Universität Münster
AG AIS – Prof. Dr. Malte Schilling
www.uni-muenster.de/AISystems Einsteinstr. 62 48149 Münster

Mini-Lab 1 zur Einführung Neuronale Netzwerke: Lineare Regression und Merkmalsanalyse

Janosch Bajorath
j.bajorath@uni-muenster.de
Simon Neumeyer
sneumeye@uni-muenster.de
Malte Schilling
malte.schilling@uni-muenster.de

Abgabe bis zum 30. 11. 2025 über das LearnWeb.

Ziel: In diesem Mini-Lab sollen Sie den Umgang mit einfachen linearen Modellen auf realen Daten demonstrieren. Sie sollen dabei üben, den gegebenen Datensatz zu untersuchen, geeignete Merkmale auszuwählen und die Leistungsfähigkeit verschiedener Modelle zu vergleichen. Dabei wird besonderes Augenmerk auf die Bedeutung einzelner Features, den Umgang mit fehlenden Daten und die Generalisierbarkeit von Modellen gelegt. Im zweiten Teil kommt dann die Anwendung des Gradientenabstiegsverfahrens hinzu, um den Transfer auf andere räumlich oder zeitlich erhobene Daten zu untersuchen, bis hin zu einer Form von Online Learning.

Tasks (jeweils mit 20% gewichtet; es werden 50% an Punkten benötigt):

1. Erste Analyse, Vorverarbeitung und Bereinigung der Daten, erstes Vergleichsmodell
2. Aufbau eines linearen Regressionsmodells und Erweiterung um zusätzliche Features
3. Modellvergleich für polynomial erweiterte Modelle der wichtigsten Merkmale
4. Transfer auf eine neue Stadt mit begrenzten Daten
5. Transfer auf einen neuen Zeitraum und inkrementelles (Online-)Lernen

Teil A: Merkmalsanalyse und Model Selection

Ziele: Einführung in den praktischen Umgang mit realen Daten, Aufbau und Vergleich erster Regressions- und Polynommodelle sowie Verständnis und Anwendung grundlegender Methoden der Modellselektion zur Vermeidung von Overfitting.

Datensatz: Grundlage ist ein Datensatz von Immobilienangeboten (gescrapt von Immobilienscout24; 2018 bis 2019), der Mietpreise, Flächenangaben und eine Vielzahl weiterer Objektmerkmale enthält. Dieser Datensatz ist aufgeteilt in einen Trainingsdatensatz und einen Validierungsdatensatz, den sie nutzen können, um während der Entwicklung ihre Modelle auf einem festen Datensatz evaluieren zu können. Ein weiterer Testdatensatz wird später in der Überprüfung der Abgaben zum Messen der Generalisierung eingesetzt und so zur abschließenden Modellbewertung verwendet.

Aufgabe 1: Datenanalyse, Bereinigung und erstes Modell

In dieser Aufgabe lernen Sie den bereitgestellten Datensatz kennen und bereiten ihn für die Modellierung vor. Ziel ist es, sich mit der Datenstruktur vertraut zu machen, fehlerhafte oder unvollständige Einträge zu behandeln und ein erstes einfaches Regressionsmodell zu erstellen, das als Baseline dient. Die Vorhersagegröße ist `totalRent`, d.h. die gesamte monatliche Miete.

Im Verlauf dieser Aufgabe sollen Sie:

- den Datensatz laden und sich einen Überblick über seine Struktur verschaffen (Spalten, Datentypen, Größenordnungen) [1],
- Ausreißer identifizieren (benutzen Sie dafür zum Beispiel auch geeignete Visualisierungen als Zwischenschritt) und ggf. entfernen oder begründen, warum sie beibehalten werden,
- fehlende Werte erkennen und sinnvoll behandeln (z.B. Mittelwert, Median, oder Abschätzung aus anderen Spalten — etwa `totalRent` aus `baseRent` und `serviceCharge`),
- ein erstes lineares Regressionsmodell zur Vorhersage von `totalRent` trainieren und evaluieren (über einer Inputdimension).

