Bestmögliches Regressionsmodell – Bericht

Yannic Lais, Marvin von Rappard, Luca Mazzotta

January 16, 2023

Abstract

In diesem Bericht finden Sie Informationen zum Prozess der Erstellung eines Regressionsmodells, um die Preise verschiedener Immobilien auf dem Schweizer Markt vorherzusagen. Die Daten wurden gesäubert und skaliert. Zudem wurden verschieden 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

Mittels Regressionsmodellen soll der Preis verschiedener Schweizer Immobilien vorhergesagt werden. Es wurden dabei Daten im 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 diverse Modelle trainiert und getestet.

2 Vorgehen

Um ein bestmögliches Regressionsmodell zu erstellen, wurde der Datensatz analysiert und gesäubert. Nach einer ausreichenden Datenanalyse wurden die Daten skaliert und mit verschiedenster Feature Engineering Methoden bearbeitet.
Anschliessend wurden mehrere Regressionsmodelle miteinander verglichen.

2.1 Data Wrangling

Es wurden zwei verschiedene Daten Wrangling Notebooks erstellt. Zuerst wurde ein Notebook für den Trainingsdatensatz erstellt und anschliessend ein weiteres Notebook für die Wettbewerbsdaten (Kaggle - Contest).
Beim Daten Wrangling geht es darum, den Datensatz so vorzubereiten, dass die Daten verwendet werden können, um ein Machine Learning 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 zum Vorgehen beim 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 in einem Bereich von 0 bis 1 hat und ein anderes Feature Werte in einem 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 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, welcher 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 (Mean absolute percentage error, MAPE).

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 Features 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 auffüllt 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 viel Zeit und Fachwissen und ist meistens eine iterative Aufgabe. Deshalb fokussiert sich das Feature Engineering auf wenige, jedoch ausführliche Methoden.

2.3.1 Imputieren von Daten

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

Für das Imputieren der Daten wurde ein KNN Imputer mit den 15 nächsten Nachbarn pro Kategorie des Immobilien-Typs gewählt. Für die Daten des Wettbewerbs wurde ebenfalls dieser Imputer benutzt. Anschliessend wurden die restlichen Daten, welche keiner Kategorie zugehörig waren, mithilfe eines Simple-Imputer imputiert.

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 Polynominale 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.

2.3.3 Feature Selektion

Feature Selektion wurde mithilfe eines Lasso-Regressionsmodells durchgeführt. Dadurch werden die weniger wichtigen Features des Modells eliminiert. Dies führt zu einer Reduktion der Anzahl der Features des Modells und kann dazu beitragen, Overfitting zu vermeiden und somit Vorhersagegenauigkeit zu verbessern.

Die Lasso-Regression kann durch die folgende Formel dargestellt werden:

$$\min_{\beta} \frac{1}{2n} \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{3}$$

Die erste Summe im Ausdruck ist die Summe der Quadratsummen der Fehler (RSS), während die zweite Summe die Summe der absoluten Koeffizienten (L1-Regularisierung) ist. Der Parameter λ kontrolliert die Stärke der Regularisierung und bestimmt, wie viele Koeffizienten auf null gesetzt werden. Je grösser λ ist, desto mehr Koeffizienten werden auf null gesetzt.

2.3.4 Variablen Transformation

Durch das Transformieren nicht normalverteilter Variablen zu normalverteilten Variablen, können bestimmte Modelle wie neuronale Netzwerke oder lineare Regressionsmodelle die Daten besser verarbeiten (ähnlich wie bei unskalierten Daten). Modelle wie Gradient Boosting Regressoren oder Random Forest Regressoren sind weniger empfindlich auf nicht normalverteilte Daten. Variablen sollten folgendermassen transformiert werden:

- Rechtsschiefe Verteilung – Logarithmisch, Wurzel
- Linksschiefe Verteilung – Exponentiell

Nach ausbleibendem Erfolg, die Modelle mit transformierten Variablen zu trainieren, wurde in diesem Fall auf das Transformieren verzichtet.

3 Regressionsmodelle

Um die abhängige Variable Preis vorherzusagen, wurden mehrere Regressionsmodelle trainiert und getestet. In den folgenden Texten werden die verwendeten Modelle erklärt.
Die Modelle wurden mit der Python-Bibliothek scikit-learn erstellt.
Beim Trainieren der Modelle wurde mit Gridsearch und Cross-Validation gearbeitet. Diese Methoden ermöglichen das Testen verschiedener Parameter auf jeweils unterschiedlichen Test- und Trainingsdaten.

3.1 Random Forest Regressor

Ein Random Forest Regressor ist ein Ensemble-Modell, das aus mehreren Entscheidungsbäumen besteht, welche gemeinsam das optimale Resultat erreichen. Der Algorithmus funktioniert, indem er zufällig ausgewählte Untergruppen der Datenpunkte und Feature verwendet, um mehrere Entscheidungsbäume zu erstellen, die unabhängig voneinander trainiert werden.

