Klassifikationsmodell – Bericht

Yannic Lais, Marvin von Rappard, Luca Mazzotta

January 16, 2023

Abstract

In diesem Bericht finden Sie Informationen zum Prozess der Erstellung eines Klassifikationsmodells, um den Typ verschiedener Schweizer Immobilien hervorzusagen. Die Daten wurden gesäubert und skaliert. Zudem wurden verschiedene Feature Engineering Methoden für eine bessere Vorhersage verwendet. Weitere Informationen sind in folgendem Github Repository zu finden: https://github.com/marvinvr/fhnw-cml1

Contents

1 Einführung

In diesem Bericht finden Sie Informationen zum Prozess der Erstellung eines Klassifikationsmodells, um den Typ verschiedener Immobilien auf dem Schweizer Markt vorherzusagen. Die Daten wurden gesäubert und skaliert. Zudem wurden verschiedene Feature Engineering Methoden für eine bessere Vorhersage verwendet. Anschliessend wurden vier verschiedene Modelle trainiert und miteinander verglichen.

2 Vorgehen

Mittels Klassifikationsmodellen soll der Typ verschiedener Schweizer Immobilien vorhergesagt werden. Es wurden dabei Daten von dem Zeitraum August bis November im Jahre 2022 verwendet.
Die Daten wurden als Erstes gesäubert und analysiert. Anschliessend wurden diese für die Modelle vorbereitet, was eine Skalierung sowie diverse Feature Engineering Methoden beinhaltet. Mit den vorbereiteten Daten wurden verschiedene Modelle trainiert und getestet.

2.1 Data Wrangling

Beim Data Wrangling geht es darum, den Datensatz so vorzubereiten, dass die Daten verwendet werden können, um ein maschinelles Modell zu trainieren.
Folgende Schritte wurden durchgeführt:

- Spalten wurden zusammengefügt und umbenannt.
- Aus kategorialen Features wurden Dummy-Variablen erstellt.
- Redundante Spalten wurden entfernt.
- Absurde Daten wurden gelöscht (z. B. Immobilien für 1.- CHF).
- Fehlende Daten wurden imputiert.

Mehr Informationen zu dem Vorgehen bei dem Imputieren der falschen oder fehlenden Daten sind im Kapitel Feature Engineering zu finden.

2.2 Skalieren

Die Skalierung der Daten bezieht sich auf die Anpassung der Wertebereiche der einzelnen Features im Datensatz, um alle Features auf einen ähnlichen Bereich zu bringen. Da einige Algorithmen davon beeinflusst werden können, wenn Features unterschiedliche Wertebereiche haben.

Wenn ein Feature Werte im Bereich von 0 bis 1 hat und ein anderes Feature Werte im Bereich von 0 bis 10000 hat, besteht die Möglichkeit, dass dieses Feature vom Algorithmus als wichtiger eingestuft wird, als es tatsächlich ist. Dies kann die Modellvorhersage beeinflussen und zusätzlich dazu führen, dass das Modell langsamer trainiert wird.

Durch die Skalierung der Features auf einen ähnlichen Bereich wird also sichergestellt, dass jedes Feature den gleichen Einfluss auf das Modell hat und dass der Algorithmus die Daten richtig interpretiert. Es gibt verschiedene Möglichkeiten, die Daten zu skalieren, wie Normalisieren oder Standardisieren. Beim Normalisieren skaliert man die Daten auf den Bereich von 0 bis 1, während beim Standardisieren die Daten so skaliert werden, dass diese eine Standardnormalverteilung haben, mit einem Mittelwert von 0 und einer Standardabweichung von 1.

Insgesamt ist die Skalierung von Daten ein wichtiger Schritt bei der Vorbereitung von Daten für ein Machine Learning Modell, da es hilft, die Leistung und die Genauigkeit des Modells zu verbessern und sicherzustellen, dass der Algorithmus die Daten richtig interpretiert.