Überprüfung / Test:
Für diesen Aufgabenteil arbeiten Sie in `src/baseline_model.py`:
`BaselineLinearModel` und `src/preprocessing.py: clean_data, encode_categorical`.

1.1 Der Trainingsdatensatz enthält mindestens `n_train > 330` bereinigte Einträge (keine fehlenden Werte; alle in numerischer Form) mit mehr als zehn Features.

1.2 Die Performance des Baseline-Modells (lineare Regression über bereinigte Daten, eine Inputdimension) liegt typischerweise bei $R^2 > 0.75$ und einem `RMSE` im Bereich von $230 - 250$ € auf dem Validierungs-/Testdatensatz.

1.3 Es werden dabei auch kategoriale Features sinnvoll extrahiert. So erreicht ein separates Modell, das ausschließlich eins dieser kategorialen Features (nach einfacher Kodierung) verwendet, eine Vorhersagegüte von mindestens $R^2 > 0.05$. Dies zeigt damit, dass auch kategoriale Informationen einen Beitrag zur Modellierung leisten können.

[1] Einen Einstieg in `pandas` geben wir in den Übungen oder finden Sie hier: `https://pandas.pydata.org/docs/user_guide/10min.html`, sowie vertiefende Beispiele für verschiedenste Einsatzzwecke hier `https://pandas.pydata.org/docs/user_guide/cookbook.html`.

Aufgabe 2: Feature-Analyse und schrittweise Modellverbesserung

Sie sollen im nächste Schritt untersuchen, welche Merkmale (Features) die größte Bedeutung für die Mietpreisvorhersage haben. Ziel ist es, den Beitrag einzelner Variablen zu quantifizieren und so ein Modell zu entwickeln, das auf den wichtigsten Features basiert.

Beginnen Sie mit einer inhaltlichen Einschätzung: Welche Merkmale könnten Ihrer Erwartung nach besonders hilfreich für die Vorhersage sein? Überlegen Sie, welche Zusammenhänge mit der Zielgröße `totalRent` plausibel sind (z.B. Fläche, Lage, Baujahr, Ausstattung, Energieeffizienz). Benutzen Sie dabei nicht `baseRent` und `serviceCharge` als Eingaben (und überlegen Sie, warum deren Verwendung problematisch wäre)!

Sie sollen so den tatsächlichen Einfluss der Features empirisch untersuchen:

- Untersuchen Sie zuerst jedes Feature für sich, um dessen individuellen Zusammenhang mit der Zielgröße `totalRent` zu bestimmen (z.B. über die Bestimmung der linearen Regression mit nur diesem Feature). So erhalten Sie eine erste Einschätzung, welche Variablen für sich genommen am stärksten mit der Zielgröße zusammenhängen.

- Entwickeln Sie anschließend ein schrittweises Auswahlverfahren, das jeweils dasjenige Feature als nächstes hinzufügt, welches die Modellgüte (z.B. gemessen an R^2 oder RMSE) am stärksten verbessert. Beginnen Sie mit dem einflussreichsten Einzel-Feature aus dem vorherigen Schritt und fügen Sie in jedem weiteren Schritt genau ein weiteres Feature hinzu — immer dasjenige, das in Kombination mit den bereits gewählten Features den größten zusätzlichen Leistungszuwachs bringt. Auf diese Weise werden redundante Merkmale automatisch benachteiligt, während sich ergänzende Merkmale bevorzugt werden.

- Vergleichen Sie die Ergebnisse beider Untersuchungen und schlussfolgern Sie, welche Features offenbar ähnliche Informationen liefern (redundant sind) und welche sich gegenseitig sinnvoll ergänzen (komplementäre Merkmale bieten).

- Untersuchen Sie dabei auch mindestens zwei kategoriale Merkmale (z.B. Lage, Heizungsart, Gebäudetyp) und analysieren Sie ihren Beitrag zur Modellgüte.

Überprüfung / Test:
Für diesen Aufgabenteil implementieren Sie Funktionen in
`src/feature_analysis.py`, `src/visualization.py` und eine Funktion für mehr-dimensionale Inputs in `src/baseline_model.py`.

