

Bachelor-Thesis

zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

an der Hochschule für Technik und Wirtschaft des Saarlandes
im Studiengang Praktische Informatik
der Fakultät für Ingenieurwissenschaften

Effiziente Generierung von Trainingsdaten in der Bildklassifikation

vorgelegt von
Jan Rauber

betreut und begutachtet von
Prof. Dr.-Ing. Klaus Berberich

Saarbrücken, 03. 09. 2025

Selbständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit (bei einer Gruppenarbeit: den entsprechend gekennzeichneten Anteil der Arbeit) selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich erkläre hiermit weiterhin, dass die vorgelegte Arbeit zuvor weder von mir noch von einer anderen Person an dieser oder einer anderen Hochschule eingereicht wurde.

Darüber hinaus ist mir bekannt, dass die Unrichtigkeit dieser Erklärung eine Benotung der Arbeit mit der Note „nicht ausreichend“ zur Folge hat und einen Ausschluss von der Erbringung weiterer Prüfungsleistungen zur Folge haben kann.

Saarbrücken, 03. 09. 2025

Jan Rauber

Zusammenfassung

Die hohen Kosten für die Annotation von Trainingsdaten limitieren den Zugang zu maschinellem Lernen, insbesondere für ressourcenbeschränkte Akteure. Active Learning verspricht erhebliche Label Einsparungen bei erhaltener Modellqualität, jedoch fehlen systematische Evaluationen für verschiedene Datensatztypen. Diese Pilotstudie untersucht drei Uncertainty-basierte Active-Learning-Strategien auf MNIST, Fashion-MNIST und einem unbalancierten Dachmaterialdatensatz mit jeweils 5 Wiederholungen pro Konfiguration und beobachtet Label Einsparungen von bis zu 98% bei hohen Effektstärken (Cliff's Delta bis 1.0), wobei Margin Sampling konsistent die besten Ergebnisse liefert. Obwohl die geringe Stichprobengröße keine statistische Signifikanz erlaubt, rechtfertigen die viel-versprechenden Ergebnisse eine umfangreichere Folgestudie mit 30-50 Wiederholungen zur Validierung des praktischen Potenzials von Active Learning für die Demokratisierung von KI-Technologien.

*We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.*

— Donald E. Knuth [42]

Danksagung

Ich danke Herrn Prof. Dr.-Ing. Klaus Berberich für die Vergabe dieses interessanten Themas und die gute Betreuung während der Bearbeitungszeit. Das Vertrauen, mir dieses besondere Thema zu überlassen, weiß ich sehr zu schätzen.

Inhaltsverzeichnis

1 Einleitung	1
2 Theoretische Grundlagen	3
2.1 Lernen: Eine Definition	3
2.2 Maschinelles Lernen	3
2.3 Passives Lernen	3
2.4 Aktives Lernen	3
2.5 Der aktive maschinelle Lernprozess	3
2.6 Auswahl der Daten mit hoher Informationsdichte	4
2.6.1 Unsicherheitsbasiertes Sampling	4
2.7 Wilcoxon Signed Rank Test	7
2.8 Der P-Wert	8
2.8.1 Berechnung des P-Werts	8
2.9 Bonferroni-Korrektur	8
2.10 Effektstärke	8
2.11 Verwendete Klassifikatoren	9
2.11.1 Random Forest	9
2.11.2 SVM	10
2.11.3 Multinomiale Logistische Regression	12
2.11.4 Neuronales Netz	12
2.11.5 Naïve Bayes	14
2.12 Konfusionsmatrix	14
2.13 Klassifikationsproblem	15
2.14 Overfitting und Underfitting	15
2.15 Effekte unbalancierter Datensätze auf die Klassifikationsleistung der Machine Learning Modelle	15
2.16 Batch Active Learning und Sequencial Active Learning	16
2.17 QGIS	16
2.18 Evaluationskriterien und erwartete optimale Kombinationen	17
2.18.1 Literaturbasierte Kombinationen für ausgewählte Evaluationskriterien	17
2.19 Dachmaterialklassifikation mittels CNNs in Luftbildern	19
3 Datensätze	21
3.1 Dachmaterialdatensatz	21
3.1.1 Aufbau	22
3.1.2 Der Zweck	28
3.1.3 Limitationen	28
3.2 MNIST	28
3.2.1 Aufbau	28
3.2.2 Der Zweck	33
3.2.3 Limitationen	33
3.3 Fashion-MNIST	33
3.3.1 Aufbau	33
3.3.2 Der Zweck	36

3.3.3 Limitationen	36
3.4 Vergleich und Entwicklung der Datensätze	36
4 Technische Grundlagen	39
4.1 Überblick über die Implementierung	39
4.2 Datenverarbeitung und -analyse	39
4.2.1 NumPy	39
4.2.2 Pandas	41
4.3 Maschinelles Lernen	42
4.3.1 Modelltraining mit Scikit-learn	43
4.3.2 Modellevaluierung mit Scikit-learn	44
4.4 Pytorch	45
4.4.1 Architektur neuronaler Netze	45
4.4.2 Optimierung mit Adam	46
4.4.3 Datenhandling mit DataLoader	47
4.5 Statistische Auswertung	48
4.6 Datenvisualisierung	48
4.6.1 Matplotlib	49
4.6.2 Seaborn	49
4.7 Sicherstellung der Reproduzierbarkeit	51
4.8 GPU-optimiertes aktives Lernen auf den Datensätzen	52
4.9 NVIDIA RTX 4060 Architektur und Funktionsweise	54
4.10 CuPy	55
4.11 RAPIDS cuML	55
4.12 TorchVision	56
5 Evaluierung	57
5.1 Konkretes Setup	57
5.1.1 Methodische Rahmenbedingungen und Interpretation	57
5.1.2 Initiale Selektion	58
5.2 Evalierung MNIST	58
5.2.1 Endgenauigkeit in Relation zu Random Sampling	58
5.2.2 Labeleinsparung in Relation zu Random Sampling	61
5.2.3 Trainingszeit	65
5.2.4 Zwischenergebnis MNIST	68
5.3 Fashion MNIST	69
5.3.1 Endgenauigkeit in Relation zu Random Sampling	69
5.3.2 Labeleinsparung in Relation zu Random Sampling	75
5.3.3 Trainingszeit	77
5.3.4 Zwischenergebnis Fashion-MNIST	83
5.4 Dachmaterialdatensatz	84
5.4.1 Endgenauigkeit in Relation zu Random Sampling	84
5.4.2 Labeleinsparung in Relation zu Random Sampling	85
5.4.3 Trainingszeit	86
5.4.4 Zwischenergebnis Dachmaterialdatensatz	87
5.5 Gründe für das Fehlen statistischer Signifikanz	87
5.5.1 Post-hoc Power-Analyse	88
5.5.2 Übersicht statistischer Kennzahlen	88
5.5.3 Interpretation der Ergebnisse	88
5.5.4 Konfidenzintervalle der Labeleinsparungen	88
5.5.5 Praktische versus statistische Signifikanz	90

5.5.6	Ökonomische Betrachtung	90
5.6	Fehlende Stopping-Kriterien	90
5.7	Versagensfälle und deren Analyse	90
5.7.1	Übersicht der Versagensfälle	90
5.7.2	Systematische Kategorisierung der Versagensfälle	90
5.7.3	Root-Cause-Analyse der Versagensmechanismen	92
5.7.4	Quantitative Analyse der Versagensfälle	93
5.7.5	Lösungsansätze für identifizierte Probleme	93
5.7.6	Warum Margin Sampling konsistent überlegen ist	94
5.7.7	Lessons Learned und Best Practices	94
5.8	Strategievergleich über alle Datensätze	95
5.8.1	Vergleich mit Meta-Analysen	95
5.8.2	Konsistenz der Strategien	95
5.8.3	Hyperparameter-Einflüsse	95
5.9	Vergleich mit aktuellen Active Learning Benchmarks	96
5.10	Zusammenfassung und Ausblick	97
6	Fazit	99
6.1	Kritische Würdigung und Limitationen der Arbeit	99
6.1.1	Methodische Einschränkungen der Experimentdurchführung	99
6.1.2	Fehlende statistische Signifikanz trotz hoher Effektstärken	100
6.1.3	Limitationen der Datensatzauswahl	100
6.1.4	Eingeschränkte Strategievielfalt	100
6.1.5	Diskrepanz zwischen Ergebnissen und Schlussfolgerungen	100
6.1.6	Unvollständige Kosten-Nutzen-Analyse	101
6.1.7	Fazit der kritischen Würdigung	101
Literatur	103	
Abbildungsverzeichnis	113	
Tabellenverzeichnis	114	
Listings	114	
Abkürzungsverzeichnis	117	

1 Einleitung

Die vorliegende Arbeit dient der Vorbereitung auf das Berufsleben als Junior-Softwareentwickler bzw. Junior-Data-Scientist. Eine häufige Kundenanforderung besteht darin, den optimal performanten Klassifikator für einen gegebenen Datensatz zu identifizieren. Diese Arbeit untersucht systematisch verschiedene Kombinationen aus Klassifikatoren und Query-Strategien mittels Brute-Force-Ansatz, um die leistungsfähigste Konfiguration zu ermitteln.

Für die Datensätze MNIST, Fashion-MNIST und den Dachmaterialiedatensatz werden individuelle Ergebnisse erwartet, da die Klassifikatoren mit ihren Auswahlstrategien unterschiedliche Stärken und Schwächen aufweisen und datensatzspezifische Eignungen zeigen.

2 Theoretische Grundlagen

Im Folgenden werden die theoretischen Grundlagen erläutert, die zum Verständnis dieser Arbeit notwendig sind.

2.1 Lernen: Eine Definition

Lernen ist die Fähigkeit des Geistes, sich seiner Umwelt anzupassen und sie sich zunutze zu machen. Zum Beispiel durch Werkzeuge wie Sprache, Mathematik, Physik, Musik und so weiter. Die Physis, die dem biologischen Lernen zugrunde liegt, zum Beispiel ein Gehirn, ist weitgehend unerforscht.[37]

2.2 Maschinelles Lernen

Maschinelles Lernen ist eine Disziplin in der Informatik. Hierbei wird versucht, die biologische Fähigkeit zu lernen, auf eine Maschine zu übertragen. Ziel der Wissenschaft ist es, in Zukunft das biologische Pendant zu übertreffen. Maschinelles Lernen gleicht einer komplizierten Excel-Tabelle und hat mit seinem biologischen Pendant nichts gemeinsam.[45]

2.3 Passives Lernen

Bei diesem Konzept bekommt die Maschine einen Datensatz und studiert diesen Datensatz vollständig, ohne zu verstehen, welche Daten relevant sind. Es gleicht eher einem Auswendiglernen des Datensatzes. Die Performance kann daher auf einen anderen Datensatz derselben Kategorie stark abweichen, weil der Fokus nicht auf dem Verstehen, sondern auf dem Auswendiglernen lag.[102]

2.4 Aktives Lernen

Hierbei lernt das Modell deutlich schneller, also mit weniger Labels, weil es sich aktiv am Lernprozess beteiligt, zum Beispiel durch das Herauspicken informativer Datenpunkte, um die Modellgüte zu verbessern.[22].

2.5 Der aktive maschinelle Lernprozess

- Der Mensch, der in diesem Fall die Rolle des Orakels einnimmt, labelt initial einen Bruchteil des Datensatzes, aber nicht den vollständigen Datensatz, wie es beim passiven Lernen der Fall wäre. [65]
- Daraufhin trainiert die Maschine auf diesen Bruchteil der Daten und wählt anschließend aus einem Pool aus den restlichen ungelabelten Daten aus dem Datensatz interessante Daten aus, die für das Modell den größten Informationsgehalt versprechen, um den größten Lernfortschritt zu erzielen. Das Modell fragt also nach. Es

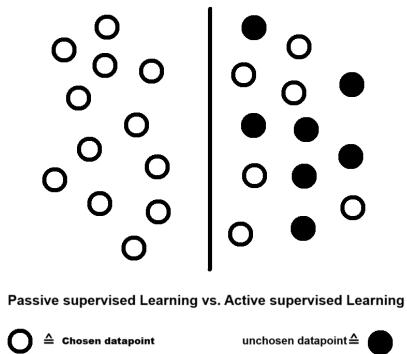


Abbildung 2.1: Diese Grafik beschreibt den iterativen Prozess, bei dem sich der Active Learner die Daten mit dem höchsten Informationsgewinn auswählt, um Labels einzuspannen gegenüber dem passiven Ansatz. Dieser Prozess wird so lange wiederholt, bis keine ungelabelten Daten mehr vorhanden sind, die vom Active Learner ausgewählt und an das Orakel zur Annotation geschickt werden können, oder bis ein Abbruchkriterium erreicht wurde. Zum Beispiel wenn der Active Learner die gewünschte Accuracy bereits erreicht hat. Der Active Learner erreicht in der Regel die gewünschte Accuracy mit deutlich weniger Labels, als beim passiven Ansatz notwendig wären.

werden somit viel schneller Verknüpfungen für das Verständnis der Daten erstellt, als das beim passiven Lernen der Fall ist, und somit wird der Annotationsaufwand für gleichbleibende oder bessere Modellleistung verringert. [65]

- Dieser Prozess wiederholt sich iterativ, bis keine zu labelnden Daten aus dem Pool mehr vorhanden sind oder ein Abbruchkriterium erreicht wurde. Die folgende Grafik illustriert den Prozess des aktiven Lernens. [65]

2.6 Auswahl der Daten mit hoher Informationsdichte

Dem Active Learner stehen mehrere Strategien zur Verfügung, um die Informationsgüte unbeschrifteter Beispiele zu schätzen. Nur die Beispiele mit der höchsten Informationsgüte sollen für das Orakel zur Annotation ausgewählt werden, um den Active Learner effizient zu trainieren. Anbei werden die Querystrategien erörtert, die für die Evaluierung im Rahmen dieser Arbeit verwendet werden. [80]

2.6.1 Unsicherheitsbasiertes Sampling

Hierbei wählt das Modell diejenigen Datenpunkte aus, bei denen sich das Modell am unsichersten über die korrekte Klasse ist und aufgrund dieser Unsicherheit bei diesen Datenpunkten nicht in der Lage ist, korrekt zu klassifizieren. Unterkategorien dieses unsicherheitsbasierten Verfahrens sind Least Confidence, Margin Sampling und Entropy Sampling. Diese Unterkategorien werden nachfolgend erläutert. [1]

2.6.1.1 Least Confidence

Bei diesem Verfahren wählt das Modell das Beispiel für das Orakel zur Annotation aus, dessen höchste Klassenwahrscheinlichkeit am geringsten ist. [1]

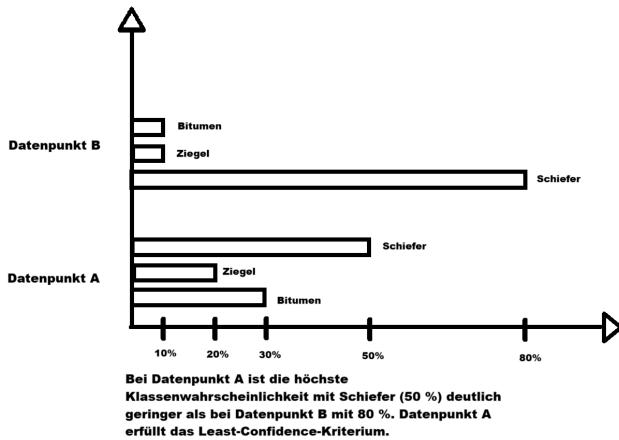


Abbildung 2.2: Visualisierung des Least Confidence Samplings

Die mathematische Definition für Least Confidence lautet:

$$x_{LC}^* = \operatorname{argmax}_x (1 - P(\hat{y}|x))$$

wobei $\hat{y} = \operatorname{argmax}_y P(y|x)$ die wahrscheinlichste Klasse darstellt.

Beispiel für Least Confidence Angenommen, es gibt in unserem Beispiel drei Möglichkeiten für das Modell, einen Datenpunkt zu labeln. Rein fiktiv in unserem Dachmaterialdatensatz zwischen den Möglichkeiten Schiefer, Ziegel und Bitumen als Dächer. Angenommen, unser Modell klassifiziert das Dach mit einer Wahrscheinlichkeit von 60 % als Schieferdach. Zu 20 % könnte es aber auch ein Dach aus Ziegeln oder Bitumen sein. Die höchste Klassenwahrscheinlichkeit beträgt hier 60 %. Ist aber zugleich die niedrigste in diesem Durchlauf, weil sich das Modell bei allen anderen Datenpunkten in dieser Iteration sicherer war. Zum Beispiel, wenn sich das Modell für einen anderen Datenpunkt zu 98 % für Ziegeln entscheidet, ist es sich viel sicherer in Relation zu einem Datenpunkt, in dem die höchstwahrscheinliche Klasse nur 60% beträgt.

$$P(\text{Ziegel}) = 0,33, \quad P(\text{Beton}) = 0,33, \quad P(\text{Schiefer}) = 0,34 \quad (2.1)$$

2.6.1.2 Entropy Sampling

Bei Anwendung dieser Strategie gilt ein Datenpunkt dann als informativ, wenn die vorhergesagten Wahrscheinlichkeiten zur Klassenzugehörigkeit in diesem Datenpunkt gleich verteilt sind. Man spricht in diesem Fall von einer hohen Entropie. [1]

$$P(A) = 0,33, \quad P(B) = 0,34, \quad P(C) = 0,33 \quad (2.2)$$

Beispiel zu Entropy Sampling Angenommen, das Modell prophezeit in unserem Dachmaterialdatensatz eine Klassenzugehörigkeit für einen Datenpunkt von 34 % für Bitumen, 33 % für Ziegel und 33 % für Schiefer als Dach. Somit ist die Entropie sehr hoch und der Datenpunkt wird vom Modell zur Annotation an das Orakel geschickt.

2 Theoretische Grundlagen

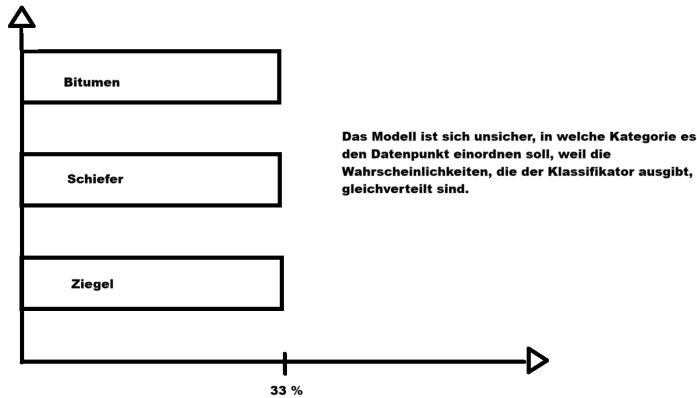


Abbildung 2.3: Visualisierung des Auswahlkriteriums für Entropy Sampling im Balkendiagramm

$$H = - \sum_i P(i) \cdot \log(P(i))$$

Konkret ergibt sich für das obige Beispiel:

$$H = -[0,33 \cdot \log(0,33) + 0,34 \cdot \log(0,34) + 0,33 \cdot \log(0,33)] \approx 1,0986$$

Dieser Wert liegt nahe am Maximum (bei drei Klassen maximal $\log(3) \approx 1,0986$), was die hohe Unsicherheit verdeutlicht.

2.6.1.3 Margin Sampling

Das Modell misst seine Unsicherheit in Bezug auf den Datenpunkt daran, wie knapp es zwischen den beiden wahrscheinlichsten Klassen entscheidet. Bei dieser Differenz in der Vorhersagewahrscheinlichkeit zwischen den beiden wahrscheinlichsten Klassen spricht man auch von Margin. Diese Margin soll möglichst klein sein, um vom Modell ausgewählt zu werden. [1]

$$\text{Margin} = P(\hat{y}_1|x) - P(\hat{y}_2|x)$$

wobei \hat{y}_1 und \hat{y}_2 die beiden wahrscheinlichsten Klassen sind.