Für alle Modelle wurde der Datensatz mit dem Min-Max-Scaler skaliert. Da der Datensatz des Kaggle Wettbewerbs andere minimale und maximale Werte hat, wurde darauf geachtet, dass auch die Daten des Wettbewerbs mit demselben Scaler skaliert wurden.

2.2.1 Min-Max-Scaler

Der Min-Max-Scaler ist eine Methode der Skalierung von Daten, die verwendet wird, um die Features auf einen festgelegten Wertebereich zu bringen. Dieser Wertebereich ist in der Regel [0, 1], aber kann auch auf einen anderen Bereich festgelegt werden.

Die Methode funktioniert, indem der Min-Max-Scaler die Werte jedes Features mit einer einfachen Formel umrechnet:

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{1}$$

X_{scaled} ist der skalierte Wert des Features x, X_{min} ist der kleinste Wert des Features und X_{max} ist der grösste Wert des Features. Durch diese Skalierung werden alle Werte des Features auf den Bereich von 0 bis 1 skaliert.

Der Min-Max-Scaler ist eine einfache und schnelle Methode der Skalierung und eignet sich besonders für Daten, bei denen es keine Ausreisser gibt und wenn die Daten nicht normalverteilt sind.

2.2.2 Robust-Scaler

Der Robust-Scaler funktioniert ähnlich wie der Min-Max-Scaler, aber anstatt die Daten auf eine feste Skala von 0 bis 1 zu bringen, normalisiert er diese auf eine Skala von -1 bis 1. Im Gegensatz zum Min-Max Scaler, der empfindlich auf Ausreisser reagieren kann, nutzt der Robust Scaler die Quartile der Daten.

$$X_{norm} = \frac{X - Q1(X)}{Q3(X) - Q1(X)} \tag{2}$$

Hierbei wird jeder Wert in einem Feature durch den Abstand zu den Quartilen Q1 und Q3 normalisiert. Durch dass das in der Formel mit den Quartilen gearbeitet wird, ist der Robust-Scaler robust gegenüber Ausreissern.

2.2.3 Beste Skalierung

Nachdem die Leistungen beider Methoden verglichen wurden, wurde sich schliesslich für die Min-Max Skalierung entschieden, da diese die besten Ergebnisse geliefert hat im Sinne der Zielmetrik ("Weighted f1-Score").

2.3 Feature Engineering

Feature Engineering ist wichtig beim Training von Machine Learning Modellen, weil es dazu beiträgt, dass die Modelle sich auf die relevanten Merkmale der Daten konzentrieren und dadurch besser generalisieren können. Es ermöglicht auch, die Qualität der Daten zu verbessern, indem es z. B. fehlende oder fehlerhafte Daten ausfüllt oder entfernt.

Ohne Feature Engineering kann es sein, dass das Modell nicht in der Lage ist, die relevanten Muster in den Daten zu erkennen, was zu einer schlechteren Leistung führt. Feature Engineering erfordert jedoch oft Zeit und Fachwissen und kann eine iterative Aufgabe sein, um die besten Merkmale zu identifizieren und zu nutzen. Deshalb fokussiert sich das Feature Engineering auf wenige, jedoch ausführliche Methoden.

2.3.1 Imputieren von Daten

Imputieren von Daten bezieht sich auf den Prozess, fehlende Daten in einem Datensatz aufzufüllen. Dies kann notwendig sein, da viele Machine Learning Modelle nicht in der Lage sind, mit fehlenden Daten umzugehen und daher eine Vorverarbeitung der Daten erfordern.

Für das Imputieren der Daten der beiden Datensätze wurde ein KNN Imputer mit den 15 nächsten Nachbarn pro Kategorie des Immobilien-Typs gewählt.

KNN Imputer:
Der KNN Imputer (K-Nearest Neighbors Imputer) ist eine Methode der Datenimputierung. Es nutzt die Beziehungen zwischen den Features von nahe beieinander liegenden Beobachtungen, um fehlende Werte vorherzusagen.