2.1 Eine geordnete Liste der wichtigsten Features (`analyze_single_features`) liegt vor, sortiert nach absteigendem R^2-Wert. Diese Liste soll mindestens zehn Einträge enthalten, davon zwei kategoriale Variablen. Die R^2-Werte liegen im plausiblen Bereich $0 \leq R^2 \leq 1$ und die Sortierung ist streng monoton fallend.

2.2 Eine geordnete Teilliste der (nicht redundanten) Features in absteigender Reihenfolge ihrer Nützlichkeit, basierend auf der schrittweisen Verbesserung der Modellgüte (`stepwise_selection`); diese soll zeigen, wie sich die Modellperformance mit jedem hinzugefügten Feature verbessert.

2.3 Ein Modell mit fünf Features erreicht auf dem Testdatensatz eine Performance von mindestens $R^2 > 0.8$ und einem `RMSE` kleiner als `230 €`.

2.4 In einem Plot werden für Trainings- und Validierungsdatensatz die Performances der Modelle unterschiedlicher Komplexität (Anzahl der genutzten Features) visualisiert, um den Einfluss zusätzlicher Features nachvollziehbar zu machen.

Aufgabe 3: Polynomiale Modellierung und Model Selection

Nachdem Sie in den vorherigen Aufgabenteilen die wichtigsten Merkmale für die Mietpreisvorhersage identifiziert haben, sollen Sie nun untersuchen, wie sich die Auswahl eines Modells und die der Modellkomplexität auf die Vorhersagegüte auswirken. Dazu betrachten Sie Polynommodelle verschiedener Grade in Kombinationen mit den zuvor identifizierten Features. Ziel ist es, ein Modell zu finden, das die Daten möglichst gut beschreibt, ohne dabei aber zufällige Schwankungen der Trainingsdaten mit zu lernen.

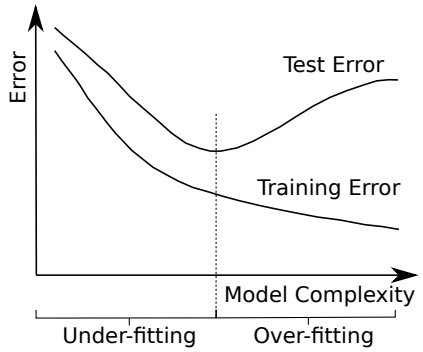

Abbildung 1: Zusammenhang Modellkomplexität und Fehler: Bei zu einfachen Modellen (links) tritt *Underfitting* auf, bei zu komplexen Modellen (rechts) *Overfitting*. Das optimale Modell liegt im Bereich minimaler Testfehler.

Verwenden Sie drei einflussreiche numerische Features aus Ihrer vorherigen Untersuchung (z.B. aus den Features `livingSpace`, `yearConstructed`, `numberOfRooms`, `floor`, `condition`) und erweitern Sie diese jeweils um polynomiale Terme bis zu einem Grad von 6. Die Modelle sollen weiterhin mithilfe der Normalengleichung (Pseudo-Inverse) über den Trainingsdatensatz gelöst und anschließend über den Validierungsdatensatz evaluiert werden.

- Erstellen Sie für die ausgewählten Features polynomiale Erweiterungen der Grade 1 bis 6 (z.B. mit `PolynomialFeatures` aus `scikit-learn`). Dieses Verfahren erzeugt

aus den ursprünglichen Eingabevariablen die zusätzlichen polynomialen Terme und stellt die Design Matrix auf, wodurch auch nichtlineare Zusammenhänge in einem linearen Modell abgebildet werden können.[2]

- Trainieren Sie für jede Kombination aus Feature-Anzahl (1–3) und Polynomgrad (1–6) ein Modell und berechnen Sie die Modellgüte (angeben als R^2 oder RMSE) auf dem Trainings- und Validierungsdatensatz.