Beispiel für Margin Sampling Angenommen, das Modell vergibt in unserem Dachmaterialdatensatz Wahrscheinlichkeiten in einem Datenpunkt, dass das Dach zu 50 % ein Ziegeldach ist, zu 48 % ein Dach aus Bitumen sein könnte und zu 2 % ein Schieferdach darstellen könnte. Hier beträgt die Margin zwischen den beiden wahrscheinlichsten Klassen Ziegel und Bitumen 2 % und der Datenpunkt wird wahrscheinlich zur Annotation an das Orakel weitergeleitet.

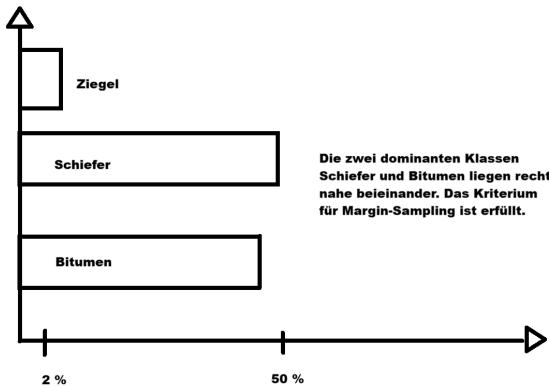


Abbildung 2.4: Visualisierung mit Balkendiagramm anhand welches Kriteriums Margin Sampling Datenpunkte auswählt

2.7 Wilcoxon Signed Rank Test

Der Wilcoxon-Signed-Rank-Test wird im Rahmen der Evaluation dieser Arbeit eingesetzt. Er wird eingesetzt, um paarweise Vergleiche zwischen der passiven Baseline (Random Sampling) und den Active-Learning-Strategien durchzuführen, um herauszufinden, ob die Query-Strategien des aktiven Lernens besser abschneiden als Random Sampling. Der Wilcoxon signed-rank-Test prüft also die Hypothese, dass aktives Lernen bei weniger verwendeten Labels besser abschneiden kann als passives Lernen. Der Vorteil dieses Hypothesentests ist, dass die Daten nicht metrisch und nicht normalverteilt vorliegen müssen. [35]

Beispiel für den Wilcoxon Signed Rank Test in Bezug auf die spätere Evaluation
 Angenommen, die Accuracy wird in Bezug auf Random Sampling zu Margin Sampling unter Verwendung des Klassifikators Random Forest verglichen. Die Berechnung wird in diesem Beispiel dreimal wiederholt und wir erhalten somit drei verschiedene Wertepaare. Die Differenzen der jeweiligen Wertepaare werden berechnet und es wird eine Rangliste erstellt, wo die Differenz am größten ist. Die verwendeten Werte in diesem Beispiel haben keine Bedeutung.

Random Sampling	Margin Sampling	Differences	Ranks
5	16	-11	3
5	3	2	1
5	12	-7	2

Jetzt wird notiert, ob der Rang aus einer negativen oder aus einer positiven Differenz stammt. Dann wird jeweils die Summe der Werte in der Rangspalte, die jeweils aus negativen Differenzen stammen, und der Werte, die aus positiven Differenzen stammen, getrennt gebildet. In diesem Beispiel beträgt die Summe der positiven Ränge 1 und die Summe der negativen Ränge 5. Wenn jetzt hier in dem Beispiel kein Unterschied vorliegt, müssten beide Summen identisch sein. Das entspricht der Nullhypothese. In diesem Fall trifft die Nullhypothese nicht zu, da die Summen unterschiedlich sind. In der späteren Evaluation werden die Experimente fünfmal wiederholt anstatt nur dreimal.

2.8 Der P-Wert

Der P-Wert wird später in der Evaluation darüber aussagen, ob die ermittelten Ergebnisse, ob aktives Lernen besser sein könnte als passives Lernen in Bezug auf Labelaufwand und Accuracy, rein zufällig zustande kamen und womöglich keine Aussagekraft haben. Bei einem P-Wert größer als fünf Prozent sagt man, die Ergebnisse kamen zufällig zustande. Wenn dieser Fall eintritt, müssen wir von der Nullhypothese ausgehen. [26]

2.8.1 Berechnung des P-Werts

Um den P-Wert zu bestimmen, muss erst der Z-Wert berechnet werden. Die Formel für den Z-Wert lautet [26]:

$$z = \frac{\bar{x} - \mu}{\frac{\sigma}{\sqrt{n}}}$$

Dabei ist \bar{x} der Mittelwert der Stichprobe, μ ist der Wert der Nullhypothese, σ ist die Standardabweichung und n repräsentiert die Anzahl der Fälle in der Stichprobe. Nach Einsetzen der Werte erhält man:

$$\mu = \frac{n(n+1)}{4} = \frac{3(3+1)}{4} = 3$$

$$\sigma = \sqrt{\frac{n(n+1)(2n+1)}{24}}$$

$$\sigma = \sqrt{\frac{3(3+1)(2 \cdot 3 + 1)}{24}}$$

$$\delta = 1,87$$

$$z = \frac{1-3}{1,87} \approx -1,069$$

Den p-Wert hierfür lässt sich in einer Tabelle nachschlagen. Es resultiert ca. 0,285 als p-Wert. Der P-Wert in diesem Beispiel ist somit größer als fünf Prozent und es gibt keinen statistischen Grund, die Nullhypothese abzulehnen.

2.9 Bonferroni-Korrektur

Die Bonferroni-Korrektur wird in der Evaluation verwendet, um das Signifikanzniveau alpha von fünf Prozent anzupassen. Um die Bonferroni-Korrektur durchzuführen, wird das gewünschte Signifikanzniveau von fünf Prozent durch die Anzahl der Tests geteilt. Als Ergebnis resultiert das korrigierte Signifikanzniveau für jeden einzelnen Test. Die Bonferroni-Korrektur ist notwendig wegen der Alpha-Fehler-Kumulierung. Das Alpharisko berechnet sich durch [84]:

$$1 - (1 - \alpha)^j$$

Dabei ist j die Anzahl der durchgeführten Tests. Diese Formel gibt das Risiko an, ob ein Test fälschlicherweise signifikant werden könnte. Um dieses Risiko zu minimieren, wird die Bonferroni-Korrektur angewendet.

2.10 Effektstärke

Die Effektstärke gibt an, wie stark der beobachtete Effekt ist. In diesem Fall, ob aktives Lernen deutlich besser ist als Random Sampling. Der Effekt steht also in diesem Fall

für den Unterschied. Die Effektstärke wird berechnet, indem der z-Wert des Wilcoxon-Tests durch die Quadratwurzel der Stichprobengröße geteilt wird. Werte, die sich der Null annähern, werden als geringe Effektstärke interpretiert, während Werte, die sich minus eins oder eins annähern, auf eine hohe Effektstärke hindeuten. Die Effektstärke gibt somit die Stärke des Unterschieds an, während der p-Wert lediglich aussagt, dass die beobachtete Messung kein Zufall ist. Die Effektstärke ist in dieser Arbeit von Nutzen, um zu prüfen, ob die Implementierung aktiven Lernens in der Praxis gegenüber passivem Lernen eine spürbare positive Veränderung in Bezug auf Labelaufwand und Accuracy bewirkt. Eine geringe Effektstärke würde zur Nullhypothese führen. [41].

Nachdem die Methoden zur statistischen Auswertung der Ergebnisse erläutert wurden, werden nun die Klassifikatoren vorgestellt, deren Leistung im Rahmen dieser Arbeit evaluiert wird.

2.11 Verwendete Klassifikatoren

Dieser Abschnitt erläutert die Funktionsweise der in der Evaluierung verwendeten Klassifikatoren. Zudem gibt es eine Erwartung zu den Ergebnissen, die der jeweilige Klassifikator auf den Datensätzen MNIST, Fashion-MNIST und dem Dachmaterialiedatensatz erzielen wird.

2.11.1 Random Forest

Random Forest basiert auf einem Ensemble von Entscheidungsbäumen. Ein Entscheidungsbaum arbeitet mit diskreten Werten. Zum Beispiel, ob der Wert größer oder gleich sieben ist. Wenn das nicht der Fall ist, wird der Wert zum entsprechenden Ast in dem Baum weitergeleitet. Die Daten werden also durch das interne „Wenn Dann“ immer weiter aufgeteilt. Man kann dadurch anschaulich betrachten, wie eine Entscheidung zustande kommt. Der Nachteil der Entscheidungsbäume ist, dass sie nur diskrete Durchschnittswerte vorhersagen können. Sie bieten keine feinen Abstufungen, um kontinuierliche Werte vorhersagen zu können. Der Baum müsste für feine Abstufungen extrem tief sein, was zu Overfitting führt. Das heißt, das Modell lernt einfach nur die Daten auswendig. Das Modell kann schlecht auf neue, unbekannte Daten in so einem Fall generalisieren. Ein Entscheidungsbaum hat eine hohe Modellvarianz. Das heißt, dass sich ein Entscheidungsbaum aufgrund der Stichprobe, die sich jedes Mal unterscheidet, ganz verschieden entwickelt und somit aufgrund der unterschiedlichen Daten in der Stichprobe jedes Mal stark abweichende Vorhersagen für dieselbe Kategorie von Daten liefern wird. Der Random Forest löst dieses Varianzproblem, indem ein ganzer Wald von unterschiedlichen Bäumen anstatt nur ein Baum implementiert wird. Durch Bootstrap-Aggregating werden die Bäume verschieden durch Bagging. Hierbei wird für jeden Baum eine Stichprobe aus den Originaltrainingsdaten gezogen, mit Zurücklegen. Die Out-of-bag-Samples, die in keinem Baum landen, werden zum Testen des Modells verwendet. Außerdem variieren im Random Forest auch die Merkmale, die die einzelnen Bäume betrachten. Der einzelne Baum darf nicht alle vorhandenen Merkmale in max. features prüfen, sondern er bekommt eine zufällig ausgewählte Teilmenge aus allen verfügbaren Merkmalen vorgegeben. Das macht die Bäume diverser und weniger korreliert untereinander. Der Random-Forest-Algorithmus nimmt den Durchschnittswert seiner einzelnen Bäume als Vorhersage. Random-Forest-Modelle können nicht extrapolieren. Das heißt, sie können keine Werte vorhersagen, die weit außerhalb des Bereichs der Daten liegen, die sie in dem Training gesehen haben. [5]

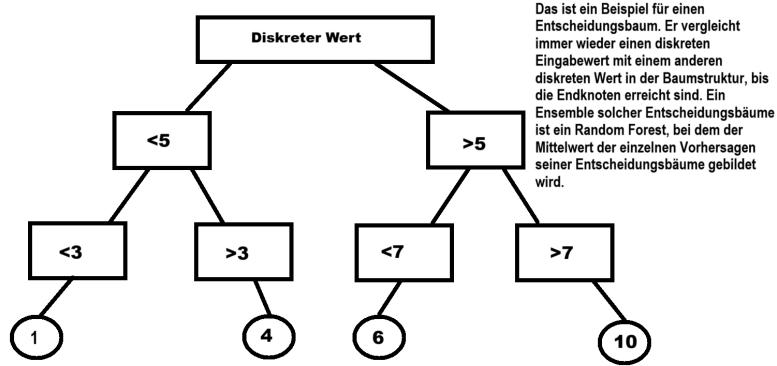


Abbildung 2.5: Visuelle Darstellung eines einzelnen Entscheidungsbaums

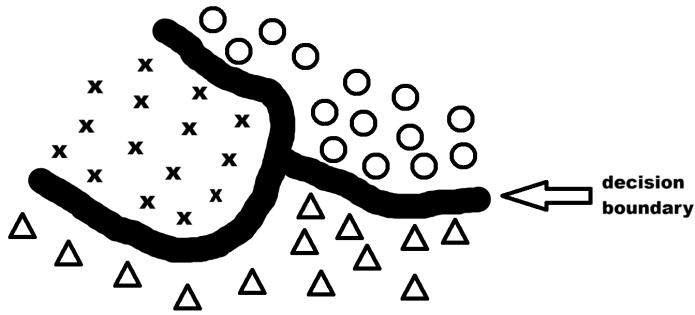
Wahrscheinlichkeitsberechnung: Die Wahrscheinlichkeit einer Klasse k ergibt sich als Anteil der Bäume, die k vorhersagen:

$$\hat{P}(Y = k|x) = \frac{1}{T} \sum_{t=1}^T \mathbb{1}\{\text{Baum}_t(x) = k\}$$

Erwartung von Random Forest in Bezug auf Performance auf den Datensätzen Erwartung für Random Forest auf den Datensätzen: Aufgrund der Mittelwertbildung in den Endknoten wird erwartet, dass Random-Forest-Modelle häufig vertretene Klassen in unbalancierten Datensätzen gut klassifizieren. Unterrepräsentierte Klassen werden jedoch möglicherweise vernachlässigt, was die Praxistauglichkeit einschränkt. Trotz erwarteter Überlegenheit gegenüber anderen Klassifikatoren auf dem Dachmaterialdatensatz bleibt die Gesamtleistung aufgrund der Datenqualität limitiert. Zusätzliche Features wie Spektraldaten könnten die Modellleistung verbessern. Auf den balancierten Datensätzen MNIST und Fashion-MNIST wird eine mittlere Performance erwartet. [108]

2.11.2 SVM

Support-Vector-Machines verfolgen das Ziel, die beste Trennlinie zwischen verschiedenen Klassen zu finden. Dies kann auch in höheren Dimensionen auf einer Hyperebene stattfinden. Die beste Trennlinie ist gefunden, wenn sie den maximalen Abstand zu den nächsten Punkten der zu klassifizierenden Klassen hat. Ein Punkt kann klassifiziert werden anhand dessen, auf welcher Seite der Trennlinie der Datenpunkt eingruppiert wird. Eine SVM arbeitet effizient. Es ignoriert die meisten Datenpunkte, weil es sich auf die Support-Vektoren konzentriert. Das sind die Datenpunkte, die den geringsten Abstand zur Trennlinie haben. Das spart Rechenaufwand. Um die optimale Trennlinie zu finden, wird ein Optimierungsverfahren eingesetzt. Zum Beispiel die Lagrange-Methode, bei der jeder Datenpunkt ein Wichtigkeitsgewicht bekommt. Die Gewichte sind für die meisten Punkte null. Nur die Stützvektoren haben ein Gewicht größer Null. Aus den Gewichtungen und den Vektoren wird die Trennlinie ermittelt. Mittels dualem Problem werden die Wichtigkeitsgewichte (Alpha) ermittelt, um die erforderliche Rechenleistung zu verringern. Anstatt direkt die Trennlinie zu zeichnen, werden erst einmal die Alpha-Gewichte ermittelt, indem eine Funktion maximiert wird, die auf den Ähnlichkeiten basiert. In diesem Fall dem Skalarprodukt zwischen Paaren von Datenpunkten. Diese Vorgehensweise



Ziel des SVM-Modells (Support-Vector-Machine) ist es, eine decision boundary, auf Deutsch Trennlinie, zwischen den verschiedenen Klassen x, o und Δ zu zeichnen.
Hier konnte keine Gerade als Trennlinie entstehen, weshalb der Kernel-Trick angewandt wurde.

Abbildung 2.6: Stellt visuell dar wie eine Support Vektor Machine eine nichtlineare decision boundary mithilfe des Kerneltricks einzeichnen kann

ist aufgrund von Ausreißern und Überlappungen selten perfekt, weshalb Softmargin eingesetzt wird. Hierbei wird dem Modell eine Toleranz eingeräumt. Das bedeutet, dass einige wenige Datenpunkte sich auf der falschen Seite der Trennlinie befinden dürfen. Für jeden falsch eingruppierten Datenpunkt wird das Modell bestraft. Wie stark das Modell bestraft wird, wird mittels Strafparameter gesteuert. Ein hoher Strafparameter birgt das Risiko für Overfitting, während ein zu kleiner Strafparameter die Qualität der Vorhersagen zu stark beeinträchtigt. Softmargin macht SVM-Modelle ergo robuster. Bei komplexen Klassifikationsaufgaben, bei denen sich keine Gerade als Trennlinie ziehen lässt, kommt der Kernel-Trick zum Einsatz. Das bedeutet, dass die Daten in eine höhere Dimension transformiert werden, wenn sie in unserer Dimension nicht trennbar sind. Zum Beispiel wenn Datenpunkte auf einem Blatt Papier nicht trennbar sind, aber trennbar werden, wenn ich einige Datenpunkte in die dritte Dimension anhebe. Der Kernel-Trick vermeidet diese Transformation aufgrund des Rechenaufwands, der sich daraus ergibt. Stattdessen wird das Skalarprodukt mithilfe der Originaldaten berechnet, als ob die Punkte im höheren Raum wären. Somit kommen die Vorteile der Transformation auf einer höheren Ebene zum Vorschein, ohne den hohen Rechenaufwand für die Transformation bewältigen zu müssen. Es gibt verschiedene Arten von Kernels, je nachdem, welche Trennlinie ich brauche. Zum Beispiel RBF, linear und polynomial. [71]

Die Wahrscheinlichkeiten werden mittels Platt-Skalierung berechnet:

$$\sigma(f(x)) = \frac{1}{1 + \exp(A \cdot f(x) + B)}$$

wobei A und B durch Maximum-Likelihood auf einem Validierungsdatensatz bestimmt werden.