Der KNN Imputer ist deshalb eine gute Methode, da sie vorwiegend auf die Ähnlichkeit mit anderen Beobachtungen achtet. Je nach Anzahl der Nachbarn, die man dem Imputer mitgibt, können sich die Ergebnisse ändern, deshalb ist es wichtig die richtige Anzahl an Nachbarn herauszufinden. Eine mögliche Methode ist es einige Werte zu testen und den Wert zu wählen, der die besten Ergebnisse liefert. Die Daten wurden am besten imputiert, indem der KNN Imputer 15 Nachbarn beachtete.

Simple-Imputer:
Der Simple-Imputer ist eine einfache und schnelle Möglichkeit zum Imputieren von fehlenden Werten in einem Datensatz. Dabei gibt es verschiedene Möglichkeiten, mit dem Mittelwert, Median oder Modus. Die Daten wurden schlussendlich mit dem Median imputiert.

2.3.2 Polynomial Expansion

Die Polynomial-Expansion ist eine Methode des Feature-Engineering, bei der neue Features durch die Potenzen der bestehenden Features erstellt werden. Dies kann dazu beitragen, Interaktionen zwischen Features zu erfassen, die das Modell sonst nicht erkennen würde.
Alle numerischen Features wurden mit den Hochzahlen [2,3,4,5,6] ergänzt.

3 Metriken

3.1 F1-Score

Die F1 Metrik findet ein Gleichgewicht zwischen Precision und Recall, denn beide werden in einem harmonischen Durchschnitt zusammengerechnet und somit ein starkes Ungleichgewicht zwischen Precision und Recall bestraft. Der F1-Score beachtet somit die Verteilung der Daten und gibt bei ungleich verteilten Daten eine bessere Einschätzung zurück.

$$F1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{3}$$

Es gibt verschiedene Arten des F1-Scores, welche je nach Verteilung der Kategorien besser oder schlechter geeignet sind.

- Micro-average F1-Score: Berechnet den F1-Score indem er die Anzahl der true positive, false positive und false negative über alle Klassen hinweg betrachtet. Diese Metrik ist geeignet, wenn die Klassen ungleich verteilt sind und die Genauigkeit des Modells über alle Klassen hinweg betrachtet werden soll.
- Macro-average F1-Score: Berechnet den F1-Score indem es die Anzahl der true positive, false positive und false negative über alle Klassen hinweg betrachtet. Diese Metrik ist geeignet, wenn der Score zu jeder Klasse unabhängig betrachtet werden soll.
- Weighted-average F1-Score: Berechnet den F1-Score indem es den F1-Score für jede Klasse berechnet und den Durchschnitt dieses F1-Scores nimmt, wobei die Klassen ungleich gewichtet werden. Diese Metrik ist geeignet, wenn die Klasse in Beziehung zu ihrer Häufigkeit betrachtet werden soll.

Für das Klassifikationsmodell wurde der Weighted-average F1-Score verwendet, da die Anzahl an Klassen sehr ungleich verteilt war. So kann garantiert werden, dass das beste Modell auch Klassen klassifiziert, von welcher nur wenige Datenpunkte vorhanden sind.

3.1.1 Recall / Sensitivity

Recall / Sensitivity berechnet, wie viele korrekt positiv aus allen Daten gelabelt wurden.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{4}$$

3.1.2 Precision

Precision berechnet, wie viele korrekt positiv aus allen positiven Daten erkannt wurden.

$$\text{Precision } = \frac{\text{TP}}{\text{TP } + FP} \tag{5}$$

Es gibt noch weitere Metriken, auf welche an dieser Stelle jedoch nicht weiter eingegangen wird.

4 Klassifikationsmodelle

Um den Typ verschiedener Schweizer Immobilien hervorzusagen, wurden vier verschiedene Modelle trainiert und getestet.
In dem folgenden Text werden die verwendeten Modelle erklärt. Die Modelle wurden mit der Python-Bibliothek scikit-learn erstellt.
Zum trainieren der Modelle wurde mit Cross-Validation und Gridsearch gearbeitet. Diese Methoden ermöglichen das Testen verschiedener Parameter auf jeweils unterschiedlichen Test- und Trainingsdaten.