- Visualisieren Sie die Ergebnisse auf zwei Arten:
  - Visualisieren Sie die Ergebnisse in einer *Heatmap*, in der die Achsen den Polynomgrad (x-Achse) und die Anzahl der genutzten Features (y-Achse) darstellen, sowie die Farbskala die jeweilige Modellgüte (z.B. R^2 oder RMSE) darstellt. Nutzen Sie hierfür z.B. `matplotlib.pyplot.imshow()` (alternativ: `seaborn.heatmap()`) und beschriften Sie die Achsen sowie geben eine Farbskala an (`plt.colorbar()`).[3] Sie können dazu dann als Erweiterung dies in einer dreidimensionale Grafik mit `matplotlib.pyplot.plot_surface()` darstellen[4].
  - Zum anderen, stellen Sie in einem separaten 2D-Plot für jede Feature-Anzahl den Verlauf der Modellgüte (siehe Abbildung) über den Polynomgrad dar (mit durchgezogenen Linien für den Trainingsfehler und gestrichelten Linien für den Testfehler; Sie erhalten so je zwei Kurven für jede der drei Feature-Kombinationen).

- Bestimmen Sie für jedes der drei betrachteten Features den optimalen Polynomgrad, bei dem das Modell die beste Testleistung erzielt, und den kleinsten Grad, ab dem Overfitting deutlich sichtbar wird. Dokumentieren Sie diese Werte in einer Tabellenform (diese wird zurück gegeben als ein `dict` als Rückgabe des Aufrufs von `analyze_polynomial_performance(poly_results)`).

Überprüfung / Test:
Für diesen Aufgabenteil implementieren Sie Funktionen in `src/polynomial_analysis.py`.

3.1 Für jedes der drei Feature-Kombinationen ist ein optimaler Polynomgrad anzugeben (typischerweise zwischen 1 und 4) sowie ein Grad, ab dem Overfitting sichtbar wird (vermutlich im Bereich von Grad 3–6). Überlegen Sie, warum dies für mehr Features instabiler wird oder werden könnte (Dimensionalität der Design-Matrix) bzw. welchen Einfluss dies auf Overfitting haben könnte.

[2] Siehe offizielle `scikit-learn`-Dokumentation zu `PolynomialFeatures` mit Beispielen: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html`

[3] Beispiel und Erläuterung einer annotierten Heatmap finden Sie in der offiziellen Matplotlib-Dokumentation: `https://matplotlib.org/stable/gallery/images_contours_and_fields/image_annotated_heatmap.html`

[4] Ein Beispiel für eine solche 3D-Oberflächendarstellung finden Sie unter `https://matplotlib.org/stable/gallery/mplot3d/surface3d.html`

3.2 Die Heatmap (oder 3D-Darstellung) soll den deutlichen Leistungsanstieg im Training und gleichzeitig den Einbruch der Testleistung bei hohen Graden aufzeigen.

3.3 In den 2D-Plots sind für alle Feature-Kombinationen über den Polynomgrad getrennte Kurven für Trainings- und Testleistung dargestellt.

3.4 Das insgesamt beste Modell erreicht eine hohe Trainingsperformanz ($R^2 > 0.83$). Für den Validierungsdatensatz sollten Sie dabei ebenso auf eine Performance von etwa $R^2 \approx 0.8$–0.9 kommen.

Teil B: Gradient Descent und Online Learning

Aufgabe 4: Räumlicher Transfer und Gradient Descent

In diesem Teil des Mini-Lab werden Konzepte der Model Selection und der Optimierung über iterative Verfahren miteinander verbunden. Sie lernen, wie ein bereits trainiertes Modell auf neue Daten übertragen werden kann (Transfer) und wie mithilfe des *Gradient Descent*-Verfahrens Modelle schrittweise verbessert oder an neue Daten angepasst werden können. Dies ist ein Schritt hin zum (dann folgenden) *Online-Learning*.