Erwartung von SVM in Bezug auf Performance auf den Datensätzen Die Performance vom SVM auf den Datensätzen MNIST und Fashion-MNIST wird davon abhängen, ob der geeignete Kernel gewählt wird und der Strafparameter korrekt eingestellt wird. Das SVM-Modell wird auf den Datensätzen MNist und Fashion-MNist gut performen, weil die Datensätze gut ausbalanciert sind, und auf den Dachmaterialdatensatz in Relation zu den anderen Klassifikatoren schlecht performen, weil aufgrund der geringen Güte des Datensatzes der Strafparameter zu hoch eingestellt werden muss, um die Robustheit gegenüber unsauberen Daten zu erhöhen. Dies wiederum beeinträchtigt die Genauigkeit

2 Theoretische Grundlagen

der Vorhersagen, die das Modell generieren wird. Auf allen Datensätzen wird der RBF-Kernel gewählt, weil dieser kurvige Trennlinien zeichnen kann. Der lineare Kernel wäre zu einfach für Bilder. Außerdem wird ein Strafparameter von zehn gewählt, als Kompromiss, um Überanpassung und Fehler beim Training zu vermeiden. [107]

2.11.3 Multinomiale Logistische Regression

Diese Methode ist eine Erweiterung der binären logistischen Regression, die für zwei Zielvariablen ausgelegt ist. Mithilfe der multinomialen logistischen Regression können mehrere Klassen klassifiziert werden, was sich im Kontext dieser Arbeit eignet. Die zu verarbeitenden Kategorien haben keine natürliche Reihenfolge wie „niedrig“, „mittel“, „hoch“. Das Modell betrachtet die Unterschiede zwischen den Klassen mithilfe der Referenzkategorie, die als Vergleichsmaßstab dient. Alle anderen Klassen werden immer in Referenz zu diesem Vergleichsmaßstab kategorisiert. Also Kategorie eins gegen Referenz, Kategorie zwei gegen Referenz etc. Das Modell schätzt erst mal die Beta-Koeffizienten, die angeben, wie sehr sich die Log-Odds verändern. Das ist eine interne Rechengröße des Modells. Man rechnet diese Betas um. Man exponentiert sie und rechnet die eulersche Zahl hoch Beta. Das Ergebnis davon ist dann das Odds Ratio. Diese gibt die Chance in Referenz zum Vergleichsmaßstab an, in einer bestimmten Kategorie zu landen. Dabei ist die Chance noch keine Wahrscheinlichkeit. Ich muss prüfen, wie gut das Modell zu meinen Daten passt. Dafür gibt es Goodness-of-Fit-Tests. Außerdem muss ich mir anschauen, ob die Odds Ratios statistisch signifikant sind. Das sehe ich an den Konfidenzintervallen. Für die Betas darf das Intervall die Null nicht einschließen und für die Odds Ratios darf das Intervall die Eins nicht einschließen. Wenn die Eins enthalten ist, ist der Effekt nicht signifikant auf dem gewählten Niveau. Das bedeutet, die Eins ist der kein Unterschiedspunkt auf der gewählten Odds Ratio. [104] Die Wahrscheinlichkeit für Klasse k bei K Klassen:

$$P(Y = k|x) = \frac{\exp(w_k^T x + b_k)}{\sum_{j=1}^K \exp(w_j^T x + b_j)}$$

Erwartung von Multinomiale Logistische Regression in Bezug auf Performance auf den Datensätzen Entscheidend ist, wie die Daten, die das Modell ausgibt, im Kontext von MNIST, Fashion-MNIST und dem Dachmaterialdatensatz interpretiert werden. Es muss verstanden werden, was ein Odds-Ratio von 1,27 bedeutet, wenn das Modell einen solchen Wert zurückgibt. Es gibt in diesem Kontext mehr Spielraum für Interpretationsfehler, als es bei Klassifikatoren wie Random Forests der Fall wäre, die out of the box schon gut funktionieren. Bei dem unbalancierten Datensatz wird davon ausgegangen, dass Minderheitsklassen ignoriert werden und das Modell generell zur Mehrheitsklasse tendiert. Auch ist denkbar, dass lineare Grenzen nicht ausreichen werden auf dem Fashion-MNIST-Datensatz. Das Modell wird wahrscheinlich auf sich selbst bezogen auf dem MNIST-Datensatz am besten performen, weil die Ziffern relativ einfach unterscheidbar sind und die lineare Trennung relativ gut funktioniert. [62]

2.11.4 Neuronales Netz

Vorausgesetzt sei ein Schwarz-Weiß-Bild als Beispiel. Die Pixelhelligkeiten werden in Werte übersetzt. Minus eins für schwarz und eins für weiß. Diese Zahlen sind der Input für die allererste Schicht von Neuronen. Jedes Neuron bekommt einen dieser Werte. Die Neuronen im nächsten Layer haben Gewichte. Jeder Wert im Inputlayer wird mit einem

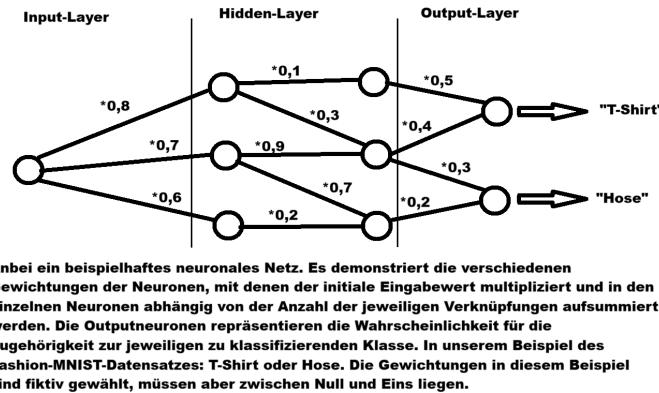


Abbildung 2.7: Verdeutlicht wie ein neuronales Netz intern zur Problemlösung unserer Klassifikationsaufgabe arbeitet.

festgelegten initialen Gewicht im kommenden Hiddenlayer multipliziert. Die resultierenden Produkte werden im Neuron des Hiddenlayers aufsummiert. Die Anzahl der Summanden im jeweiligen Hiddenneuron entspricht der Anzahl der Verbindungen zu verschiedenen Neuronen im Inputlayer. Die gebildete Summe wird zusätzlich in eine Aktivierungsfunktion geleitet, wie die Sigmoid-Funktion, um den Wert in einen festen Bereich von minus Eins und plus Eins zu bringen. Dadurch wird kein Wert fälschlicherweise übergewichtet. Das Prozedere wiederholt sich über mehrere Hiddenlayer, bis der Outputlayer erreicht wurde. Dabei entstehen immer größere rezeptive Felder. Das heißt, dass die Neuronen mit jedem Layer einen immer größeren Bereich der Eingabewerte sehen. Bei unserer Klassifikationsaufgabe soll im Outputlayer nur das Neuron feuern, das beispielsweise im MNIST-Datensatz für eine bestimmte Zahl steht. Wir haben also zehn Outputneuronen für die Zahlen eins bis zehn. Die Gewichte im neuronalen Netz kommen aus dem Training, in dem tausende Beispiele gezeigt werden. Wenn das neuronale Netz im Training Fehler macht, werden die Differenzen der Fehler zu der richtigen Antwort ermittelt und mittels Backpropagation zurück ins Netz geschickt. Dabei wird ausgerechnet, wie stark die Gewichtungen der einzelnen Neuronen zu diesem Fehler beigetragen haben. Anschließend werden die betreffenden Gewichtungen ein klein wenig angepasst, um die Modellleistung zu verbessern. Dabei spielt die Lernrate eine entscheidende Rolle. Ist die Lernrate zu klein gewählt, lernt das Netz ewig. Ist die Lernrate zu groß, schießt das Modell über das Ziel hinaus. [92]

Ausgabeschicht verwendet Softmax-Aktivierung:

$$\sigma(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^K \exp(z_j)}$$

Erwartung von Neuronalen Netzen in Bezug auf Performance auf den Datensätzen

Denkbar ist, dass das neuronale Netz bei den ausbalancierten Datensätzen MNIST und Fashion-MNIST sehr gut performen wird, weil es auch räumliches Denken beinhaltet, was sonst kein Klassifikator berücksichtigt. Es kann sehr granulare Details der Bilddaten aufnehmen, um präzise Vorhersagen zu generieren. Bei dem unbalancierten Dachmaterialdatensatz können die unbalancierten Daten zu einem Bias für häufige Klassen führen, weshalb anzunehmen ist, dass das Modell für diesen konkreten Datensatz aufgrund der Qualität des Datensatzes nicht praxistauglich sein wird. [83]

2.11.5 Naïve Bayes

Bei diesem Klassifikator geht es darum, Dinge aufgrund von Wahrscheinlichkeit in Kategorien einzuteilen. Dem Naïve-Bayes-Klassifikator liegt das Bayes-Theorem zugrunde.

Angenommen, es gibt eine Hypothese H. Zum Beispiel: Eine E-Mail ist Spam. Dann bekomme ich neue Daten D. Das sind die Wörter in der Mail. Das Bayes-Theorem sagt aus, wie wahrscheinlich die Hypothese H jetzt ist, nachdem die Daten D gesehen wurden. [59]

$$P(H|D) = \frac{P(D|H) \cdot P(H)}{P(D)}$$

Am E-Mail-Beispiel erklärt:

- $P(H|D)$ ist die Wahrscheinlichkeit, dass die Mail Spam ist, wenn diese Wörter gesehen wurden.
- $P(H)$ ist die ursprüngliche Einschätzung. Also: Wie wahrscheinlich war Spam, bevor die Mail angesehen wurde?
- $P(D|H)$ ist die Wahrscheinlichkeit, genau diese Wörter zu sehen, wenn die Mail tatsächlich Spam ist.
- $P(D)$ ist die allgemeine Wahrscheinlichkeit, diese Kombination von Wörtern überhaupt zu sehen.

Wahrscheinlichkeitsberechnung: Für jede Klasse wird berechnet:

$$P(C = k | x) = \frac{P(C = k) \prod_j P(x_j | C = k)}{\sum_l P(C = l) \prod_j P(x_j | C = l)}$$

Diese Formel ergibt sich direkt aus dem Bayes-Theorem. Im E-Mail-Beispiel wäre $C = \text{Spam}$ oder $C = \text{Kein Spam}$, und x_j sind die einzelnen Wörter in der E-Mail. Die "naïve Annahme ist, dass die Wörter unabhängig voneinander auftreten. Wahrscheinlichkeiten sind kalibriert, aber sensitiv gegenüber Abhängigkeitsverletzungen[Gohari2023].[59]

2.12 Konfusionsmatrix

Die prognostizierten Werte eines Klassifikators lassen sich anhand einer solchen binären Konfusionsmatrix wiedergeben. Hierbei werden vier Kategorien unterschieden [48, 85]:

- Richtig negative: Das sind Werte, die vom Modell als negativ klassifiziert werden. Diese Vorhersagen sind korrekt.[48, 85]
- Falsch negative: Das sind Werte, die vom Modell als negativ klassifiziert werden. Diese Vorhersagen sind jedoch nicht korrekt.[48, 85]
- Falsch positive: Das sind Werte, die vom Modell als positiv klassifiziert werden. Diese Vorhersagen sind nicht korrekt.[48, 85]
- Richtig positiv: Das sind Werte, die vom Modell als positiv klassifiziert wurden. Diese Vorhersagen sind korrekt.[48, 85]

Auf den Hauptdiagonalen einer binären Konfusionsmatrix liegen die richtig positiv klassifizierten Werte und die richtig negativ klassifizierten Werte. Diese Werte sollen maximiert werden. Zudem lässt sich eine Mehrklassenklassifikation in solch eine binäre Konfusionsmatrix aufteilen, um auf deren Basis Gütemaße wie Accuracy zu erhalten. Die Accuracy, auch Korrektklassifizierungsrate genannt, misst die Fähigkeit des Klassifikators, Datenpunkte der richtigen Klasse zuzuordnen.[48, 85]

$$KKR = \frac{RP+RN}{FP+FN+RP+RN}$$

2.13 Klassifikationsproblem

Bei der Klassifikation sortiert das Modell Datenpunkte in mehrere Kategorien. Diese Kategorien können beispielsweise Äpfel und Birnen sein. Das Modell versucht also in diesem Fall, Bilder von Obst nach Äpfeln oder Birnen zu sortieren. Es liegt also ein binäres Klassifikationsproblem vor. [68]

2.14 Overfitting und Underfitting

Angenommen, ein Algorithmus lernt durch Ausprobieren wie ein Kind. Angenommen, der Babyalgorithmus hat noch nie einen Apfel gegessen und merkt: Der grüne Apfel ist sauer. Daraus schlussfolgert der Algorithmus, dass alle Äpfel sauer sind, weil es noch keinen Vergleich hat. Dann wird ein kleiner roter Apfel probiert, der wiederum süß schmeckt. Das Modell führt eine Regel ein, die heißt: „Grüne Äpfel sind sauer und rote Äpfel sind süß.“ Algorithmus probiert ganz viele Äpfel mit der Erkenntnis, dass die Regel zu stimmen scheint. Das Modell soll generalisieren können. Angenommen, das Modell stößt auf eine Ausnahme: auf einen sauren roten Apfel. Anstatt zu generalisieren, dass rote Äpfel meistens süß sind, sucht das Modell nach einer komplexeren Regel, um diese eine Ausnahme zu erklären. Wenn das Modell nicht gut generalisiert und versucht, alle Anomalien anhand von Regeln zu erklären, sprechen wir von Overfitting. Technisch gesehen passiert Overfitting, wenn das Modell zu komplex ist für die Daten. Wenn es zu viele Regler und Freiheitsgrade besitzt. Das Modell versucht dann, alles zu erklären. Es kann Muster sehen, wo gar keine sind. Underfitting bedeutet, das Modell ist zu simpel. In dem Fall scheint es nicht mal das grundlegende Muster in den Daten zu erkennen. In dieser Analogie kann das Modell, wenn Underfitting eintritt, die Äpfel gar nicht unterscheiden. In diesem Fall hat es einen hohen Bias. Das bedeutet, es hat eine starke Voreingenommenheit durch die eigene Einfachheit des Modells gegenüber den Daten und ignoriert die enthaltenen Informationen in den Daten. Das ist das Bias-Variance-Dilemma. Wohingegen also Overfitting auf hohe Varianz hindeutet. Das Modell reagiert zu stark auf spezifische Trainingsdaten. Das Ziel ist das Mittelmaß zwischen Under- und Overfitting. Dafür gibt es Techniken wie Kreuzvalidierung, Regularisierung etc. [20, 73]

2.15 Effekte unbalancierter Datensätze auf die Klassifikationsleistung der Machine Learning Modelle

Ca. 80 Prozent aller vorliegenden realen Klassifikationsdatensätze weisen eine Form von Unbalanciertheit auf. Das führt zu einem Bias hin zur Mehrheitsklasse. Das Modell lernt, dass es meistens richtig liegt, wenn es einfach die am häufigsten vorkommende Klasse vorhersagt, was die Genauigkeit auf dem Papier maximiert. Es kommt zum Accuracy-Paradoxon. Das Modell ignoriert die Minderheitsklassen und kann trotzdem eine Accuracy von neunundneunzig Prozent erreichen. Wenn beispielsweise im Bilddatensatz neunzig

2 Theoretische Grundlagen

Prozent Jacken sind und nur zehn Prozent Hosen, dann wird es immer Jacken vorhersagen und eine Accuracy von neunzig Prozent erzielen. Das ist kritisch zu betrachten, denn die Daten sind unterrepräsentiert und sind meistens relevant, wie Systemausfälle, Krankheiten etc. Die Modelle können deren Muster nicht lernen aufgrund mangelnder Daten. Wir können in diesem Fall die Klassifikationsleistung des Modells nicht durch Accuracy bestimmen, sondern müssen auf Precision und Recall zurückgreifen. Precision gibt an, wie viele es wirklich waren von denen, die erkannt wurden. Recall gibt an, von denen, die tatsächlich selten waren, wie viele gefunden wurden. Diese Metriken werden im F1-Score kombiniert oder es wird die balanced accuracy verwendet, die die Leistung über alle Klassen mittelt. Lineare Modelle sind am häufigsten vom Accuracy-Paradoxon betroffen, weil diese selten vorkommende Datenpunkte nicht einfach isolieren können. Es können künstlich mehr Beispiele der seltenen Klasse erzeugt werden, um dem entgegenzuwirken. Ein Verfahren hierfür ist SMOTE. Man kann auch Undersampling betreiben, indem die Mehrheitsklasse reduziert wird, um den Datensatz auszubalancieren. Bei SMOTE sind die künstlich generierten Datenpunkte nicht immer realistisch, während man bei Undersampling informative Datenpunkte weg wirft. Eine weitere Strategie ist cost-sensitive learning, bei dem man dem Modell sagt, dass ein Fehler bei der Minderheitsklasse schwerer wiegt als bei einer Mehrheitsklasse. Zusammengefasst sind unausgewogene Daten ein weit verbreitetes Problem im Machine Learning. Sie führen zu einer trügerischen Genauigkeit, während zugleich seltene Ereignisse übersehen werden. [33, 106]

2.16 Batch Active Learning und Sequencial Active Learning

Beim sequenziellen Ansatz geht man schrittweise vor. Man wählt ein Label aus, bekommt die Info, was das ist, lernt daraus und wählt erst im Anschluss darauf den nächsten Punkt aus, unter Berücksichtigung des gerade Gelernten. Der sequenzielle Ansatz ist sehr anpassungsfähig. Nach jedem neuen Label kann die Strategie angepasst werden. Beim Badge-Active-Learning hingegen wird ein ganzer Schwung von Daten gleichzeitig ausgewählt. Die Auswahl kann daher nicht ganz so zielgerichtet sein wie beim sequenziellen Vorgehen. Die Frage, welche Methode vorzuziehen ist, liegt in der Praktikabilität und der Zeit. Sequenziell ist deutlich langsamer, weil man auf das eine Label warten muss, bevor der nächste Schritt kommt. Nach jedem Label wird das Modell neu trainiert, was sehr rechenintensiv ist. Rein von der Leistung ist das sequenzielle Vorgehen nie schlechter als der beste Badgeansatz. Redundanz ist beim Badgeansatz ein Thema. Wenn die k-informativsten Beispiele ausgewählt werden, ist es gut möglich, dass diese Beispiele sich sehr ähneln. In der Evaluation dieser Arbeit werde ich den batchbasierten Ansatz verwenden, weil es ressourcenschonender ist. [32, 70]

2.17 QGIS

Im Grunde ist QGIS ein vielseitiges Werkzeug für Geodaten, was Straßen, Städte, Flüsse, Bevölkerungsdichte etc. enthalten kann. Es ist ein digitales Schweizer Taschenmesser für Landkarten und räumliche Analysen. Zudem ist es Open Source und ist kostenlos. Solche Systeme sind sehr relevant, beispielsweise in der Stadtplanung, im Umweltschutz oder in der Logistik. QGIS ist weit verbreitet und läuft auf Windows, Mac, Linux und Android. Gestartet ist das Projekt 2002 und hieß früher Quantum GIS. Damit kann man Daten anschauen, analysieren und Karten gestalten. Es gibt achthundert eingebaute Werkzeuge. Es beherrscht 2D- und 3D-Ansichten von Gebäuden. Abfragen wie „Finde alle Schulen im Umfeld von dieser Straße“ lassen sich mit QGIS bewältigen. Die Chancen stehen zudem

gut, aufgrund der großen Community, dass bereits ein Plugin entwickelt wurde, sollte eine gewünschte Funktion in QGIS nicht standardmäßig verfügbar sein. Es wächst mit den Anforderungen. Beispielsweise lässt sich QGIS mit POSTGIS verbinden, einer Datenbank, um riesige Datenmengen zu verwalten. Es wurde in vierzig Sprachen übersetzt. Über 900 wissenschaftliche Publikationen, die QGIS nutzen, mit einer Wachstumsrate von jährlich 40 Prozent in Bezug auf die Community. Es ist ein Werkzeug, das die Arbeit mit räumlichen Informationen durch die große Community und ständige Weiterentwicklung demokratisiert. [52, 75]

2.18 Evaluationskriterien und erwartete optimale Kombinationen

Active Learning zielt darauf ab, durch gezielte Auswahl informativer Instanzen die Effizienz des Lernprozesses zu maximieren. In diesem Abschnitt werden zentrale Evaluationskriterien betrachtet und auf Basis des aktuellen Forschungsstands Klassifikator-Strategie-Kombinationen vorgestellt, die sich in der Literatur als vielversprechend erwiesen haben. Es handelt sich hierbei um hypothetisch optimale Kombinationen, die in der Praxis validiert werden sollten.

2.18.1 Literaturbasierte Kombinationen für ausgewählte Evaluationskriterien

Kriterium	Vorgeschlagene Kombination	Begründung laut Literatur
Anzahl gelabelter Instanzen	SVM + Uncertainty Sampling	SVMs profitieren stark von Instanzen nahe der Entscheidungsgrenze, welche durch Uncertainty Sampling effizient identifiziert werden können [38, 69].
Genauigkeit (Accuracy)	Random Forest + Query-by-Committee (QBC)	Die natürliche Diversität der Random-Forest-Bäume bietet eine geeignete Basis für QBC, das instabile Instanzen mit hoher Informationsdichte selektiert [38, 69].
F1-Score	SVM + Hybrid (Uncertainty + Diversity)	Die Kombination berücksichtigt sowohl Unsicherheit als auch Repräsentativität, was besonders bei unausgeglichenen Klassenverteilungen zu besseren Ergebnissen führt [101, 103].
Lernkurve	CNN + Expected Error Reduction (EER)	Die Auswahl von Instanzen mit maximal erwarteter Fehlerreduktion beschleunigt das Lernen bei komplexen Modellen wie CNNs [38, 105].
Annotierungszyklen	Random Forest + Batch QBC	Durch Batch-Auswahl mehrerer diverser Instanzen pro Zyklus werden die benötigten Annotierungsrunden reduziert [38].

Kriterium	Vorgeschlagene Kombination	Begründung laut Literatur
Trainingszeit	Naive Bayes + Uncertainty Sampling	Naive Bayes ist besonders schnell trainierbar, und Uncertainty Sampling lässt sich effizient darauf anwenden [69].
Auswahldiversität	Random Forest + Hybrid (QBC + Diversity + Density)	Diversitäts- und Dichtekriterien vermeiden redundante Beispiele und sichern eine gute Merkmalsraumabdeckung [38].