Die endgültige Vorhersage des Modells wird durch die Durchschnittsvorhersage aller Entscheidungsbäume berechnet. Da jeder Baum auf einer zufällig ausgewählten Teilmenge der Daten und Feature trainiert wird, sind die Bäume weniger anfällig für Überanpassung und haben tendenziell eine bessere allgemeine Leistung als ein einzelner Entscheidungsbaum.

Der Random Forest Regressor hat auch die Fähigkeit, die Bedeutung jedes Features für die Vorhersage zu messen, indem es die relative Häufigkeit der Verwendung des Features in den Entscheidungsbäumen berechnet.
Insgesamt ist der Random Forest Regressor ein mächtiges Modell, das in vielen Anwendungen verwendet wird und eine hervorragende Leistung bietet, insbesondere bei komplexen und grossen Datensätzen, wie in diesem Fall.

Zur Berechnung der jeweiligen Unterteilung der Daten können verschiedene Metriken wie die ≪Gini-Impurity≫ oder ≪Chi-Square≫ verwendet werden. Dabei werden die Trennungen, welche am besten abschneiden ganz oben im Entscheidungsbaum und somit als Erstes verwendet, um die Daten zu unterteilen.

3.2 Gradient Boosting Regressor

Der Gradient Boosting Regressor ist ein Machine Learning Algorithmus, der auf dem Konzept des Gradient Boosting aufbaut. Es ist eine Ensemble-Methode, was bedeutet, dass mehrere schwache Modelle zusammengeführt werden, um ein starkes Modell zu erhalten. Dieser Algorithmus verbessert die Genauigkeit des Modells durch zusätzliche Modelle, welche die Fehler der Modelle, welche vorher erstellt wurden, korrigiert. Der Prozess beginnt mit dem Training eines einfachen Modelles auf den Datensatz. Anschliessend wird die Vorhersagegenauigkeit des Modells bewertet, in dem die Residuen berechnet werden. Diese Residuen werden verwendet, um dann ein weiteres Modell zu trainieren, welches speziell auf die Korrektur dieser Residuen ausgerichtet ist. Das wird so lang wiederholt, bis eine bestimmte Anzahl von Modellen oder eine bestimmte Genauigkeit erreicht wurde. Am Schluss werden die Vorhersagen aller Modelle zusammengeführt, um dann eine gute Vorhersage zu erhalten.
Dieser Algorithmus erzielt in der Regel gute Leistungen auf komplexen und nicht linear abhängigen Problemen, jedoch ist es anfällig für Overfitting, weshalb eine gute Hyperparameter-Optimierung erforderlich ist.

3.3 Ridge Regression

Die Ridge Regression ist eine Variante der linearen Regression, die verwendet wird, um Überanpassung zu vermeiden. Sie tut dies, indem sie einen zusätzlichen Faktor zur Regularisierung zur Funktion der Kostenoptimierung hinzufügt. Dieser Term besteht aus der Summe der Quadrate der Koeffizienten (auch Gewichte genannt), multipliziert mit einem Regularisierung-Koeffizienten lambda.

Der Regularisierungs-Term bestimmt die Stärke der Regularisierung und beeinflusst die Koeffizienten des Modells. Ein kleinerer Wert von lambda führt zu grösseren Koeffizienten und damit zu einem komplexeren Modell, während ein höherer Wert von lambda zu kleineren Koeffizienten und damit zu einem einfacheren Modell führt.

Die Ridge Regression versucht, einen Kompromiss zwischen der Fähigkeit des Modells, die Trainingsdaten gut zu erklären, und der Einfachheit des Modells zu finden, indem es die Koeffizienten beschränkt. Dies führt zu einer geringeren Empfindlichkeit gegenüber kleinen Änderungen in den Daten und damit zu einer besseren allgemeinen Leistung des Modells.

Insgesamt ist die Ridge Regression eine nützliche Methode, um Überanpassung in linearen Regressionsmodellen zu vermeiden und damit bessere Resultate zu erzielen. Es kann insbesondere nützlich sein, wenn der Datensatz viele Features hat und die Koeffizienten der linearen Regression schwer zu interpretieren sind.

3.4 MLP Regressor

Ein Multilayer Perzeptron (MLP) ist eine Art von künstlichen neuronalen Netzwerken, das hauptsächlich für Regressionsprobleme verwendet wird. Es besteht aus mehreren Schichten von Neuronen (auch als Knoten bezeichnet), von welchen die erste Schicht die Eingabe- und die letzte Schicht die Ausgabeschicht ist. Die Schichten dazwischen werden als verborgene Schichten bezeichnet.