Nachdem Sie in den bisherigen Aufgabenteilen Modelle zur Mietpreisvorhersage für Münster erstellt und optimiert haben, sollen Sie nun untersuchen, inwieweit sich diese Modelle auf Daten von Bielefeld übertragen lassen. Zugleich sollen Sie als weiteres numerisches Optimierungsverfahren Gradient Descent anwenden, das eine schrittweise Anpassung von Modellparametern erlaubt und so auch für inkrementelles Lernen geeignet ist.

Ziel: Untersucht wird, wie gut sich trainierte Modelle auf neue Regionen übertragen lassen, wie stark sie dabei über- oder unteranpassen, und ob ein fortlaufendes Lernen über Gradient Descent helfen kann zu einer verbesserten Generalisierung bzw. zum Abstellen von Overfitting. Die Daten der neuen Stadt (Bielefeld) liegen in gleicher Struktur vor wie die bisherigen Datensätze, enthalten jedoch nur sehr wenige Trainingsbeispiele (ca. 30).

- Wenden Sie Ihre bestehende Datenbereinigung (aus `preprocessing.py`) auf den Datensatz `train_bielefeld.csv` an. Prüfen Sie einmal kurz, welche Spalten vollständig vorliegen, und ob sich deren Verteilungen von den Münster-Daten unterscheiden (was für Erwartungen ergeben sich daraus für die Vorhersagequalität?).

- Wenden Sie ein lineares Regressionsmodell (Achtung – Anpassung: Sie müssen nicht das Polynommodell nutzen!), das auf allen Münsteraner Daten trainiert wurde (alle 10 oder mehr ihrer Features verwenden), direkt auf den Bielefeld-Datensatz an.

- Trainieren Sie als Vergleich dazu ein Modell, das ausschließlich auf den Bielefeld-Daten basiert. Verwenden Sie dabei dieselbe Modellstruktur (alle extrahierten Features). Überprüfen Sie ob –und in welchem Maße– das Modell overfittet. Sie können auch Cross-Validation testen.

- Implementieren Sie ein lineares Modell mit Gradient Descent in `src/gradient_descent.py`. Trainieren Sie dieses Modell zunächst auf den Münster-Daten (alle 10 oder mehr Features verwenden). und verwenden Sie anschließend das gleiche Modell, um es auf die Bielefeld-Daten weiter zu trainieren. Beobachten Sie dabei, wie sich Trainings- und Validierungsfehler im Verlauf des weiteren Trainings auf den wenigen Samples entwickelt. Nutzen Sie *Early Stopping*, um das Training zu beenden, sobald die Testleistung beginnt, sich zu verschlechtern.[5] Vergleichen Sie dieses inkrementell trainierte Modell mit dem analytisch gelösten linearen Modell aus dem ersten Aufgabenteil.

Überprüfung / Test:
Für diesen Aufgabenteil implementieren Sie Funktionen in `src/spatial_transfer.py` und das Gradientenabstiegsverfahren in `src/gradient_descent.py` (darin ist als Hilfe ein mögliches Interface vorgegeben und dazu einzelne Teile schon implementiert, wie zum Beispiel die notwendige Normalisierung der Daten).

4.1 Anwendung der `clean_data`-Funktion auf den neuen Datensatz führt zu mindestens `n_train > 25` gültigen Trainingsbeispielen.

4.2 Das übertragene Münster-Modell zeigt auf den Bielefeld-Daten einen drastischen Abfall der Testgüte (Validierung auf Bielefeld-Daten liefert R^2 um $0.2 - 0.4$).

4.3 Das lokal auf Bielefeld trainierte Modell zeigt Overfitting ($R^2_{\text{train}} > 0.8$, $R^2_{\text{test}} < 0.65$).

4.4 Die Implementierung von Gradient Descent ist funktional und konvergiert mit geeigneter Lernrate (z.B. $\eta \in [10^{-3}, 10^{-2}]$).

4.5 Der Gradient-Descent-Ansatz mit Early Stopping liefert ein verbessertes Modell (kein Overfitting auf den wenigen Trainingsdaten und bessere Generalisierung als ein auf einem anderen Ort trainiertes Modell, $R^2_{\text{test}} > 0.65$).