Anzahl gelabelter Instanzen

Die Anzahl der manuell von Expert:innen annotierten Beispiele fungiert als Maßstab für den Aufwand, der für die Annotation im Active-Learning-Prozess erforderlich ist [64].

Genauigkeit (Accuracy)

Der Anteil der richtig klassifizierten Fälle von allen überprüften Fällendaten gibt eine einfache Gesamteinschätzung des Klassifikators. Es ist jedoch anfällig für Verzerrungen bei ungleicher Klassenverteilung [72].

F1-Score

Das harmonische Mittel von Genauigkeit und Erinnerung ist ein Maß für den Kompromiss zwischen Fehlalarm und Auslassung. Es quantifiziert die Balance und ist besonders hilfreich bei ungleicher Klassenverteilung [72].

Lernkurve

Grafische Darstellung zeigt die Leistung des Modells (wie Genauigkeit oder F1-Wert), abhängig von der Anzahl der gelabelten Trainingsdaten und verdeutlicht den Effizienzzuwachs durch Active Learning sowie die Tendenz zur Stabilisierung des Modells [61].

Annotierungszyklen

Einzelne Durchläufe des Active-Learning-Prozesses umfassen das Training und die Abfrage neuer Instanzen sowie deren Annotation. Die Anzahl der Zyklen spiegelt den realen Koordinationsaufwand wider [64].

Zeitlicher Aufwand

Der zeitliche Aufwand für das Training des Modells oder einen vollständigen Active-Learning-Schritt wird verwendet, um die rechnerische Effizienz verschiedener Strategien zu bewerten [64].

Auswahldiversität

Maß für die Heterogenität der in einem Batch gewählten Instanzen; hohe Diversität verringert Redundanz und verbessert die Repräsentativität des gelabelten Datensatzes [64].

2.18.1.1 Kontextabhängige Empfehlungen

Die nachfolgend genannten Kombinationen sind als literaturgestützte Vorschläge zu verstehen und sollten je nach Anwendungsszenario individuell validiert werden:

- **Begrenzte Rechenressourcen:** Naive Bayes + Uncertainty Sampling. [44]

- **Unausgeglichene Daten:** SVM + Hybridstrategie. [89]
- **Hohe Genauigkeit:** Random Forest + QBC (+ Diversität). [86]
- **Begrenztes Annotationsbudget:** SVM + Uncertainty Sampling. [10]

2.19 Dachmaterialklassifikation mittels CNNs in Luftbildern

Informationen über Dächer sind wichtig für Entscheidungen am Boden, die die Allgemeinheit betreffen. Es lassen sich 3 Merkmale aus den Luftbildern manuell extrahieren. Zum einen die spektralen Merkmale. Dazu gehören die Farben Rot, Grün und Blau, aber auch das Nahinfrarotlicht, welches für Menschen nicht sichtbar ist. Nahinfrarotlicht wird jedoch von unterschiedlichen Materialien ganz anders reflektiert, was für gute Zusatzinformationen sorgt. Als zweiten Punkt haben wir Texturmerkmale. Hier geht es darum, ob die Oberflächentextur des Materials rau, glatt oder beispielsweise ein Muster aufweist. Hierfür gibt es Methoden wie die Gray-Level Co-occurrence Matrix (GLCM). Diese misst, wie oft bestimmte Grautöne nebeneinander vorkommen, um Rauheit zu messen. Oder man nutzt Kantendetektoren wie den Gabor-Filter, um Linien und Kanten auf dem Dach zu finden. Die dritte Ebene sind die geometrischen Merkmale, welche die Form vom Dach, Neigung, Größe sowie Ausrichtung beinhalten. Dieses Merkmal allein gibt bereits Auskunft über den Gebäudetyp, beispielsweise ob es sich um ein Lagerhaus handelt.

CNNs lernen nicht nur die Merkmale selber, sondern auch die komplexen Zusammenhänge dazwischen. Dazu zählen Muster, die Menschen oft gar nicht sehen oder beschreiben könnten, wodurch die Genauigkeit der Klassifikation enorm anstieg. ResNet-50 und ResNet-152 zeigen hervorragende Leistung bei der Klassifikationsaufgabe mit Genauigkeiten von 89,3 % bzw. 91,2 % während U-Net und Mask R-CNN eignen sich besonders für Segmentierungsaufgaben und erreichen Genauigkeiten von über 92 %. Segmentierung heißt, die sagen nicht mehr nur die Klasse „Ziegel“ voraus, sondern diese neuronalen Modelle können pixelgenau die Fläche auf dem Dach erkennen, die aus Ziegeln besteht, wodurch eine höhere Detailtiefe erreicht wird. Das wird durch einen hierarchischen Aufbau bewerkstelligt. Dazu gibt es Konvolutionsschichten, die nach kleineren Mustern Ausschau halten, wie Kanten und Ecken. In tieferen Schichten lernen diese Modelle, daraus komplexere Schichten zusammenzusetzen. Beispielsweise Ziegelmuster und Lüftungsschächte. Pooling-Schichten verdichten die Info und machen die Modelle robuster gegenüber kleinen Verschiebungen. Ganz am Ende entscheiden die Fully Connected Layers basierend auf allem, was erkannt wurde, welches Material vorliegt. Moderne Ansätze nutzen Attention-Mechanismen, welche dem neuronalen Netz helfen, sich auf die wichtigen Bildteile zu konzentrieren. Herausforderungen bleiben Beleuchtung und Schatten. Der Fotograf weiß, dass je nach Beleuchtung dieselbe Fläche anders aussieht. Die zweite Herausforderung stellt die Menge an hochauflösenden Daten dar, die von Experten beschriftet werden müssen. Der Datensatz muss also von hoher Qualität sein, damit das CNN die anderen Klassifikationsmodelle outperformen kann. Zudem brauchen Convolutional Neural Networks sehr viel Rechenpower, wodurch der Einsatz für Echtzeit-anwendungen oder kleinere Systeme limitiert ist. Der Nutzen ist trotz der Hürden sehr direkt. Beispielsweise lässt sich im Katastrophenmanagement schnell einen Überblick über die Lage verschaffen mittels Dachmaterialklassifikation, um zu erkennen, welche Dächer kaputt sind. Oder man könnte beschädigte Asbestzementdächer damit finden, weil die freigesetzten Fasern ein Gesundheitsrisiko darstellen. Auch kann die Dachmaterialklassifikation bei der Stadtplanung helfen, um zu analysieren, welche Dächer sich im Sommer am meisten aufheizen, um städtische Wärmeinseln zu erkennen. Es lässt sich ebenfalls das Potenzial von Solaranlagen auf tausenden Dächern abschätzen. Die resultierenden

2 Theoretische Grundlagen

Daten lassen sich in digitale Geo-Informationssysteme speisen, um Entscheidungen für ganze Städte zu treffen. [14, 82]

3 Datensätze

In diesem Kapitel werden drei unterschiedliche Datensätze vorgestellt, die exemplarisch die Bandbreite von Klassifikationsaufgaben im maschinellen Lernen demonstrieren: von hochauflösenden Fernerkundungsdaten über den klassischen MNIST-Benchmark bis hin zu dessen modernem Nachfolger Fashion-MNIST.

3.1 Dachmaterialdatensatz

Als Beispiel für einen spezialisierten Anwendungsdatensatz dient eine Sammlung von circa 8.200 annotierten Dachflächen aus dem Saarland, die mittels Fernerkundung erfasst und nach ihrem Material klassifiziert wurden.



Abbildung 3.1: Die Luftaufnahme zeigt das Industriegebiet der Fordwerke mit zentralen Gleisanlagen für die Logistik, Produktionshallen auf der linken und großen Mitarbeiterparkplätzen auf der rechten Seite. Das Werksgelände ist von Grünflächen umgeben und zeigt die typische Infrastruktur eines Automobilproduktionsstandorts.

3.1.1 Aufbau

Die Datenstruktur umfasst Tensoren und RGB-Daten. Dieser Datensatz umfasst ca. 8.200 Dächer. Diese geografischen Objekte sind in 11 verschiedene Materialklassen eingeteilt. Die Auflösung beträgt 0,1 Meter pro Pixel. Das bedeutet, ein einzelner Bildpunkt entspricht einer Fläche von nur 10 mal 10 Zentimetern. Für die Analyse, gerade im Bereich Machine Learning, werden Tensoren verwendet. Ein Tensor ist ein Datencontainer für das Bild. Also eine mehrdimensionale Matrix oder ein Array. Die typische Form für ein Bild wäre HW^3 . H steht für die Höhe des Bildausschnitts in Pixeln, W für die Breite und die drei steht für die Farbkanäle rot, grün und blau. Die Basis stellen diese drei RGB-Kanäle, aber Fernerkundungssensoren können mehr sehen als das menschliche Auge, was Daten im Infrarot- und Thermalbereich umfasst. Diese zusätzlichen Informationen lassen sich als weitere Kanäle, insgesamt bis zu 12, hinzufügen. Dann hätte der Tensor nicht mehr die Form HW^3 , sondern $HW12$. Bei $HW12$ liegen multispektrale Daten vor. Die Dächer in diesem Datensatz wurden annotiert. Eine Person hat für jedes der 8.200 Dächer in den Bildern die Umrisse mit Polygonen nachgezeichnet. Die Geometrie und Form des Polygons wird im WKT-Format (Well Known Text) gespeichert. Das ist eine Textbeschreibung der Koordinaten der Eckpunkte, wodurch das System weiß, welcher Bildbereich zu welchem Dach gehört. Zusätzlich gibt es weitere Metadaten wie die Fläche des Dachs in Quadratmetern. Eine große Fläche ist wahrscheinlich eher eine Industriehalle, eine kleine vermutlich eher eine Garage. [99]

3.1 Dachmaterialdatensatz

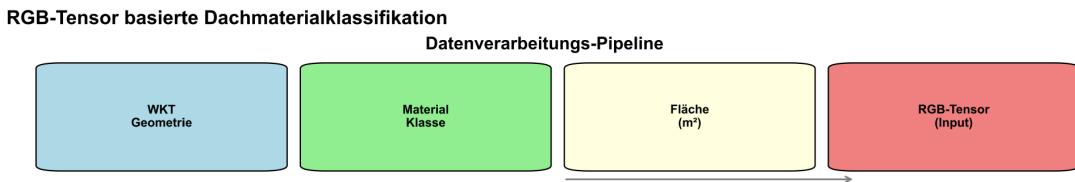


Abbildung 3.2: Datenverarbeitungs-Pipeline für die RGB-Tensor basierte Dachmaterialklassifikation mit vier Hauptkomponenten: WKT-Geometrie, Materialklasse, Fläche und RGB-Tensor Input.

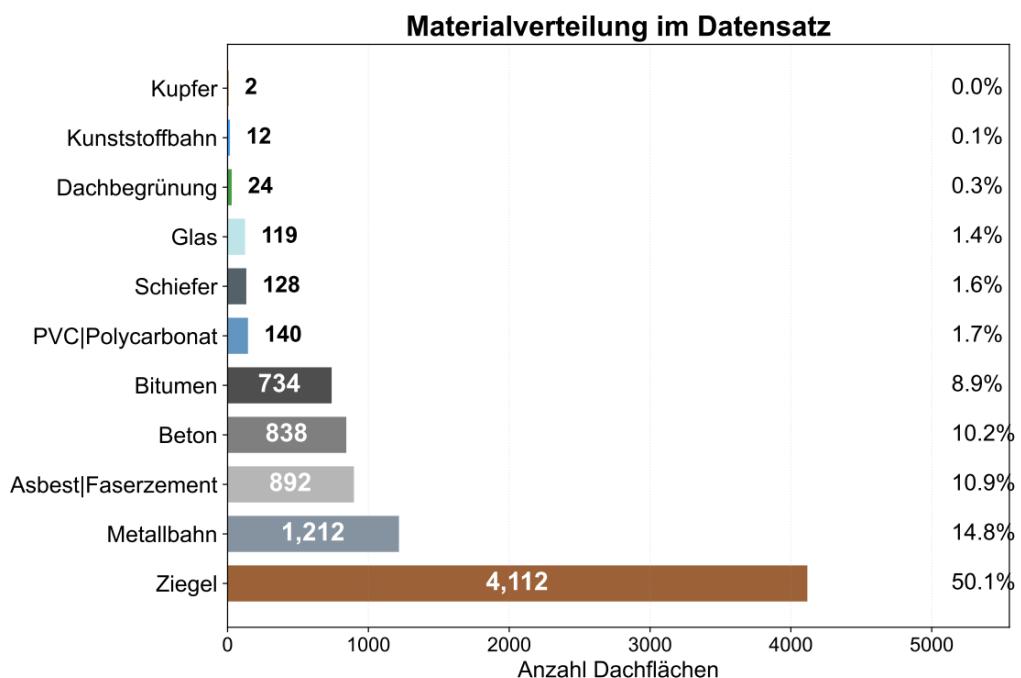


Abbildung 3.3: Verteilung der Dachmaterialien bei 8.213 Dachflächen. Ziegel dominiert mit 50,1%, gefolgt von Metallbahn (14,8%), Asbest/Faserzement (10,9%) und Beton (10,2%).

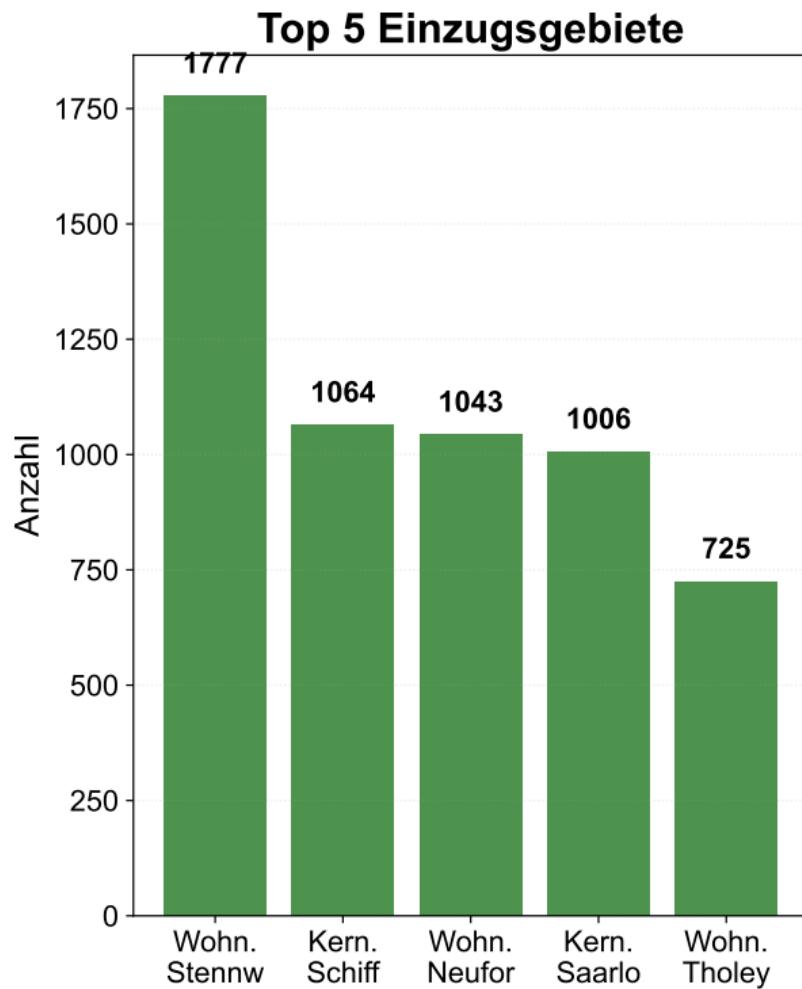


Abbildung 3.4: Die fünf größten Einzugsgebiete: Wohngebiet Steinwald (1.777), Kerngebiet Schiffweiler (1.064), Wohngebiet Neuforstheck (1.043), Kerngebiet Saarlouis (1.006) und Wohngebiet Tholey (725).

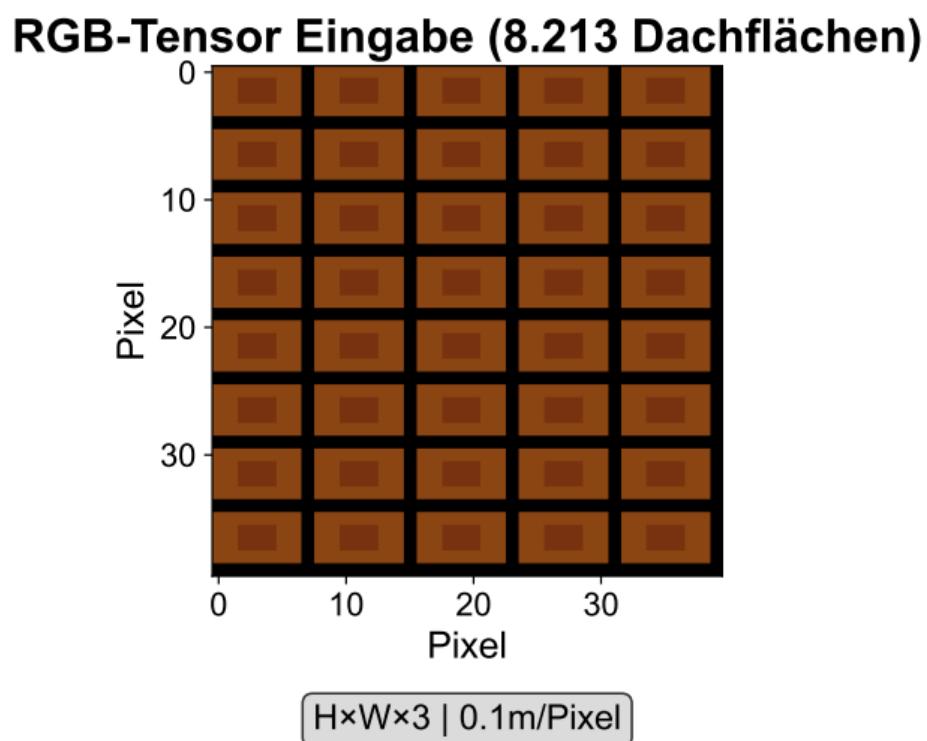


Abbildung 3.5: RGB-Tensor Visualisierung der 8.213 Dachflächen mit 0,1m/Pixel Auflösung. Braune Färbung zeigt die dominierenden Ziegeldächer.

Klassifikations-Info

Area Types:

Type 0: 477
Type 1: 4,112
Type 2: 2,537
Type 3: 1,087

Abbildung 3.6: Flächentyp-Klassifikation: Type 0 (477), Type 1 (4.112), Type 2 (2.537) und Type 3 (1.087) Flächen.