Jedes Neuron in einer Schicht ist mit den Neuronen in der nächsten Schicht verbunden, diese Verbindungen werden auch Gewichte genannt. Die Neuronen in einer Schicht empfangen Eingaben von den Neuronen in der vorherigen Schicht, multiplizieren sie mit den Gewichtungen und geben dann eine Ausgabe an die Neuronen in der nächsten Schicht weiter.

Diese Ausgabe entsteht, indem das Neuron alle Eingaben aufsummiert und auf diese Summe wird dann eine Aktivierungsfunktion angewendet. Es gibt verschiedene Aktivierungsfunktionen.

Das MLP Modell wird durch die Veränderung der Gewichte trainiert, um die Ausgaben des Modells an die tatsächlichen Werte anzupassen. Dies erfolgt durch Verwendung von Optimierungsverfahren wie dem Backward-Propagation Algorithmus.

Das MLP Modell ist besonders nützlich bei der Handhabung von komplexen Beziehungen in den Daten und kann auch mit grossen Datensätzen gut umgehen.

Das Trainieren eines MLP Models benötigt jedoch relativ viele Ressourcen, weshalb es mit einem grösseren Mehraufwand verbunden ist.

4 Resultate

Um den Preis vorherzusagen, wurde schlussendlich ein GradientBoosting Regressor mit einem Min-Max skalierten Datensatz verwendet. Es wurde über alle Schweizer Immobilien ein MAPE von rund 26 % erreicht auf den Daten des Kaggle-Wettbewerbes.

4.1 Variable Importance

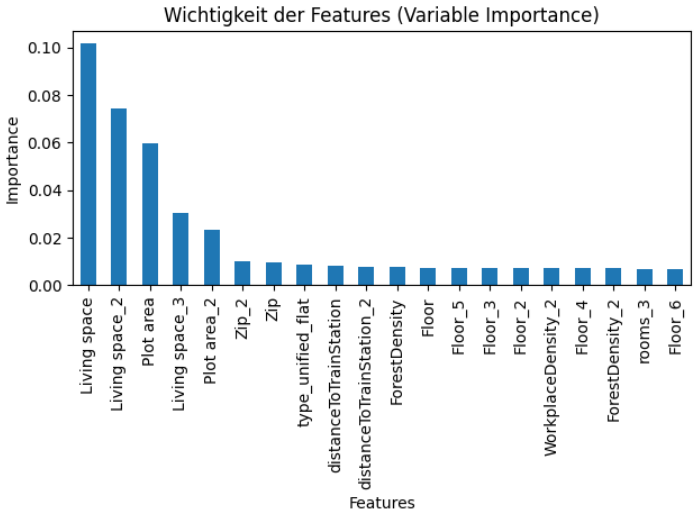

Figure 1: Die 20 wichtigsten Features vom besten Regressionsmodell

Der oben dargestellte Plot zeigt die Wichtigkeit der einzelnen Features für das beste Regressionsmodell. Die Wichtigkeit eines Features, gibt an, wie stark ein bestimmtes Feature dazu beiträgt, die Vorhersage des Modells zu verbessern. Es misst die Abhängigkeit der Zielvariable «price» von einem bestimmten Feature. Wie man im Plot erkennen kann, haben einige Features eine höhere Wichtigkeit als andere. Zum Beispiel ist «Living space» am stärksten dafür verantwortlich, die Vorhersage des Modells zu verbessern, gefolgt vom Feature «Living space_2», was das Quadrat von «Living space» ist. Im Plot sind die 20 wichtigsten Features aufgelistet, wobei 3 davon von «Living space» abstammen. Allgemein lässt sich sagen, dass dieser Plot wichtige Informationen über die Features liefert, die am stärksten dazu beitragen, die Vorhersage des Modells zu verbessern. Dies kann helfen, die Bedeutung der Features besser zu verstehen und Entscheidungen bezüglich der Auswahl von Features für zukünftige Modelle zu treffen.

4.2 Rückblick

Mit einem MAPE-Score von 26 Prozent erhalten wir ein stabiles Modell, welches jedoch im Schnitt rund ein Viertel von dem tatsächlichen Preis abweicht.
Eine Möglchkeit zur Verbesserung des Modelles wäre das hinzuziehen weiterer Feature engineering Methoden wie Interaktionen (mehrere features miteinander kombinieren).
Ausserdem könnte man mit einer tieferen Residuenanalyse eventuell systematische Fehler finden.
Zusätzlich könnte man auch weitere Modelle trainieren und diese vergleichen. Da mit dem Gradient Boosting das beste Resultat erziehlt wurde, wäre es naheliegend weiter Modelle mit diesem Ansatz wie Beispielsweise ein XGboost Regressor zu trainieren.