4.1 KNN Classifier

Ein KNN (k-Nearest Neighbors) Classifier ist ein einfaches Lernverfahren, welches auf der Annahme basiert, dass ähnliche Beispiele wahrscheinlich derselben Klasse zugeordnet werden. Es arbeitet, indem es die k-nächsten Beispiele eines gegebenen Eingabebeispiels im Trainingsset betrachtet und die Mehrheit der Klassen der k-nächsten Nachbarn als die Vorhersage des gegebenen Eingabebeispiels verwendet.

Der Prozess beginnt mit dem Speichern aller Trainingsbeispiele und ihrer zugehörigen Klassenlabels. Wenn ein neues Eingabebeispiel gegeben wird, berechnet das Modell die Distanz zwischen diesem Eingabebeispiel und allen Trainingsbeispielen. Es wählt dann die k-nächsten Trainingsbeispiele basierend auf diesen Distanzen aus. Danach bestimmt das Modell die Mehrheit der Klassenlabels unter den k-nächsten Nachbarn und verwendet diese Mehrheit als die Vorhersage des Eingabebeispiels.

Der Hyperparameter k gibt an, wie viele Nachbarn betrachtet werden sollen. Ein höherer Wert von k führt dazu, dass das Modell stabiler gegenüber Ausreissern ist, aber es kann auch dazu führen, dass das Modell ungenauer wird.

4.2 Random Forest Classifier

Ein Random Forest Classifier ist ein Ensemble-Lernverfahren, das aus mehreren Entscheidungsbaum-Modellen besteht. Es arbeitet, indem es zufällig ausgewählte Merkmale und Beispiele verwendet, um mehrere Entscheidungsbäume zu erstellen und diese schliesslich zu kombinieren, um eine endgültige Vorhersage zu treffen.

Der Prozess beginnt mit dem Erstellen von Entscheidungsbäumen. Dies geschieht, indem zufällig ausgewählte Merkmale und Beispiele verwendet werden, um jeden Baum zu trainieren. Dies führt dazu, dass jeder Baum unterschiedliche Merkmale und Beispiele verwendet und dadurch unterschiedliche Entscheidungen trifft.

Danach werden die Vorhersagen aller Bäume zusammengefügt, um die endgültige Vorhersage des Random Forest Classifier zu erhalten. Dies geschieht in der Regel durch Abstimmung, bei der jeder Baum abstimmt, welche Klasse er für die gegebene Eingabe vorhersagt. Die Klasse, die die meisten Stimmen erhält, wird als die endgültige Vorhersage ausgewählt.

Durch die Verwendung von mehreren Entscheidungsbaum-Modellen und die Verwendung von zufällig ausgewählten Merkmalen und Beispielen, kann der Random Forest Classifier sehr robuste und zuverlässige Modelle erstellen, die gegenüber Überanpassung resistent sind.

4.3 Gradient Boosting Classifier

Ein Gradient Boosting Classifier ist ein Ensemble-Lernverfahren, ähnlich wie ein Random Forest Classifier, das aus mehreren einfachen Entscheidungsbaum-Modellen besteht. Es arbeitet, indem es in jedem Schritt einen neuen Baum hinzufügt, der die Fehler des vorherigen Baums korrigiert.

Der Prozess beginnt mit dem Training eines einfachen Baumes auf den Trainingsdaten. Dann berechnet das Modell die Abweichungen (auch genannt Residuen) zwischen den vorhergesagten und den tatsächlichen Werten für jedes Beispiel im Trainingsdatenset.

Anschliessend trainiert das Modell einen weiteren Baum, aber diesesmal auf den Residuen anstatt auf den tatsächlichen Werten. Dieser Baum versucht also, die Abweichungen des ersten Baumes zu korrigieren.