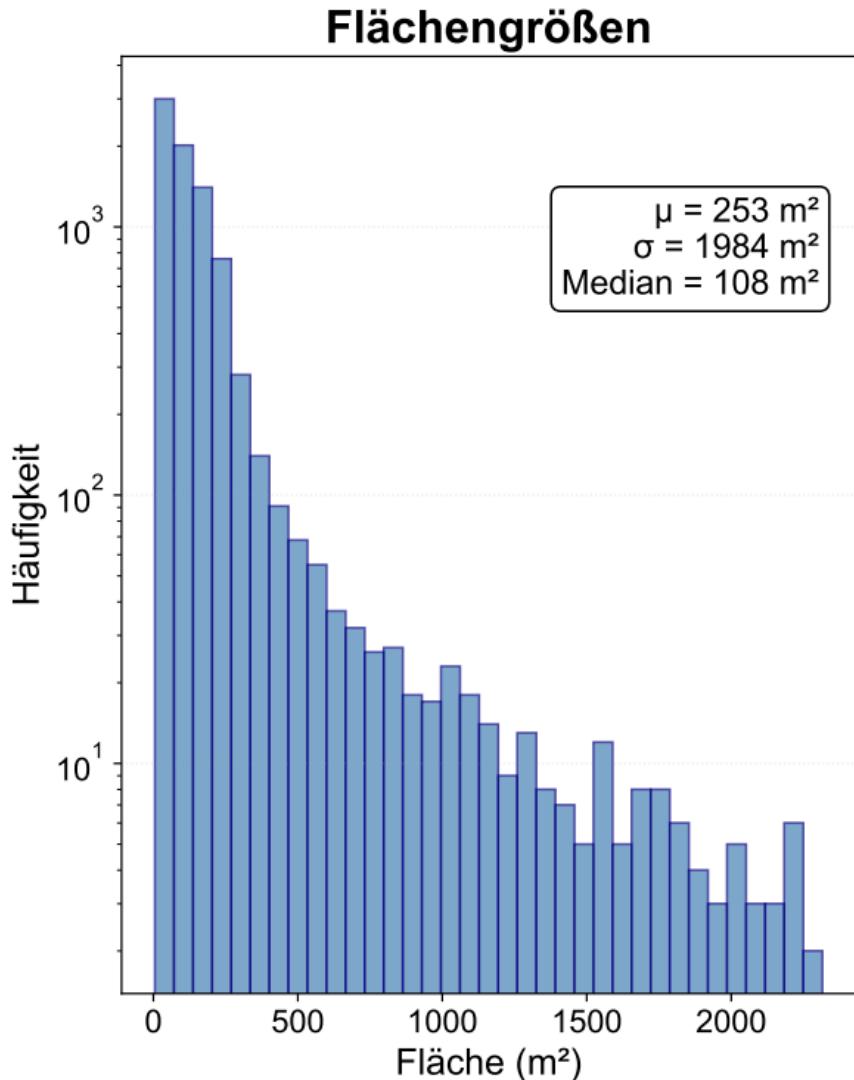


Abbildung 3.7: Histogramm der Dachflächengrößen mit logarithmischer y-Achse. Mittelwert: 253 m², Standardabweichung: 1.984 m², Median: 108 m².



Abbildung 3.8: Kompletter ML-Workflow von der Eingabe über Annotation und Feature-Extraktion bis zur Validierung mit Active Learning und 5 ML-Modellen.

3 Datensätze

3.1.2 Der Zweck

Es handelt sich hierbei um reale Daten aus der Fernerkundung. Der Vorteil durch multispektrale Daten ist, dass Materialien, die für das menschliche Auge ähnlich aussehen wie ein graues Metalldach und eine graue Kunststoffbahn, unter Nahinfrarot beispielsweise ganz unterschiedlich reflektieren und somit besser unterscheidbar sind. Die spektrale Signatur ist daher gut unterscheidbar. Das gibt für das Klassifikationsmodell einen zusätzlichen Hinweis, den es nur mit den drei Farben Rot, Grün und Blau nicht hätte. Mehrere Kanäle können helfen, ähnliche Dinge besser zu unterscheiden. [95]

3.1.3 Limitationen

Es existiert ein Klassenungleichgewicht innerhalb des Datensatzes. Die Klasse Ziegel dominiert den Datensatz, weil über die Hälfte der 8200 Dächer Ziegeldächer sind. Bei anderen Klassen wie Kunststoffbahnen liegen nur zwei Beispiele vor. Das Klassifikationsmodell lernt hier perfekt, Ziegeldächer zu erkennen. Das Modell hat keine Chance, Kunststoffbahnen oder Dachbegrünung (24 Instanzen) zu erkennen, weil die Trainingsdaten nicht ausreichen. Modell wird bei den unterrepräsentierten Klassen Ziegel vorschlagen, weil die statistische Wahrscheinlichkeit hierfür am höchsten ist. Die hohe Auflösung ist im Datensatz vorhanden, aber das Modell lernt nicht, auf die ausreichenden Details zu achten, weil Minderheitsklassen zu stark unterrepräsentiert sind. Durch Featureengineering kann man versuchen, noch bessere Merkmale aus den Daten zu ziehen, um die Modellleistung zu erhöhen. Es gibt auch Techniken, um den Datensatz künstlich auszubalancieren, indem die seltenen Beispiele öfter ins Training gegeben werden. Diese Technik wird Oversampling genannt. Oder es werden die häufigen Beispiele reduziert, was Undersampling genannt wird. Es macht jedoch Sinn, dass Klassen, die mit lediglich 2 oder 3 Instanzen repräsentiert sind, vollständig rausgefiltert werden, weil sie aufgrund der Unterrepräsentativität vom Modell kaum lernbar sind. Convolutional Neural Networks sind oft sehr gut darin, räumliche Kanten, Texturen und Formen in Bildern zu erkennen, was hier ausgleichend wirken könnte. Im Saarland gibt es nun mal größtenteils Ziegeldächer. [12]

3.2 MNIST

MNIST ist eine Abkürzung für „Mixed National Institute of Standards and Technology“. Der Datensatz wurde 1998 von Yann LeCun und seinem Team veröffentlicht. Der Datensatz ist eine Sammlung von handgeschriebenen Ziffern. [58]

3.2.1 Aufbau

Der Datensatz enthält 70.000 kleine Graustufenbilder, mit denen Algorithmen im Bereich des maschinellen Lernens getestet werden. Der Datensatz lässt sich aufteilen in 60.000 Trainingsbilder und 10.000 Bilder zum Testen der trainierten Klassifikationsmodelle. Die Bilder sind lediglich 28 mal 28 Pixel groß. Jeder Pixel hat einen Wert zwischen 0 (also ganz schwarz) und 255 (ganz weiß), was pro Bild 784 Werte darstellt. Das ist der digitale Fingerabdruck jeder Ziffer. Die Rohbilder wurden auf eine einheitliche Größe gebracht und innerhalb des 28×28 -Rasters zentriert, basierend auf dem visuellen Mittelpunkt der Ziffer. Obwohl alles genormt ist, ist die Vielfalt, wie eine Ziffer unterschiedlich dargestellt wird, dennoch hoch. In der nachfolgenden Grafik werden viele Wege dargestellt, die Ziffer 8 aufzuschreiben. Die Acht ist mit fast 10 % der Bilder relativ häufig. In der nachfolgenden Grafik wird zudem t-SNE dargestellt. Das steht für „t-distributed Stochastic Neighbor Embedding“. Diese Methode nimmt hochdimensionale Daten, in diesem Fall die 784

3.2 MNIST

Pixelwerte pro Bild, und stellt diese Werte auf einer 2D-Karte dar, sodass Ähnlichkeiten sichtbar werden. Punkte, die in den 784 Pixelwerten ähnlich waren, sollen auch auf dieser 2D-Karte nahe beieinander liegen. Jeder Punkt stellt ein Bild dar und die Farbe sagt aus, um welche Ziffer es sich handelt. [2]

3 Datensätze

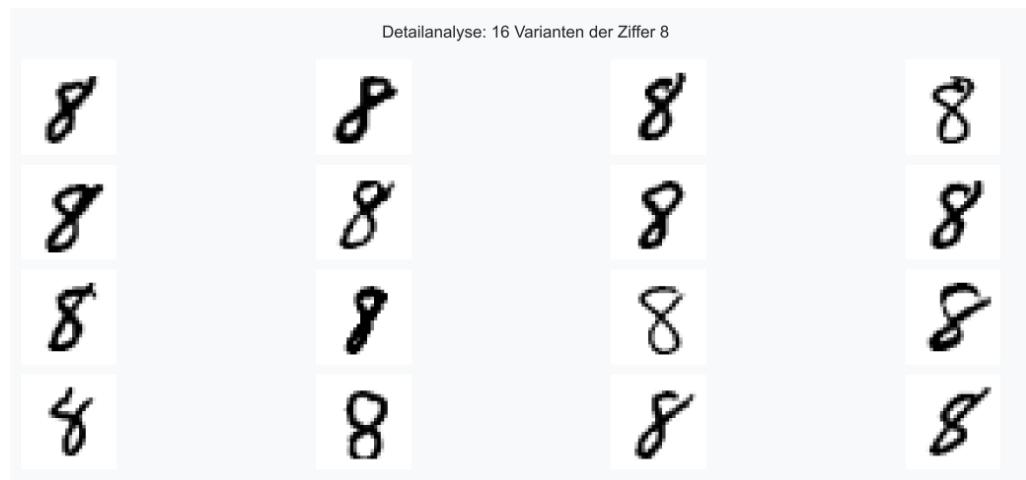


Abbildung 3.9: 16 verschiedene Varianten der Ziffer 8 aus dem MNIST-Datensatz zeigen die hohe Variabilität in Handschriftstilen.



Abbildung 3.10: MNIST-Ziffern 0-9 mit ihren jeweiligen Durchschnittsbildern als Heatmap-Visualisierung.



Abbildung 3.11: Potenzielle Verwechslungspaare im MNIST-Datensatz: 3 vs 8, 5 vs 6, 4 vs 9, und 1 vs 7 mit Erklärung der Ähnlichkeitsmerkmale.

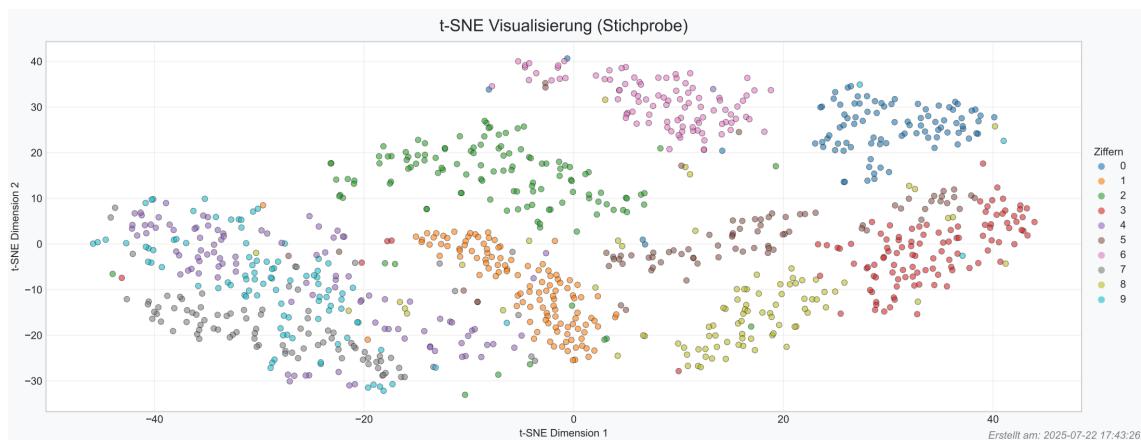


Abbildung 3.12: t-SNE Dimensionsreduktion einer MNIST-Stichprobe zeigt deutliche Cluster-Bildung der 10 Ziffernklassen.

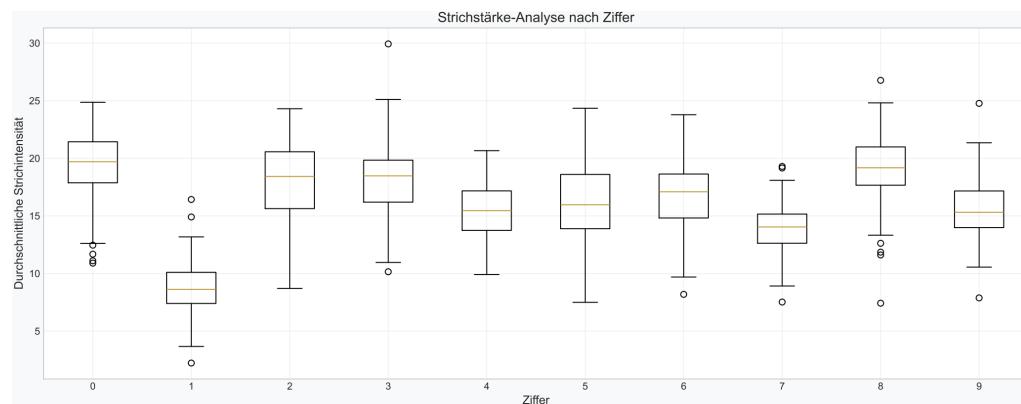


Abbildung 3.13: Boxplot-Analyse der durchschnittlichen Strichintensität nach Ziffer. Ziffer 1 zeigt die geringste, Ziffer 0 die höchste mittlere Strichstärke.

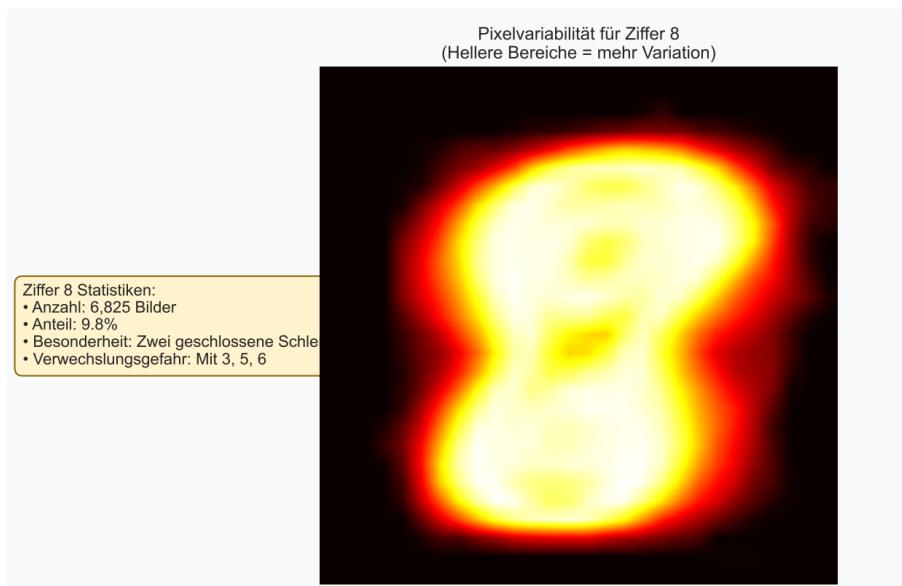


Abbildung 3.14: Heatmap der Pixelvariabilität für Ziffer 8 (6.825 Bilder, 9,8% des Datensatzes). Helle Bereiche zeigen hohe Variation, besonders an den Schleifenrändern.

[MNIST DATENSATZ - ZUSAMMENFASSUNG]

ALLGEMEINE INFORMATIONEN:

- Gesamtanzahl Bilder: 70,000
- Trainings/Test-Split: 60,000 / 10,000 (Standard)
- Bildauflösung: $28 \times 28 = 784$ Pixel
- Farbtiefe: 8-bit Graustufen (0-255)
- Gesamtgröße: ~0.1 Milliarden Pixel

PIXELSTATISTIKEN:

- Durchschn. Pixelintensität: 33.4
- Standardabweichung: 78.7
- Min/Max Werte: 0 / 255

VERWENDUNGSZWECKE:

- Maschinelles Lernen Benchmark
- Computer Vision Training
- Handschrifterkennung (OCR)
- Neural Network Evaluation

INTERESSANTE FAKTEN:

- Erstellt von Yann LeCun (1998)
- Basiert auf NIST-Datenbank
- Zentrierte & normalisierte Bilder
- Meistgenutzer ML-Datensatz

Abbildung 3.15: Übersicht des MNIST-Datensatzes: 70.000 Bilder handgeschriebener Ziffern, 28×28 Pixel, 8-bit Graustufen, Standard-Split 60.000/10.000.

3.2.2 Der Zweck

Der Datensatz hat sich als fundamentaler Benchmark durchgesetzt. Es handelt sich um ein Klassifikationsproblem, weil das Modell feststellen muss, um welche Zahl es sich bei dem jeweiligen Graustufenbild handelt. MNIST hat diese Daten gemischt und neu aufgeteilt, um eine faire Vergleichbarkeit zwischen den Algorithmen zu gewährleisten. Standardisierung ist der Schlüssel zur Vergleichbarkeit der Ergebnisse. [100]

3.2.3 Limitationen

LeCun hat das Problem beim Originaldatensatz vom National Institute of Standards and Technology erkannt, dass die Trainings- und Testdaten von unterschiedlichen Schreibergruppen stammen. Das eine Mal waren es Angestellte vom Census-Büro, das andere Mal Highschool-Schüler, was methodisch gesehen einen Fehler darstellt. Dadurch wurde getestet, wie gut ein Modell auf neue Schreiber generalisiert. Verwechslungen mit anderen Ziffern stellen eine Herausforderung dar. Die Cluster, die daraus entstehen und sich überlappen, zeigen auf, wo die algorithmischen Hürden sind. [97]

3.3 Fashion-MNIST

Fashion-MNIST wurde 2017 von Zalando Research als direkter Drop-in-Ersatz für MNIST konzipiert und ersetzt handgeschriebene Ziffern durch Modeartikel, um die Grenzen moderner Algorithmen besser austesten zu können. [19]

3.3.1 Aufbau

Anstatt Zahlen hat dieser Datensatz Bilder von Modeartikeln. Fashion-MNIST besteht ebenfalls aus 70000 Graustufenbildern. Jedes Bild ist ebenfalls 28 mal 28 Pixel groß. Es gibt hier auch wie bei MNIST 60.000 Bilder zum Training von Modellen und 10.000 Bilder zum Testen. Es gibt 10 Kategorien, die Modelle klassifizieren können. darunter verschiedene Kleidungsstücke, beispielsweise Hosen, Jacken und Schuhe. Es gibt exakt 7.000 Bilder pro Klasse. [91]

3 Datensätze

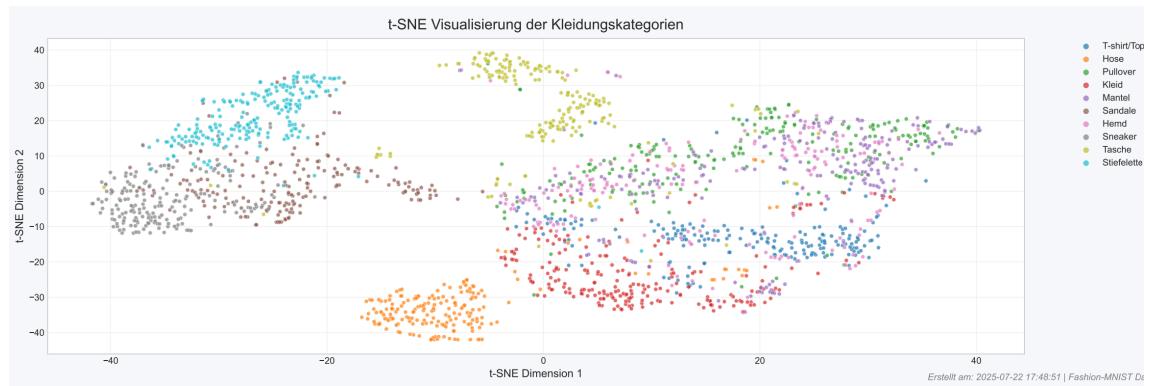


Abbildung 3.16: t-SNE Visualisierung der 10 Kleidungskategorien zeigt deutliche Cluster-Bildung für verschiedene Kleidungstypen.

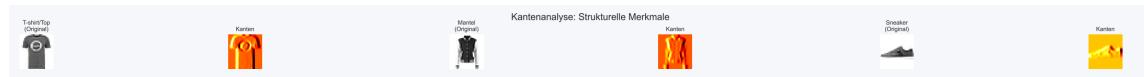


Abbildung 3.17: Strukturelle Merkmale der Kleidungskategorien: T-shirt/Top, Kanten von Hosen, Mantel, Kanten von Kleidern, Sneaker und Kanten von Taschen.



Abbildung 3.18: 16 Beispiele aus der Oberbekleidungskategorie zeigen Variationen in T-shirts, Pullovern, Mänteln und Hemden.



Abbildung 3.19: Potentielle Verwechslungspaare: Hose vs Kleid (Länge), Mantel vs Hemd (Stil), Sneaker vs Stiefelette (Höhe), Pullover vs Hemd (Kragen).

