- loss_function: AUC
- verbose: False

Die ausführlichen CatBoost-Ergebnisse des Feature Engineerings und des Hyperparameter-Tunings sowie die Ergebnisse des besten Modells können dem Jupyter-Notebook entnommen werden, das separat zur Verfügung gestellt wird. Zudem wurde das Modell auf dem von der Auftraggeberin zur Verfügung gestellten Testdatensatz aus 2019 angewendet, um Vorhersagen für alle Transaktionen zu treffen. Das Ergebnis wird als CSV-Datei zur Verfügung gestellt, in der die Werte 0 (normale Transaktionen) und 1 (fehlerhafte Transaktionen) für jede Transaktion enthalten sind. Dabei wurden auf den Testdaten aus 2019 insgesamt 5082 fehlerhafte Transaktionen erkannt.

7.1 Darstellungsform und Ergebnistyp

Die eigentliche Aussage des Algorithmus ist eine Klassifikation eines Einkaufs in *fehlerhaft* ($label = 1$) oder *normal* ($label = 0$). Bei einer fehlerhaften Transaktion würde ein Mitarbeiter der Auftraggeberin den Einkauf nachkontrollieren. Damit ist die Darstellungsform lediglich eine binäre Variable.

7.2 Grenzen des Modells

Die Interpretierbarkeit des Modells ist nur indirekt möglich. Aber über den Parameter *feature_ importance* kann man einen guten Einblick gewinnen, welche Attribute einen höheren Einfluss auf die jeweilige Klassifikation haben.
Eine weitere interessante Möglichkeit, die Entscheidung des Modells für einzelne Transaktionen nachzuvollziehen, bieten die sogenannten *SHAP Values* (**SH**apley **A**dditive ex**P**lanations). Anhand dieser lassen sich die entscheidenden Attribute für die jeweils getroffene Entscheidung nachvollziehen.

Abbildung 4 zeigt eine Transaktion, die das Modell als *normal* klassifiziert hat. Die SHAP Values zeigen, welche Attribute die Wahrscheinlichkeit für *fraud* nach oben (in rot) und nach unten (in blau) beeinflussen.

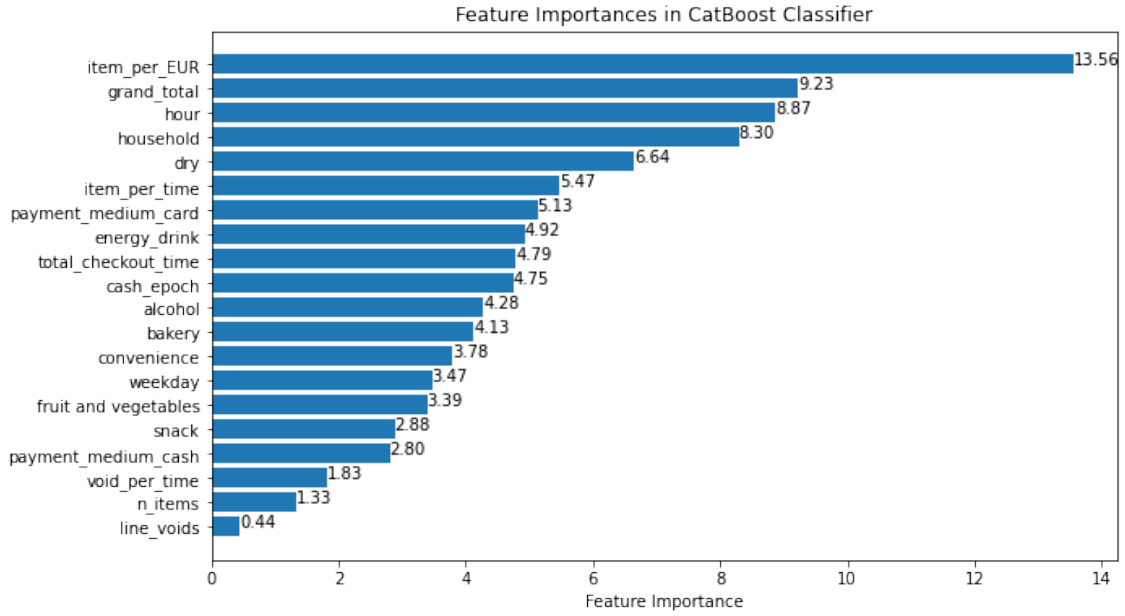

Abbildung 3: Feature Importance für CatBoost

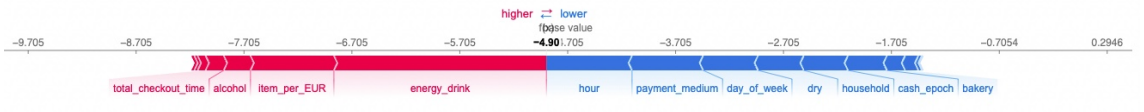

Abbildung 4: SHAP Values für Klassifikation *normal*

Zum Vergleich zeigt Abbildung 5 die SHAP Values für eine Transaktion, die das Modell als *fraud* klassifiziert hat. Hier kann man erkennen, dass der Kauf von Produkten der Kategorien *household* und *energy_ drink*, die Art der Bezahlung und der Zeitpunkt des Einkaufs neben anderen Faktoren das angezeigte Maß, welches proportional zur Wahrscheinlichkeit für *fraud* ist, in den positiven Bereich gedrückt haben.

Wir würden vorschlagen, den Mitarbeitern, die die Nachkontrolle durchführen, diese Werte mitzugeben, damit das Modell etwas transparenter und nachvollziehbar wird.

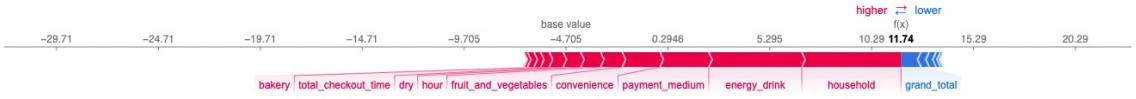

Abbildung 5: SHAP Values für Klassifikation *fraud*