Dieser Prozess wiederholt sich und fügt immer mehr Bäume hinzu, bis das Modell eine zufriedenstellende Leistung erbringt oder eine bestimmte Anzahl von Bäumen erreicht ist. Schliesslich werden die Vorhersagen aller Bäume zusammengefügt, um die endgültige Vorhersage des Gradient Boosting Classifier zu erhalten. Dies geschieht in der Regel durch einfaches Aufaddieren der Vorhersagen, wobei jeder Baum ein bestimmtes Gewicht hat.

4.4 MLP Classifier

Ein MLP Classifier ist ein künstliches neuronales Netzwerk, das für die Klassifizierung verwendet wird. Es besteht aus einer Schicht von Eingabe-Neuronen, einer oder mehreren Schichten versteckter Neuronen und einer Schicht von Ausgabe-Neuronen. Der Prozess beginnt mit dem Eingeben von Daten in die Eingabeschicht. Die Eingabedaten werden dann durch jedes Neuron in der ersten versteckten Schicht gesendet, wo sie mit einem Gewicht multipliziert werden. Diese gewichteten Daten werden dann mit einem Bias-Wert addiert und durch eine Aktivierungsfunktion geleitet. Die Ausgabe jedes Neurons in der letzten versteckten Schicht repräsentiert dann die Wahrscheinlichkeit, dass die Eingabe in die entsprechende Klasse eingestuft wird.

Das Modell wird trainiert, indem die Gewichte angepasst werden, um die Vorhersagen des Modells mit den tatsächlichen Zielen zu minimieren. Durch die Verwendung von mehreren Schichten versteckter Neuronen und die Anpassung der Gewichte kann das MLP Modell komplexe Beziehungen in den Eingabedaten erlernen und seine Vorhersagen verbessern.

5 Resultate

Das beste entwickelte Modell basiert auf einer Random Forest Methode. Damit wurde ein **gewichteter F1-Score von 0,76** erzielt, was bedeutet, dass das Modell gewisse Erfolge in der Klassifizierung von Daten hat, jedoch gibt es noch Raum für Verbesserungen. Es ist wichtig zu beachten, dass dieser Score nur ein Indikator für die Leistung des Modells ist und es möglicherweise Schwächen in bestimmten Bereichen hat. In den folgenden Punkten werden die Stärken und Schwächen des Modells genauer analysiert.