[FASHION-MNIST DATENSATZ - ZUSAMMENFASSUNG]

ALLGEMEINE INFORMATIONEN:

- Gesamtanzahl Bilder: 70,000
- Trainings/Test-Split: 60,000 / 10,000
- Bildauflösung: $28 \times 28 = 784$ Pixel
- Farbtiefe: 8-bit Graustufen (0-255)
- Kategorien: 10 Kleidungstypen

BILDSTATISTIKEN:

- Durchschn. Pixelintensität: 73.0
- Standardabweichung: 90.0
- Hellste Kategorie: Mantel (98.5)
- Dunkelste Kategorie: Sandale (34.9)

DATASET EIGENSCHAFTEN:

- Erstellt von: Zalando Research (2017)
- Zweck: MNIST-Alternative für CV
- Schwierigkeit: Höher als MNIST
- Realistische Produktbilder

ANWENDUNGEN:

- E-Commerce Klassifikation
- Fashion-Tech Entwicklung
- Computer Vision Benchmark
- Deep Learning Training

Abbildung 3.20: Fashion-MNIST Datensatz: 70.000 Bilder, 10 Kleidungskategorien, 28×28 Pixel, erstellt von Zalando Research (2017).



Abbildung 3.21: Die 10 Kleidungskategorien mit ihren Mittelbildern: T-shirt/Top, Hose, Pullover, Kleid, Mantel, Sandale, Hemd, Sneaker, Tasche, Stiefelette.

3 Datensätze

3.3.2 Der Zweck

Der Ansporn war, einen Nachfolger zum sehr bekannten MNIST-Zifferndatensatz zu schaffen. Für viele moderne Algorithmen ist MNIST zu leicht geworden. Das verhindert, dass ein Algorithmus nur lernt, T-Shirts zu erkennen, weil diese zufällig viel häufiger vorkommen würden. Das sorgt für Fairness. Der Kernpunkt zur Verwendung dieses Datensatzes ist, dass MNIST zu einfach ist. Gute Algorithmen haben auf MNIST eine Genauigkeit von 99,7 % erreicht. Das bedeutet, dass auf MNIST keine spürbare Verbesserung neuerer Algorithmen messbar ist, weil es kaum Luft nach oben gibt. Fashion-MNIST ist ein schwierigeres Klassifikationsproblem. Es gibt viele Ähnlichkeiten zwischen den einzelnen Datenpunkten, was die Unterscheidung schwierig macht. Beispielsweise sehen manche T-Shirts aus wie ein Hemd in dem Datensatz. Dieser Datensatz wird in der Forschung als Standard-Benchmark angewendet, um neue Algorithmen fair zu vergleichen, beispielsweise im Bereich Computer Vision mit Convolutional Neural Networks. Der Datensatz wird auch in der Lehre eingesetzt, weil er sehr anschaulich ist. Ein weiterer Anwendungsfall findet sich in der Industrie als Startpunkt für Transferlearning in komplexeren Aufgaben. E-Commerce-Unternehmen verwenden diesen Ansatz für viele Produktempfehlungssysteme, automatische Produktkategorisierung und visuelle Suchfunktionen. Als 2017 der Datensatz rauskam, arbeitete man überwiegend mit Support Vector Machines, die eine Genauigkeit von ca. 87 % erlangten. Mit Deep Learning, speziell Convolutional Neural Networks, konnten ca. 99 % erreicht werden. Durch die kontinuierliche Weiterentwicklung lässt sich Fortschritt messen. [27]

3.3.3 Limitationen

Die Schwächen des Datensatzes umfassen jedoch, dass die Unterbringung feiner Details auf einer 28-mal-28-Pixel-Auflösung nicht möglich ist. Es gibt zudem keine Farbinformationen, weil es nur Graustufenbilder sind. Es könnte zudem Duplikate zwischen Trainings- und Testsets geben, was schlecht für die Bewertung wäre. Für heutige Deep-Learning-Ansprüche gilt der Datensatz eher als klein. Er enthält zudem nur westliche Modeartikel, da er von Zalando kommt, und die Bilder stammen ausschließlich von Zalando. [29]

3.4 Vergleich und Entwicklung der Datensätze

Die historische Entwicklung zeigt, wie sich die Genauigkeiten bei beiden Datensätzen über die Jahre verbessert haben:

3.4 Vergleich und Entwicklung der Datensätze

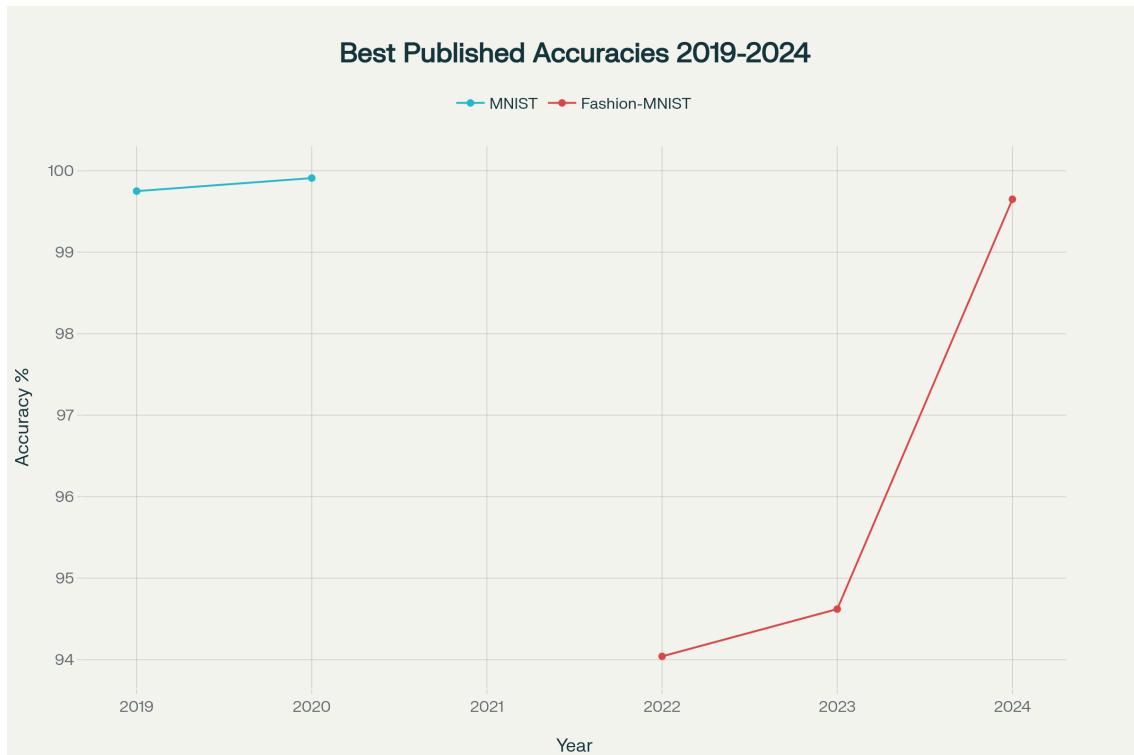


Abbildung 3.22: Historische Entwicklung der besten publizierten Genauigkeiten für MNIST und Fashion-MNIST von 2019 bis 2024. Die Grafik basiert auf folgenden Quellen: MNIST erreichte 2021 eine State-of-the-Art Genauigkeit von 99,67% [81] und stagniert seitdem bei etwa 99,9%. Fashion-MNIST zeigte 2022 eine Genauigkeit von 94,9% [40] und erreichte 2024 einen deutlichen Sprung auf über 99,6% [17]. Diese Grafik wurde aus diesen zitierten Quellen mit Perplexity generiert.

4 Technische Grundlagen

Dieses Kapitel beschreibt die technischen Werkzeuge und Bibliotheken, die für die Implementierung der Active-Learning-Experimente verwendet wurden.

4.1 Überblick über die Implementierung

Die Experimente wurden in drei Jupyter-Notebooks implementiert, die mit Unterstützung von Claude Opus 4 (Anthropic) entwickelt wurden. Die Notebooks evaluieren systematisch verschiedene Kombinationen von Klassifikatoren und Query-Strategien auf drei Datensätzen: MNIST, Fashion-MNIST und einem unbalancierten Dachmaterialdatensatz.

Der Quellcode ist verfügbar unter: <https://github.com/jan1a234/bachelorarbeit.git>

4.2 Datenverarbeitung und -analyse

In diesem Abschnitt werden die für die Evaluation verwendeten, fachspezifischen Python-Bibliotheken für maschinelles Lernen erörtert, wobei grundlegende Python-Kenntnisse vorausgesetzt werden. Der Fokus liegt zunächst auf den Werkzeugen zur Datenverarbeitung und -analyse, da die Aufbereitung der Daten eine elementare Voraussetzung für das Training von Klassifikationsmodellen darstellt.

4.2.1 NumPy

Bei der Entdeckung von Gravitationswellen oder bei dem ersten Bild einer astronomischen Singularität hat Numpy eine entscheidende Rolle gespielt. [21]¹ Numpy ist daher sehr leistungsfähig bei wissenschaftlichen Durchbrüchen. Numpy bietet die Möglichkeit, effizient höhere Mathematik auf riesige Datensätze anzuwenden. Anstatt mit einzelnen Zahlen zu hantieren, arbeitet Numpy mit multidimensionalen Arrays, vergleichbar mit einem Würfel aus Zahlen, was die Berechnungen auf diesen Datensätzen erheblich beschleunigt. [28]

Listing 4.1: Praktisches Beispiel für die im Text erwähnte effiziente Array-Verarbeitung mit NumPy:
Transformation von Trainingsdaten in float32-Arrays für beschleunigte Berechnungen -
Aus: Dachmaterialien_F1_Score.ipynb

```
# X_train_processed ist ein Numpy-Array
X_train_processed = preprocessor.fit_transform(X_train).
                     astype(np.float32)
```

Diese multidimensionalen Arrays erlauben schnellere Operationen als normale Python-Listen, weil sie im Speicher viel effizienter organisiert sind und viele Operationen im schnellen C-Code implementiert sind. Pures Python wäre zu langsam. Diese

¹Für die offizielle Dokumentation siehe NumPy Developers (2024): <https://numpy.org/doc/stable/>. Die hier verwendeten Implementierungen basieren auf NumPy Version 1.x, dem fundamentalen Paket für wissenschaftliches Rechnen in Python.

4 Technische Grundlagen

Arrays sind auch Grundlage für weitere Python-Bibliotheken wie Pandas zur Datenanalyse, Matplotlib zum Visualisieren und Scikit-learn fürs maschinelle Lernen. All diese Bibliotheken bauen auf NumPy auf. [28]

Listing 4.2: Demonstration wie andere Bibliotheken (hier Scikit-learn) auf NumPy-Arrays als Grundlage aufbauen, wie im Text beschrieben - Aus: Dachmaterialien_F1_Score.ipynb

```
# Scikit-learn nutzt NumPy-Arrays für die Metrik-Berechnung
final_f1 = f1_score(y_test, y_pred, average='macro')
```

NumPy bietet fortschrittliche Techniken wie die Vektorisierung. Das bedeutet, dass sich Rechenoperationen wie beispielsweise „-4“ auf alle Zahlen in meinem Datengitter anwenden lassen, ohne selbst eine Schleife programmieren zu müssen. [28]

Listing 4.3: Konkrete Umsetzung der im Text erläuterten Vektorisierung: Entropie-Berechnung ohne explizite for-Schleife durch NumPy-Operationen - Aus: Alle Active-Learning Notebooks

```
# Vektorisierte Berechnung der Entropie ohne for-Schleife
entropies = -np.sum(probs * np.log(probs), axis=1)
```

Außerdem gibt es noch Broadcasting. Das heißt, bei Arrays, die nicht dieselbe Form oder Größe haben, versucht NumPy, diese auf eine intelligente Art anzugeleichen, indem es die kleinere Form auf die größere anpasst. Broadcasting bedeutet in diesem Kontext das die kleinere Form auf die größere Form erweitert wird. Dadurch müssen diese unterschiedlichen Arrays nicht von Hand angepasst werden. Das macht den Code kürzer und schneller lesbar. [28]

Listing 4.4: Praktische Anwendung des im Text beschriebenen Broadcasting-Konzepts: Automatische Anpassung unterschiedlicher Array-Formen bei der Softmax-Berechnung - Aus: Alle Active-Learning Notebooks

```
# Broadcasting: np.max(...).shape: (n,1) wird von decision.shape: (n,m) subtrahiert
exp_decision = np.exp(decision - np.max(decision, axis=1,
                                         keepdims=True))
probs = exp_decision / np.sum(exp_decision, axis=1,
                               keepdims=True)
```

NumPy wird zudem zu einem universellen Übersetzer. Es kann als Brücke dienen für Daten, die auf der CPU liegen, und Daten, die auf leistungsfähigen Grafikkarten verarbeitet werden sollen, beispielsweise für maschinelles Lernen. Oder als Brücke auf verteilten Systemen über mehrere Rechner hinweg. Diese Fähigkeit, verschiedene Systeme zu verbinden, wird als Interoperabilität bezeichnet. [28]

Listing 4.5: Beispiel für die im Text genannte Interoperabilität: NumPy als Brücke zwischen CPU-Daten und GPU-fähigen PyTorch-Tensoren - Aus: MNIST/Fashion-MNIST CNN Notebooks

```
# Interoperabilität: Umwandlung eines NumPy-Arrays in einen PyTorch-Tensor für die GPU
dataset = TensorDataset(
    torch.from_numpy(x_np).float(),
    torch.from_numpy(y_np).long()
)
```

Die Nutzung von ‘torch’ illustriert die Interoperabilität von NumPy, indem es CPU-Daten in GPU-fähige PyTorch-Tensoren für maschinelles Lernen überführt.²

4.2.2 Pandas

Pandas ist wichtig für die Datenanalyse in Python in Bezug auf strukturierte Daten.³ Strukturierte Daten sind Daten, die in Tabellen mit Zeilen und Spalten vorliegen, ähnlich wie in Excel, nur kann Python hier mitbenutzt werden. Damit kann man Data Frames und deren Zeilen und Spalten mit eigenen Labels handhaben. [57]

Listing 4.6: Praktisches Beispiel für das im Text erwähnte Einlesen strukturierter Daten in Pandas DataFrames - Aus: Dachmaterialien_F1_Score.ipynb

```
df = pd.read_csv(filepath)
```

Oder man kann nur eine Series manipulieren, welche eine einzelne Zeile oder Spalte betrifft. [57]

Listing 4.7: Demonstration der im Text beschriebenen Series-Manipulation: Extraktion einzelner Spalten aus einem DataFrame - Aus: Dachmaterialien_F1_Score.ipynb

```
y = df[target_col].copy()
```

Diese Series hat jedoch einen Index, um die einzelnen Datenpunkte zu verbinden. Daten säubern, filtern, sortieren lassen sich mit Pandas bewerkstelligen. Beispielsweise lassen sich Daten bereinigen und Duplikate entfernen. [57]

Listing 4.8: Umsetzung der im Text genannten Datenbereinigung: Filterung des DataFrames nach gültigen Klassen - Aus: Dachmaterialien_F1_Score.ipynb

```
df = df[df[target_col].isin(valid_classes)].copy()
```

Oder die Behandlung fehlender Werte in dem unausgeglichenen Dachmaterialiendatensatz lässt sich bewerkstelligen. [57]

Listing 4.9: Konkrete Anwendung der im Text erwähnten Behandlung fehlender Werte im Dachmaterialdatensatz - Aus: Dachmaterialien_F1_Score.ipynb

```
('imputer', SimpleImputer(strategy='median'))
```

Mit groupby und apply lassen sich komplexe Funktionen auf Daten anwenden. [57]

Listing 4.10: Praktisches Beispiel für die im Text beschriebene groupby-Funktionalität zur komplexen Datenaggregation - Aus: Alle Notebooks mit Evaluation

```
summary = results_df.groupby(['classifier', 'strategy', 'budget_pct'])['f1_score'].agg(['mean', 'std'])
```

Pandas arbeitet intern sehr schnell, was auf Vektorisierung beruht. Operationen auf ganze Datenblöcke werden auf einmal gemacht und nicht in einer Schleife. Zugrunde liegen C-Bibliotheken. Im Rahmen der Evaluation wird diese Bibliothek zur Datenaufbereitung für das maschinelles Lernen verwendet. Die Bibliothek findet jedoch bei jeder Art von wissenschaftlicher Arbeit zur Datenmanipulation Verwendung. [57]

²<https://numpy.org>

³Für die offizielle Dokumentation siehe The pandas development team (2024): <https://pandas.pydata.org/docs/>. Die hier verwendeten Funktionalitäten basieren auf pandas Version 2.x.

4.3 Maschinelles Lernen

Im Vordergrund der Evaluation stehen Datenvorbereitung und faire Bewertung. Man kann nicht einfach die Rohdaten nehmen. Der unbalancierte Dachmaterialdatensatz enthält numerische Features wie die Hausgröße in Quadratmetern und daneben Materialbeschreibung wie Ziegel oder Beton. Der Algorithmus kann mit diesen unterschiedlichen Formaten wenig anfangen. Es besteht die Gefahr, dass große Zahlenwerte kleine Zahlenwerte, die dennoch wichtig sind, überlagern. Deshalb müssen die Daten vergleichbar sein. Realisiert wird das beispielsweise mit dem `StandardScaler`, der alles auf einen gemeinsamen Maßstab bringt. Das bewirkt, dass keine Zahl nur wegen ihrer Größe dominiert. Der korrekte Begriff lautet `StandardScaler`, welcher technisch eine z-Transformation durchführt, bei der jedes Feature auf einen Mittelwert von 0 und eine Standardabweichung von 1 normalisiert wird ($z = \frac{x-\mu}{\sigma}$). Diese Standardisierung stellt sicher, dass alle numerischen Features denselben Einfluss auf das Modell haben, unabhängig von ihrer ursprünglichen Einheit oder Größenordnung. [11, 67]

Listing 4.11: Implementierung der im Text erläuterten Datenvorbereitung: Pipeline mit `SimpleImputer` und `StandardScaler` zur Vermeidung dominierender Zahlenwerte - Aus: Dachmaterialien_F1_Score.ipynb

```
numeric_transformer = Pipeline(steps=[  
    ('imputer', SimpleImputer(strategy='median')),  
    ('scaler', StandardScaler())  
])
```

Für kategoriale Daten wie Ziegel oder Beton gibt es verschiedene Ansätze. Der `LabelEncoder` macht aus Text einfach Zahlen. Dem String „ja“ wird einfach 1 zugeordnet, dem String „nein“ beispielsweise 2. Bei mehreren kategorialen Merkmalen wie Ziegel, Beton, Kupfer etc. ist der `One-Hot-Encoder` besser, weil dieser für jedes Merkmal eine eigene „Ja“-„Nein“-Spalte erstellt, wodurch verhindert wird, dass das Modell eine künstliche Reihenfolge annimmt. Es könnte sein, dass das Modell denkt, Beton sei mehr wert als Ziegel, nur weil diesem String eine höhere Zahl zugeordnet wurde. [11, 67]

Listing 4.12: Umsetzung der im Text beschriebenen One-Hot-Encoding-Strategie zur Vermeidung künstlicher Ordnung bei kategorialen Daten - Aus: Dachmaterialien_F1_Score.ipynb

```
categorical_transformer = Pipeline(steps=[  
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),  
    ('onehot', OneHotEncoder(drop='first', handle_unknown='ignore'))  
])
```

Durch diese Vorgehensweise werden die Daten so aufbereitet, dass die Daten optimal genutzt werden, ohne falsche Schlüsse zu ziehen. Diese Vorgehensweise kann bei sehr vielen zu verarbeitenden Merkmalen dennoch unübersichtlich werden, weshalb `ColumnTransformer` und `Pipeline` zum Einsatz kommen. Der `ColumnTransformer` ist mit einem `Regisseur` vergleichbar, der festlegt, welcher Schritt auf welchen Spalten der Daten ausgeführt wird. [11, 67]