Aufgabe 5: Online Learning und zeitlicher Transfer

In dieser abschließenden Aufgabe sollen Sie Gradient Descent anwenden als eine Form von *inkrementellem Lernen* (Online Learning). Dabei soll ein Modell schrittweise an neue Daten angepasst werden, die zu einem späteren Zeitpunkt (Jahr 2025) erhoben wurden.

Als Ausgangspunkt dienen die Modelle und Funktionen aus Aufgabe 4, insbesondere die Implementierung von Gradient Descent. Die neuen Daten besitzen eine ähnliche, aber nicht vollständig identische Struktur wie die bisherigen Datensätze (2018-2019) und es haben sich Features geändert. Daher sollen nur jene Features berücksichtigt werden, die in beiden Datensätzen vorkommen.

Ziel: Untersuchen Sie, wie sich ein bereits trainiertes lineares Modell auf zeitlich neuen Daten verhält und dann auf diese neue Daten anpassen lässt. Vergleichen Sie das Verhalten

[5] Ein Überblick zu Gradient Descent und Early Stopping findet sich in der `scikit-learn`-Dokumentation: `https://scikit-learn.org/stable/modules/sgd.html`

eines Modells, das vollständig neu auf den 2025-Daten trainiert wurde, mit einem Modell, das durch schrittweise Weiteranpassung (Online-Learning) verbessert wird.

- Machen Sie sich vertraut mit dem Datensatz `train_2025.csv` und der Struktur der Daten. Führen Sie notwendige Vorverarbeitung durch und identifizieren Sie gemeinsame Features, die sowohl in den 2018- als auch in den 2025-Daten enthalten sind.
- Trainieren Sie ein lineares Regressionsmodell (wie in den vorhergehenden Aufgaben aufgestellt mit der Normalengleichung) nur auf den Trainingsdaten aus 2025 (es liegen nur 25 Samples dafür vor). Verwenden Sie Cross-Validation zur Abschätzung der Generalisierungsleistung.
- Nutzen Sie anschließend Ihre bestehende Gradient Descent-Implementierung, um ein Modell zunächst auf den Münster-2018-Daten vorzutrainieren und danach schrittweise (inkrementell) auf die neuen 2025-Daten weiter anzupassen. Zeichnen Sie während des Trainings eine *Learning Curve*, die den Verlauf von Trainings- und Testfehler (RMSE) über die Epochen zeigt.
- Vergleichen Sie das weitertrainierte Modell mit dem Modell, das nur auf den 2025-Daten trainiert wurde, sowie mit dem unveränderten Modell, das nur auf den 2018-Daten basiert. Überlegen Sie, welche Variante die beste Balance zwischen Stabilität und Anpassungsfähigkeit bietet.
- Variieren Sie Lernrate (η) und ggf. Regularisierung (λ) und beobachten Sie, wie diese Parameter den Anpassungsverlauf und das Risiko des Überanpassens beeinflussen.[6]

Überprüfung / Test:
Für diesen Aufgabenteil implementieren Sie Funktionen in `src/temporal_transfer.py` und ggf. in `src/gradient_descent.py` .

5.1 Ein neu trainiertes 2025-Modell zeigt erwartungsgemäß starkes Overfitting ($R^2_{\text{train}} > 0.9$, $R^2_{\text{validation}} < 0.7 - 0.80$).

5.2 Ein auf den 2018er Daten ursprüngliches Gradient-Descent-Modell zeigt anfangs eine sehr schlechte Performanz ($R^2_{\text{validation\_2025}} < 0.3$), erreicht dann aber beim weitertrainieren mit neuen Daten aus 2025 eine verbesserte Testleistung von etwa $R^2 \approx 0.7 - 0.85$ bei stabiler Trainingskurve (kein Überanpassen).

5.3 Zeigen Sie eine Learning Curve.

[6] Eine Einführung zu Lernratensteuerung und inkrementeller Modellanpassung findet sich in der `scikit-learn`-Dokumentation zu SGD: `https://scikit-learn.org/stable/modules/sgd.html`