5.1 Confusion Matrix

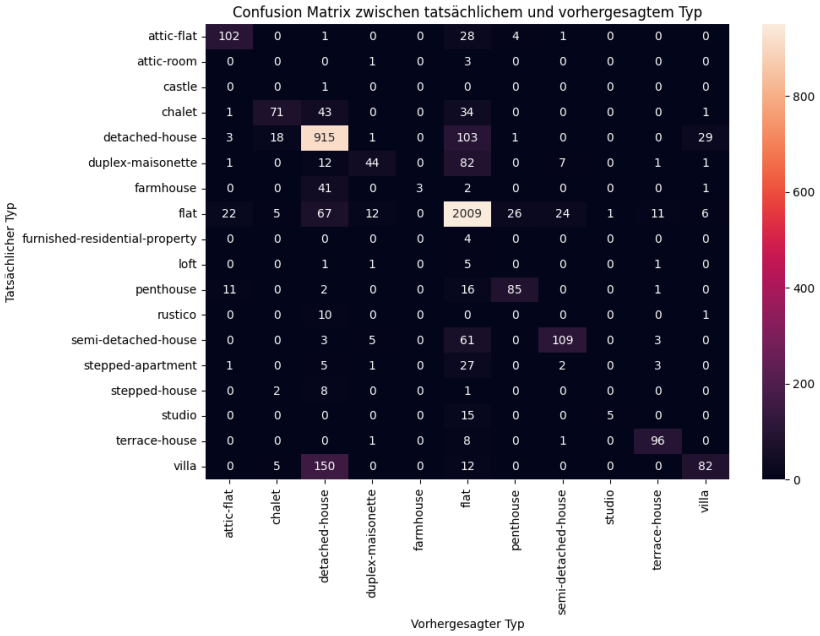

Figure 1: Confusion Matrix der Resultate aus dem Random Forest Classifier

In der Confusion Matrix ist zu sehen, welche Typen gewisse Objekte tatsächlich haben und was das Modell vorhersagt. Zu sehen ist, dass sich das Modell auf deutlich weniger verschiedene Typen beschränkt als tatsächlich vorhanden sind. Das hängt damit zusammen, dass die meisten Typen nur sehr wenige Datenpunkte haben und daher vom Modell nicht extrem gut verstanden werden. Offensichtlich ist jedoch, dass die beiden am meisten vorkommenden Typen, «detached-house» und «flat» grösstenteils korrekt vorhergesagt werden, was bedeutet, dass das Modell die Essenz des Datensatzes gut verstanden hat.

5.2 Variable Importance

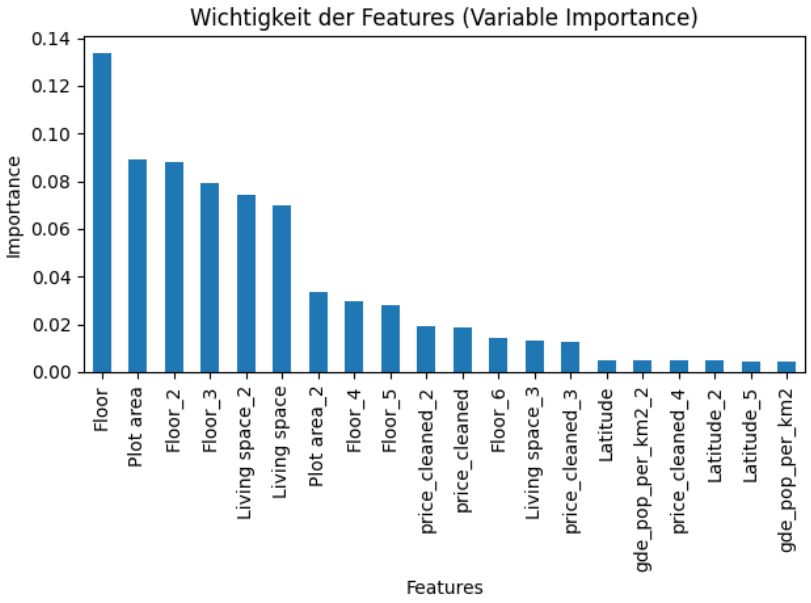

Figure 2: Variable Importance der Resultate aus dem Random Forest Classifier

In der Variable Importance, welche aus dem Modell ausgelesen wurde, sehen wir, dass ≪Floor≫ und ≪Plot area≫ am meisten Einfluss auf das Resultat des Modells haben. Ausserdem ist zu erkennen, dass die Polynomial-Erweiterung des Datensatzes ein wichtiger Schritt in der Datenverarbeitung war. Das Modell scheint eine Menge Informationen aus den hochgerechneten Werten herauslesen zu können.

5.3 Performance auf neue Daten

Damit das Klassifikationsmodell auch auf neuen Daten gute Resultate erzielt, gibt es einige Punkte, die beachtet wurden.

5.3.1 Cross Validation

Mithilfe von Cross Validation kann das Modell mehrmals mithilfe von Daten geprüft werden, mit welchen es noch nie in Berührung kam. So wird eine Überanpassung verhindert und das Modell wird auch in Zukunft mit neuen Daten gute Resultate erzielen.

5.3.2 Grosser Testdatensatz

Durch das Testen mit einem grossen Testdatensatz kann eine Überanpassung verhindert werden. Die Testdaten beinhalteten 20 Prozent des gesamten Datensatzes.

5.4 Rückblick

Aus dem gewichteten F1-Score von 0,76 und der Analyse des Modells ist zu sehen, dass die Resultate des Random Forest Classifiers keineswegs perfekt sind. Es zeigt jedoch auch, dass das Modell einen Grossteil des Datensatzes bereits sehr gut versteht und somit sicherlich ein guter Ansatz für das Problem ist.