Listing 4.13: Praktische Anwendung des im Text als `Regisseur` beschriebenen `ColumnTransformers` zur koordinierten Datenverarbeitung - Aus: Dachmaterialien_F1_Score.ipynb

```
preprocessor = ColumnTransformer(  
    transformers=[
```

```
('num', numeric_transformer, ['area']),
('cat', categorical_transformer, ['area_type', 'Shape', 'ezg'])
])
```

Die Pipeline packt alles zusammen, was aus One-Hot-Encoder für die Textdaten und dem Standard-Encoder für die Zahlenwerte resultiert. Diese Aufteilung macht den Prozess wiederholbar und verhindert Dataleakage. Dataleakage passiert, wenn Infos aus den Testdaten, die zum Überprüfen da sind, für das Training genommen werden, was das Ergebnis verfälscht. Pipelines sorgen dafür, dass das nicht passiert, durch saubere Abläufe, wodurch bereits Ordnung und Fairness bei der Vorbereitung gewährleistet werden. [11, 67]

4.3.1 Modelltraining mit Scikit-learn

Im Anschluss an die Datenvorbereitung wird geprüft, um die Leistungsfähigkeit des Modells zu bewerten, durch Train-Test-Split. Man teilt hier die Daten zum Lernen, also fürs Training, und einen unabhängigen Teil zum Testen. Es gibt StratifiedShuffleSplit, was besonders wichtig bei unausgeglichenen Datensätzen ist,⁴ was bedeutet, dass einige Klassen viel seltener vorkommen als andere. Beispielsweise sind die interessanten Fälle wie bei Betrugserkennung oder seltenen Krankheiten oft sehr selten. StratifiedShuffleSplit sorgt dafür, dass das Verhältnis der Klassen, also wenige Betrugsfälle und viele Normale im Trainings- und Testset, gleich bleibt wie im Original. Sonst könnte es passieren, dass im Testset zufällig keine seltenen Fälle landen, wodurch die Bewertung sinnlos wäre. [11, 67]

Listing 4.14: Konkrete Implementierung des im Text erläuterten StratifiedShuffleSplit zur Erhaltung der Klassenverteilung bei unbalancierten Datensätzen - Aus: Dachmaterialien_F1_Score.ipynb

```
# Robuster Train/Test Split
# Verwende StratifiedShuffleSplit für bessere Kontrolle
splitter = StratifiedShuffleSplit(n_splits=1, test_size
                                  =0.2, random_state=SEED)
train_idx, test_idx = next(splitter.split(X, y_encoded))

X_train = X.iloc[train_idx]
y_train = y_encoded[train_idx]
```

Anschließend werden verschiedene Machine-Learning-Modelle evaluiert. Dazu gehören Klassifikatoren aus der Scikit-learn Bibliothek wie Naïve Bayes (GaussianNB), Zufallswald (RandomForestClassifier), Logistische Regression (LogisticRegression) und Support Vector Machines (SVC). Zusätzlich wird ein benutzerdefiniertes Neuronales Netz berücksichtigt. Diese breite Auswahl an Modellen ist notwendig, da es kein einzelnes perfektes Klassifikationsmodell für jeden Datensatz gibt und der Vergleich verschiedener Ansätze wichtig ist, um die beste Leistung zu erzielen. [11, 67]

Listing 4.15: Definition der im Text genannten verschiedenen Klassifikatoren für die vergleichende Evaluation - Aus: Alle Active-Learning Notebooks

```
# Klassifikatoren und Strategien definieren
classifiers = ['Neural Network', 'Naive Bayes', 'Random
Forest', 'Logistic Regression', 'SVM']
```

⁴Für die offizielle Dokumentation und API-Referenz siehe Scikit-learn Developers (2024): <https://scikit-learn.org/stable/>. Die hier verwendeten Implementierungen basieren auf Scikit-learn Version 1.x.

4 Technische Grundlagen

```
# ... Schleife zur Durchführung der Experimente ...
for classifier_name in classifiers:
# ...
```

Random-Forest ist ein Ensemble-Modell, das oft sehr robust ist. Eine Support Vector Machine nutzt Kernels, um auch komplexere Muster zu finden, während Gaussian Naïve Bayes ein schneller probabilistischer Ansatz ist. Es werden verschiedene Werkzeuge getestet, um zu prüfen, welches Werkzeug auf den konkreten Datensätzen MNIST, Fashion-MNIST und dem unausgeglichenen Dachmaterialdatensatz am besten funktioniert. [11, 67]

4.3.2 Modellevaluierung mit Scikit-learn

Die verwendeten Evaluierungsmaßzahlen sind dokumentiert unter https://scikit-learn.org/stable/modules/model_evaluation.html. Zum Messen der Modellgenauigkeit kommt der F1-Score zum Einsatz, weil die Accuracy bei unausgeglichenen Datensätzen oft trügerisch ist. Bei ausgeglichenen Datensätzen wie MNIST oder Fashion-MNIST hingegen ist die Accuracy als Maß zum Messen der Modellgenauigkeit ausreichend. Ein Modell, das nur die häufigsten Klassen vorhersagt, also keine Krankheiten oder Betrugsfälle, kann eine sehr hohe Accuracy haben, ist aber in den seltenen Fällen nutzlos. Deshalb wurde speziell der Macro-F1-Score auf dem unausgeglichenen Dachmaterialdatensatz angewandt. Dieser F1-Score ermittelt Präzision und Trefferquote über alle Klassen hinweg, wobei jede Klasse denselben Wert hat. [11, 67]

Listing 4.16: Praktische Anwendung des im Text beschriebenen Macro-F1-Scores für unausgewogene Datensätze - Aus: Dachmaterialien_F1_Score.ipynb

```
# Evaluation mit F1-Score
y_pred = model.predict(X_test)

# Verwende macro average F1-Score für unausgewogene Datensätze
final_f1 = f1_score(y_test, y_pred, average='macro')
final_acc = accuracy_score(y_test, y_pred)
```

Das ergibt ein umfangreicheres Bild, gerade wenn die Erkennung seltener Klassen wichtig ist. Der Classification-Report liefert alle Detailwerte pro Klasse, wodurch die Bewertung deutlich differenzierter wird. [11, 67]

Listing 4.17: Import der im Text erwähnten Metriken zur differenzierten Modellbewertung - Aus: Alle Notebooks

```
# Import der notwendigen Metriken aus Scikit-learn
from sklearn.metrics import accuracy_score, f1_score,
classification_report
```

Das Verständnis dieser Schritte im maschinellen Lernen gibt die Möglichkeit, die Ergebnisse der Evaluation kritisch zu hinterfragen. [11, 67]

4.4 Pytorch

PyTorch ist eine sehr flexible Open-Source-Bibliothek in Python.⁵ Dabei ist das „define by run“-Konzept ein Schlüsselement in Pytorch. Define by run bedeutet, der Bauplan entsteht während der Laufzeit, wodurch das Programm viel intuitiver wird, gerade bei komplexeren Netzen. Es nutzt Grafikkartenbeschleunigung, falls vorhanden. [51]

4.4.1 Architektur neuronaler Netze

Der Bau eines neuronalen Netzes in PyTorch beginnt mit dem Import des zentralen Werkzeugkastens `torch.nn`, der wie ein Legokasten alle notwendigen Bausteine (z.B. `nn.Linear`, `nn.Conv2d`) enthält. [51]

Listing 4.18: Import des im Text als "Legokasten" beschriebenen `torch.nn` Moduls mit allen Bausteinen für neuronale Netze - Aus: MNIST/Fashion-MNIST CNN Notebooks

```
import torch.nn as nn
```

Die Bibliothek bietet einen umfangreichen und modularen Baukasten für neuronale Netze. Hier sind alle Bausteine für ein künstliches Gehirn vorhanden. Hier gibt es die grundlegenden Verbindungsstücke wie `nn.Linear` oder `nn.Conv2d`, was gut darin ist, Bilder und andere räumliche Daten zu erkennen. `nn.Conv2d` ist ein automatischer Merkmalsdetektor, ohne dass die Bildverarbeitung in neuronalen Netzen nicht funktionieren kann. Diese Schicht lässt einen Kernel über die Pixel gleiten, um Ecken, Kanten und Texturen eines Bildes zu identifizieren. [51]

Listing 4.19: Beispiel für die im Text genannten grundlegenden Verbindungsstücke: Linear Layer mit 256 Neuronen - Aus: Dachmaterialien_F1_Score.ipynb (TabularNN)

```
# Eine lineare Schicht aus dem Modell für tabellarische Daten:  
nn.Linear(input_dim, 256),
```

Dann gibt es die Schalter mit Aktivierungsfunktionen wie `nn.ReLU` oder `nn.Sigmoid`, die entscheiden, ob ein Signal weitergeleitet wird. [51]

Listing 4.20: Konkrete Anwendung der im Text beschriebenen Aktivierungsfunktion als SSchalter"für Signalweiterleitung - Aus: Alle Neural Network Notebooks

```
# Die ReLU-Aktivierungsfunktion im Einsatz:  
nn.ReLU(inplace=True),
```

Zudem gibt es Bausteine, die messen, wie gut das Netz bereits performt, wie Normalisierungsschichten und Verlustfunktionen, beispielsweise `nn.CrossEntropyLoss`, die beschreiben, wie weit die Vorhersagen noch vom Ziel entfernt sind. [51]

Listing 4.21: Implementierung der im Text erwähnten Normalisierungsschicht zur Stabilisierung des Trainings - Aus: Dachmaterialien_F1_Score.ipynb (TabularNN)

```
# Eine Normalisierungsschicht zur Stabilisierung des Trainings:  
nn.BatchNorm1d(256),
```

Listing 4.22: Umsetzung der im Text genannten Verlustfunktion zur Messung der Vorhersageabweichung - Aus: Alle Neural Network Notebooks

⁵Für die offizielle Dokumentation und weitere Tutorials siehe PyTorch Foundation (2024): <https://pytorch.org>.

4 Technische Grundlagen

```
# Definition der Verlustfunktion in der Trainingsmethode:  
loss_fn = nn.CrossEntropyLoss()
```

Die Bibliothek ist modular aufgebaut, weshalb jedes vorstellbare Modell gebaut werden kann. Die Bausteine lassen sich also frei kombinieren, wie beim Prototyping. [51]

Listing 4.23: Demonstration der im Text beschriebenen modularen Bauweise: Freie Kombination verschiedener Bausteine im Sequential-Container - Aus: Dachmaterialien_F1_Score.ipynb (TabularNN)

```
# Die Modularität zeigt sich im nn.Sequential-Container,  
# der verschiedene Bausteine zu einem Feature-Extraktor verbindet:  
self.features = nn.Sequential(  
    # Layer 1  
    nn.Linear(input_dim, 256),  
    nn.BatchNorm1d(256),  
    nn.ReLU(inplace=True),  
    nn.Dropout(0.4),  
  
    # Layer 2  
    nn.Linear(256, 128),  
    nn.BatchNorm1d(128),  
    nn.ReLU(inplace=True),  
    nn.Dropout(0.3),  
  
    # Layer 3  
    nn.Linear(128, 64),  
    nn.BatchNorm1d(64),  
    nn.ReLU(inplace=True),  
    nn.Dropout(0.2),  
)
```

Die Forschenden haben die Möglichkeit, zur Laufzeit des Programms Designentscheidungen zu treffen, wie das Netz aufgebaut sein soll. Bei älteren Frameworks musste der Plan vor der Laufzeit des Programms fertig sein. [51]

4.4.2 Optimierung mit Adam

Damit das neuronale Netz lernt, kommen diverse Algorithmen zum Einsatz. Im verwendeten Programm für die Evaluierung dieser Arbeit wurde der Adam-Optimierer des **torch.optim-Moduls** verwendet. Adam steht für Adaptive Moment Estimation. Er fungiert als eine adaptive Steuerungsstrategie für das neuronale Netz. Wenn der Weg stetig und zuverlässig in die richtige Richtung führt, wird Adam mutiger und gibt dem Netz an, einen größeren Schritt zu machen. Dabei ist sich bildlich vorzustellen, dass das neuronale Netz vom Berg ins Tal der optimalen Lösung runter möchte. Adam passt dabei für jede kleine Stellschraube im Netz die Schrittgröße an. Adam schaut zurück und reflektiert die Steigung im Durchschnitt und wie stark sie geschwankt hat, um anzugeben, ob die Richtung verlässlich ist. Andernfalls gibt Adam an, wenn es hügelig ist, einen kleinen Schritt zu gehen. Adam kommt dadurch schneller ans Ziel als ältere Methoden, die eine feste Schrittgröße verwendeten, weil Adam sich dem Gelände anpasst. [51]

Exemplarisch wird die Initialisierung des Optimierers in der Trainingsfunktion des neuronalen Netzes gezeigt:

Listing 4.24: Praktische Implementierung des im Text erläuterten Adam-Optimierers mit adaptiver Schrittgrößenanpassung - Aus: Dachmaterialien_F1_Score.ipynb (TabularNN)

```

def fit(self, X_np, y_np, epochs=10, lr=1e-3, batch_size
       =256, verbose=False):
    """
    Trainiert das TabularNN mit optimierten Hyperparametern.
    """
    self.train()
    # ... (weitere Initialisierungen)
    # Hier wird der AdamW-Optimierer instanziert
    optimizer = optim.AdamW(self.parameters(), lr=lr,
                           weight_decay=1e-4)
    scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer,
                                                      T_max=epochs)
    loss_fn = nn.CrossEntropyLoss()
    # ... (Trainingsschleife)

```

4.4.3 Datenhandling mit DataLoader

Um das neuronale Netz mit Daten zu füttern, gibt es beispielsweise `torch.utils.data`. [51]

Listing 4.25: Import der im Text beschriebenen Datenhandling-Komponenten `TensorDataset` und `DataLoader` - Aus: Alle PyTorch Notebooks

```
from torch.utils.data import TensorDataset, DataLoader
```

Es ist zu beachten, dass die Datenversorgung bei Deep Learning oft der Flaschenhals ist und die Datenversorgung deshalb einen eigenen Baustein innerhalb dieser Python-Bibliothek darstellt. Verwendet werden dabei das `TensorDataset` und der `DataLoader`. `TensorDataset` ist nur da, um die Eingabedaten mit den zugehörigen Zielwerten ordentlich zu verpacken. [51]

Listing 4.26: Konkrete Umsetzung des im Text erwähnten `TensorDatasets` zum Verpacken von Eingabedaten und Zielwerten - Aus: Alle PyTorch Notebooks

```

# Create dataset
try:
    dataset = TensorDataset(
        torch.from_numpy(X_np).float(),
        torch.from_numpy(y_np).long()
    )
except Exception as e:
    logger.error(f" Fehler beim Erstellen des Datasets: {e}")
    raise

```

Eine zentrale Komponente für den Trainingsprozess ist der `DataLoader`. Er ist für die effiziente Bereitstellung der Daten verantwortlich, indem er den Datensatz in kleine Pakete, sogenannte `Batches`, aufteilt und diese an das neuronale Netz weiterleitet. Der `DataLoader` kann auch die Reihenfolge der Daten mischen, damit das Netz nicht die Reihenfolge auswendiglernt. Der `DataLoader` kann zudem die `Batches`, die übergeben werden müssen, auf mehreren Kernen vorbereiten. Das sorgt dafür, dass die Grafikkarte immer beschäftigt ist, um die Trainingszeit des neuronalen Netzes zu verkürzen. Das effiziente Datenhandling ist daher für die Reduzierung der Laufzeit des Programms am wichtigsten. [51]

4 Technische Grundlagen

Listing 4.27: Praktische Konfiguration des im Text erläuterten DataLoaders mit Batch-Aufteilung und Shuffle-Option zur effizienten GPU-Auslastung - Aus: Alle PyTorch Notebooks

```
# DataLoader mit optimierten Settings
loader = DataLoader(
    dataset,
    batch_size=batch_size,
    shuffle=True,
    num_workers=0, # Immer 0 für Kompatibilität
    pin_memory=(self.device.type == 'cuda'),
    drop_last=False
)
```

4.5 Statistische Auswertung

Mittels Wilcoxon signed rank test wird geprüft, ob die beobachteten Unterschiede zwischen Machine-Learning-Modellen statistisch signifikant sind.⁶ Beispielsweise Modell A und Modell B, die auf denselben Daten laufen. Man kann nicht einfach den Durchschnittsfehler vergleichen. Es braucht den Wilcoxon-Test, weil die Daten nicht normalverteilt vorliegen. Der Wilcoxon-Test umgeht die Gefahr durch Ausreißer, die besteht, wenn nur der Durchschnittsfehler genommen würde. Der Wilcoxon schaut sich nicht die genauen Fehlerwerte an, sondern nur die Rangfolge der Unterschiede. Für jeden Datensplit fragt man: Wer war besser und wie groß war der Unterschied im Vergleich zu den anderen Unterschieden? Mit Python und SciPy lässt sich der Wilcoxon-Test in einer Codezeile durchführen. [93]

Listing 4.28: Import des im Text beschriebenen Wilcoxon-Tests für nicht-normalverteilte Daten - Aus: Alle Notebooks mit statistischer Analyse

```
from scipy.stats import wilcoxon
```

Listing 4.29: Konkrete Anwendung des im Text erläuterten Wilcoxon-Tests zur robusten Modellvergleichung trotz Ausreißern - Aus: Alle Notebooks mit statistischer Analyse

```
statistic, p_value = wilcoxon(
    strategy_data, baseline_data,
    alternative='greater',
    zero_method='zsplit'
)
```

4.6 Datenvisualisierung

Diese Bibliotheken werden zur Datenvisualisierung verwendet. Sie machen Zahlen anschaulich zu Bildern und Grafiken. [98]

⁶Für eine detaillierte Einführung in statistische Tests und den Wilcoxon-Test siehe Küchenhoff (2017): https://www.stablab.stat.uni-muenchen.de/lehre/veranstaltungen/veranstaltungsinhalte/stat2_bwl_vl9_neu.pdf, S. 359.

4.6.1 Matplotlib

Matplotlib⁷ ist ein Werkzeugkasten fürs Zeichnen. Durch diese Bibliothek erhält der Anwender die Kontrolle über jeden Pixel innerhalb der zu erstellenden Visualisierung. `matplotlib.pyplot` bildet hierzu das Fundament. Es ist sehr mächtig, jedoch ist es nötig, jedes Detail von Hand einzustellen. [98]

4.6.2 Seaborn

Obwohl Matplotlib bereits sehr mächtig ist, kann es vorteilhaft sein, durch Seaborn⁸ Kontrolle abzugeben. Seaborn baut auf Matplotlib auf und nutzt Matplotlib im Hintergrund. Seaborn bietet jedoch eine zugänglichere und spezialisiertere Oberfläche im Vergleich zu Matplotlib. Seaborn ist insbesondere für statistische Plots wie in diesem Active-Learning-Experiment der Fall heranzuziehen. [98]

Die Einrichtung von `seaborn` zusammen mit `matplotlib` erfolgt typischerweise am Anfang eines Skripts, wobei oft auch ein bestimmter Stil für die Grafiken festgelegt wird, um die Lesbarkeit und Ästhetik zu verbessern. [98]

Listing 4.30: Setup der im Text genannten Bibliotheken Matplotlib und Seaborn mit Fehlerbehandlung und Stil-Konfiguration - Aus: Alle Notebooks mit Visualisierung

```
import matplotlib
matplotlib.use('Agg') # Für Server ohne GUI
import matplotlib.pyplot as plt
# Seaborn mit Fehlerbehandlung
try:
    import seaborn as sns
    # Prüfe ob der Style verfügbar ist
    try:
        plt.style.use('seaborn-v0_8-whitegrid')
    except:
        try:
            plt.style.use('seaborn-whitegrid')
        except:
            plt.style.use('ggplot')
        except ImportError:
            print("Warnung: Seaborn nicht installiert. Verwende
                  Standard-Matplotlib .")
sns = None
```

Durch Seaborn sind komplexe statistische Plots mit wenigen Codezeilen machbar. Zum Beispiel lassen sich mithilfe von Seaborn Boxplots einfach erstellen, um Verteilungen zu verstehen. Ebenfalls lassen sich hiermit sehr einfach Heatmaps erstellen, um Beziehungen zwischen Variablen mit einem Augenblick zu erkennen. [98]

Der folgende Ausschnitt zeigt, wie eine Heatmap mit `seaborn` erzeugt wird, um die Effektstärke (Cliff's Delta) über verschiedene Budgets und Strategien darzustellen. [98] Die Effektstärke gibt an, wie viel besser eine verwendete Methode ist, die hier verglichen wird. Cliff's Delta ist die hier verwendete Art, die Effektstärke zu messen. [76]

Listing 4.31: Praktisches Beispiel für die im Text beschriebene Heatmap-Erstellung mit Seaborn zur Visualisierung von Cliff's Delta Effektstärken - Aus: Dachmaterialien_F1_Score.ipynb

```
# Heatmap der Effektstärken
```

⁷<https://matplotlib.org>

⁸<https://seaborn.pydata.org>

4 Technische Grundlagen

```
pivot_effect = stat_results.pivot_table(  
    values='cliffs_delta',  
    index=['classifier', 'strategy'],  
    columns='budget_pct',  
    fill_value=0.0  
)  
if sns is not None and not pivot_effect.empty:  
    sns.heatmap(pivot_effect,  
        annot=True,  
        fmt='.3f',  
        cmap='coolwarm',  
        center=0,  
        vmin=-1,  
        vmax=1,  
        cbar_kws={'label': "Cliff's Delta"},  
        ax=ax2)
```

Zudem bringt Seaborn ansprechende Designs und Farbpaletten als Standardkonfiguration bereits mit. Es lassen sich daher schnell Einblicke gewinnen über beispielsweise Normalverteilung und Gruppierung der Daten. Eine Konfusionsmatrix zeigt außerdem, wo ein Machine-Learning-Modell Fehler macht, und lässt sich mit Seaborn leicht erstellen. Auch einfache Balkendiagramme können generiert werden, um wichtige Faktoren hervorzuheben. [98]

Hier wird ein gruppiertes Balkendiagramm erstellt, um die prozentuale Verbesserung des F1-Scores gegenüber dem Zufalls-Sampling zu visualisieren. [98]

Listing 4.32: Umsetzung des im Text erwähnten gruppierten Balkendiagramms zur Darstellung der prozentualen F1-Score-Verbesserung - Aus: Alle Notebooks mit Visualisierung

```
# Gruppiertes Barplot  
x = np.arange(len(BUDGET_PERCENTAGES))  
width = 0.15  
# ... (Schleife über Klassifikatoren und Strategien)  
offset = (i * len(strategies) + j - len(classifiers) * len(  
    strategies)) / 2) * width  
bars = ax.bar(x + offset, values, width,  
label=f'{classifier} - {strategy}',  
color=colors[color_idx])
```

Seaborn unterstützt Barrierefreiheit, indem es Farbpaletten für Menschen mit Sehbehinderung mitliefert. Für Publikationen wie diese lassen sich Grafiken als PDF-Datei exportieren, durch Seaborn. Der Befehl plt.savefig aus matplotlib, auf dem seaborn aufbaut, ermöglicht das Speichern der erstellten Grafiken in verschiedenen Formaten und mit hoher Auflösung. [98]

Listing 4.33: Konkrete Implementierung des im Text genannten PDF-Exports mit plt.savefig für publikationsreife Grafiken - Aus: Dachmaterialien_F1_Score.ipynb

```
# Speichern mit Fehlerbehandlung  
filename = f'plots/dachmaterial_{classifier.lower().replace  
    (" ", "_")}_f1_with_significance.png'  
try:  
    plt.savefig(filename, dpi=300, bbox_inches='tight')  
    logger.info(f"[ok] Visualisierung mit F1-Score und  
        Signifikanz fuer {classifier} erstellt: {filename}")  
except Exception as e:
```

```
logger.error(f" Fehler beim Speichern der Visualisierung
fuer {classifier}: {e} ")
```

4.7 Sicherstellung der Reproduzierbarkeit

Die Sicherstellung der Reproduzierbarkeit in dieser Arbeit ist elementar, damit die Ergebnisse der hier durchgeführten Experimente nachvollzogen werden können. Reproduzierbarkeit ist das Fundament für das Vertrauen in die hier erhaltenen Ergebnisse. Angenommen, ein weiterer Wissenschaftler übernimmt die Daten und Vorgehensweise dieser Arbeit. In diesem Fall sollten exakt dieselben Resultate hervorkommen. Reproduzierbarkeit schafft daher Transparenz und macht die hier durchgeführte Forschung überprüfbar. Wenn diese Arbeit nicht reproduzierbar ist, so ist sie nicht glaubwürdig. Gerade im maschinellen Lernen wäre das sehr kritisch, weil Zufallsprozesse eine übergeordnete Rolle spielen. Beispielsweise die zufällige Aufteilung der Daten in Trainings- und Testsets, die Initialisierung der Gewichte in neuronalen Netzen. Wenn nicht bewusst auf Reproduzierbarkeit geachtet wird, würden die Experimente bei jedem Durchlauf andere, leicht unterschiedliche Ergebnisse liefern. Das bedeutet, die Ergebnisse wären nicht vergleichbar oder wiederholbar. Der Seed ist ein zentrales Werkzeug zur Gewährleistung der Reproduzierbarkeit dieser Arbeit. Der Seed repräsentiert einen festgelegten Startpunkt für den Pseudozufallszahlengenerator eines Computers. Wird demselben Zufallsgenerator exakt derselbe Seed übergeben, so erzeugt er immer dieselbe Abfolge von Zufallszahlen. Diese Zufallszahlen sind jedoch durch den Seed nicht mehr zufällig, sondern deterministisch. In dem genutzten Evaluierungscode, dem Jupyter-Notebook für die Active-Learning-Experimente, wird der globale Seed 42 verwendet. 42 ist hierbei eine Konvention, da sie eine wiedererkennbare Zahl ist. Anschließend wird der Seed bei allen wichtigen Bibliotheken wie PyTorch oder NumPy gesetzt. Die Seeds werden am Anfang des Programms gesetzt, um sicherzustellen, dass Zufallsoperationen bei jedem Run des Programms identisch ablaufen. [13]

Listing 4.34: Praktische Umsetzung der im Text als zentral beschriebenen Seed-Initialisierung mit dem konventionellen Wert 42 für deterministische Zufallszahlen - Aus: Alle Notebooks

```
# -----
# Reproduzierbarkeit
# -----
SEED = 42
np.random.seed(SEED)
torch.manual_seed(SEED)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(SEED)
```

Bei dem Einsatz von Grafikkarten muss beachtet werden, dass PyTorch auf hochoptimierte Routinen zurückgreift. Einige dieser Routinen sind nicht deterministisch, weshalb diese Routinen im Programm per Konfiguration abgeschaltet werden. Dadurch wird Geschwindigkeit zum Gewährleisten der Reproduzierbarkeit geopfert, damit die Ergebnisse belastbar sind. [13]

Listing 4.35: Implementierung der im Text erwähnten Deaktivierung nicht-deterministischer GPU-Routinen zur Gewährleistung der Reproduzierbarkeit - Aus: Alle GPU Notebooks

```
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

4.8 GPU-optimiertes aktives Lernen auf den Datensätzen

Für die Validierung der Klassifikatoren werden die Berechnungen aus Gründen der Praktikabilität ausschließlich auf der GPU ausgeführt, um die Trainingszeit massiv zu verkürzen, da die GPU für parallele Berechnungen spezialisiert ist und die CPU einen Allzweckprozessor darstellt. Es wäre ein Vergleich zwischen Äpfeln und Birnen. Zum Einsatz kommen die Bibliotheken `Rapids cuML`, welche die Standard Klassifikatoren direkt für die NVIDIA-GPU implementieren. Die Implementierung über `Rapids cuML` ist 10 bis 50 Mal schneller als über `Scikit-Learn`, was über die CPU läuft. Der `RMM` (Rapids Memory Manager) verwaltet den zur Verfügung stehenden Grafikspeicher. [16, 74] Zum Beispiel:

Listing 4.36: Konkrete Konfiguration des im Text beschriebenen Rapids Memory Managers mit 5-6.5GB Speicherpool auf der GPU - Aus: MNIST/Fashion-MNIST GPU Notebooks

```
rmm.reinitialize(
    pool_allocator=True,
    initial_pool_size="5GB",
    maximum_pool_size="6.5GB",
    managed_memory=False
)
```

konfiguriert einen Speicherpool auf der GPU. In diesem Fall 5 GB initial und maximal 6,5 GB, da der maximal zur Verfügung stehende VRAM 8 GB beträgt. Der `Rapids-Memory-Manager` passt auf, dass der Speicher nicht fragmentiert. Er enthält auch eine Funktion wie „`clear gpu memory`“, was entscheidend ist, um nach jedem Durchlauf eines Active-Learning-Experiments den Grafikspeicher erneut freizugeben, um Platz für die nachfolgenden Durchläufe zu schaffen. [16, 74]

Listing 4.37: Praktische Implementierung der im Text erwähnten GPU-Speicherfreigabe nach jedem Active-Learning-Durchlauf - Aus: Alle GPU Notebooks

```
def clear_gpu_memory():
    if torch.cuda.is_available():
        torch.cuda.empty_cache()

    if CUBLAS_AVAILABLE:
        mempool = cp.get_default_memory_pool()
        mempool.free_all_blocks()
        gc.collect()
```

Der Code implementiert die bereits in dem Kapitel „Theoretische Grundlagen“ erläuterten Active-Learning-Strategien Margin-Sampling, Entropy-Sampling und Least-Confidence und vergleicht diese mit Random-Sampling als Baseline. Zudem wird die Trainingszeit gemessen und die Labelersparnis gemessen, wenn nur 95 % Genauigkeit erreicht werden sollen, gegenüber Random-Sampling als Baseline. Zudem implementiert der Code den Wilcoxon-signed-rank-Test, um die Aussagekraft der hier durchgeföhrten Experimente zu belegen. [16, 74]

Listing 4.38: Umsetzung der im Text genannten statistischen Analyse mit Wilcoxon-Test zum Strategienvergleich - Aus: Alle Notebooks mit statistischer Auswertung

```
def perform_statistical_analysis(results_df, metric='accuracy'):
    for strategy in strategies:
        if strategy == 'Random Sampling':
            continue
```

```

strategy_data = results_df[...][metric].values
baseline_data = results_df[
    results_df['strategy'] == 'Random Sampling'
][metric].values

statistic, p_value = wilcoxon(
    strategy_data, baseline_data,
    alternative='greater',
    zero_method='zsplit'
)

```

Er vergleicht die Strategien über mehrere Durchläufe hinweg signifikant mit der Random-Baseline. Mit Cliff's Delta wird die Effektstärke berechnet, die angibt, wie viel besser eine Strategie in der Praxis gegenüber der Random Baseline ist, falls sie besser ist. [16, 74]

Listing 4.39: Konkrete Berechnung der im Text beschriebenen Cliff's Delta Effektstärke zur Quantifizierung der Strategieverbesserung - Aus: Alle Notebooks mit Effektstärken-Berechnung

```

def clifffs_delta(x, y):
    nx, ny = len(x), len(y)
    greater = sum(xi > yj for xi in x for yj in y)
    less = sum(xi < yj for xi in x for yj in y)
    d = (greater - less) / (nx * ny)
    return np.clip(d, -1.0, 1.0)

```

Weil mehrere Vergleiche durchgeführt werden, wird zudem die Bonferroni-Korrektur durchgeführt, um sicherzustellen, dass die Ergebnisse verlässlich sind, um die Alpha-Fehler-Kumulierung, wie im Kapitel „Theoretische Grundlagen“ bereits erläutert, zu vermeiden. [16, 74]

Listing 4.40: Implementierung der im Text erläuterten Bonferroni-Korrektur zur Vermeidung der Alpha-Fehler-Kumulierung bei multiplen Tests - Aus: Alle Notebooks mit multiplen Tests

```

n_comparisons = len(stat_df)
stat_df['p_value_corrected'] = np.minimum(
    stat_df['p_value'] * n_comparisons, 1.0
)
stat_df['significant'] = stat_df['p_value_corrected'] <
    SIGNIFICANCE_LEVEL

```

Zudem werden die Daten normalisiert und aufbereitet, damit die Klassifikatoren damit arbeiten können. [16, 74]

Listing 4.41: Praktisches Beispiel für die im Text genannte Datennormalisierung und -aufbereitung für die Klassifikatoren - Aus: MNIST Notebooks

```

def load_mnist_data():
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
    ])

    train_dataset = torchvision.datasets.MNIST(
        root=data_dir, train=True, download=True, transform=
            transform
    )

```

4 Technische Grundlagen

```
)  
  
# Flatten for SVM (2D: batch, features)  
X_train_flat = X_train.view(X_train.size(0), -1).numpy()
```

Nachdem die Experimente durchlaufen sind, zeigen Lernkurven, die mit Matplotlib und Seaborn erstellt wurden, auf, wie die Genauigkeit mit zunehmenden Datenpunkten ansteigt. Heatmaps visualisieren die Ergebnisse der statistischen Tests. Signifikanz und Effektstärke auf einen Blick. [16, 74]

Listing 4.42: Umsetzung der im Text beschriebenen Visualisierung mit Matplotlib/Seaborn inklusive Signifikanz-Markierung der Ergebnisse - Aus: MNIST/Fashion-MNIST SVM Notebooks

```
def plot_gpu_svm_results(all_results, stat_results):  
    # German matplotlib configuration  
    plt.rcParams['font.family'] = 'DejaVu Sans'  
    plt.rcParams['axes.unicode_minus'] = False  
  
    # Significance-based visualization  
    for strategy, color in strategy_colors.items():  
        is_significant = check_significance(strategy, stat_results)  
        label = strategy_labels_de.get(strategy, strategy)  
        if is_significant:  
            label += f" *({effect_size})"
```

4.9 NVIDIA RTX 4060 Architektur und Funktionsweise

Das Betreiben aller durchgeführten Experimente auf der Grafikkarte ist elementar für diese Arbeit. Ohne GPU-optimierte Ausführung wäre die Aufgabenstellung nicht innerhalb der vorgegebenen Bearbeitungszeit von 3 Monaten zu bewältigen. Konkret werden die Experimente auf der Nvidia RTX 4060 GPU ausgeführt. Diese Grafikkarte besitzt 3072 CUDA-Kerne. Die GPU kann man sich vorstellen wie eine große Fabrikhalle, innerhalb derer tausende Mitarbeiter einfache Handgriffe simultan machen. Die CPU schafft diese Parallelverarbeitung in dem Grad, wie es die GPU kann, für den Anwendungsfall des maschinellen Lernens nicht. Für das aktive Lernen muss die Unsicherheit für tausende Datenpunkte gleichzeitig berechnet werden, um die Datenpunkte, bei denen der Klassifikator die größte Unsicherheit hat, auszuwählen. Das macht die GPU parallel. Die CPU kann das nur nacheinander abarbeiten, was zu viel längeren Laufzeiten führen würde. Diese Grafikkarte basiert auf der Ada-Lovelace-Architektur. Sie besitzt 96 Tensorkerne der 4. Generation, die für das maschinelle Lernen spezialisiert sind. Diese Tensorkerne beschleunigen Matrixoperationen, die in neuronalen Netzen ständig anfallen. Diese Tensorkerne unterstützen FP8. Das bedeutet, sie unterstützen niedere Genauigkeit, was schneller ist und weniger Energie verbraucht. Das ist ein Hebel, um Auswahlstrategien im Active Learning schnell genug zu machen. Die Grafikkarte enthält 8 GB VRAM, was den eigenen internen Speicher darstellt. Darauf liegen die Daten, auf die die CUDA-Kerne ständig zugreifen. Beispielsweise die Modellgewichte und die Datenpunkte. Der VRAM ist für den vollständigen MNIST-Datensatz ausreichend. Die Geschwindigkeit der Grafikkarte beträgt 272 Gigabyte pro Sekunde. Die Ada-Lovelace-Architektur beinhaltet auch einen großen L2-Cache, der 24 Megabyte beträgt bei der 4060-Grafikkarte. Der L2-Cache ist ein schneller Puffer zwischen den Kernen und dem VRAM. Der L2-Cache reduziert die Zugriffe auf den langsameren VRAM und hält die Kerne besser beschäftigt, um die Effizienz zu steigern. Die Ada-Lovelace-Architektur setzt auf Parallelität mit für das Machine-Learning

spezialisierten Tensor-Kernen und einem schnellen Speichersystem. Zusammenfassend ermöglicht die Ausführung auf der GPU die Bewältigung der Aufgabenstellung, was auf einer CPU nicht praktikabel wäre. [56, 90]

4.10 CuPy

Wenn in Python bereits NumPy für numerische Berechnungen zum Einsatz kommt, ist CuPy ein Ersatz dafür, was auch „Drop-in replacement“ bezeichnet wird. CuPy läuft jedoch nicht auf dem Hauptprozessor, der CPU, wie NumPy, sondern auf der Grafikkarte, der GPU. Der Vorteil darin ist, dass GPUs tausende Rechenkerne haben, wohingegen CPUs nur eine Handvoll Rechenkerne besitzen, wodurch ein deutlich höherer Grad an Parallelisierung möglich wird, was wiederum zu einer kürzeren Laufzeit der Experimente des maschinellen Lernens führt. Matrixmultiplikationen, worauf die Support-Vector-Machine beruht, können dadurch 10 bis 12 Mal schneller sein. Reduktionsoperationen, bei denen Summen über große Arrays gebildet werden, laufen bis zu 25 Mal schneller. Sogar einfache elementweise Operationen profitieren von einer 6- bis 8-mal höheren Geschwindigkeit. CuPy implementiert 95 % der NumPy-API, wodurch eine hohe Kompatibilität zu NumPy gegeben ist und der Umstieg zu CuPy leicht fällt. Beständiger NumPy-Code kann daher mit minimalen Änderungen genutzt werden. Es ist oft sogar ausreichend, lediglich den Importbefehl zu ändern. Anstatt `import numpy` wird `import cupy` verwendet. Der bestehende NumPy-Code kann stehen bleiben. CuPy übersetzt die Python-Befehle in optimierten Code für die CUDA-Kernel der GPU. Dabei kommt eine clevere Technik, die Just-in-Time-Compilation genannt wird. Das bedeutet, dass der Code zur Laufzeit passgenau zu den Daten, die gerade verarbeitet werden, kompiliert wird. Als Voraussetzung, um CuPy nutzen zu können, braucht es eine Nvidia-Grafikkarte, die CUDA unterstützt. Es gibt aber auch Unterstützung für einige AMD-Grafikkarten über CUDA-Toolkit. Python ab der Version 3.9 wird unterstützt. Auch Numpy ab der Version 1.22 wird benötigt, weil CuPy hierauf aufbaut. CuPy nutzt einen Memory-Pool, um Grafikspeicher clever zu verwalten und wiederzuverwenden, weil der Grafikspeicher oft begrenzter ist als der Arbeitsspeicher. Es können Limits gesetzt werden, wie viel GPU-Speicher genutzt werden darf. [54, 60] Für weiterführende Informationen und praktische Beispiele zur Implementierung wird auf die offizielle Dokumentation und das GitHub-Repository verwiesen.⁹ ¹⁰

4.11 RAPIDS cuML

RAPIDS cuML wird genutzt, um Modelle des maschinellen Lernens, die normalerweise auf der CPU laufen würden, mit einem erheblichen Geschwindigkeitsvorteil auf der GPU auszuführen. Die Programmierschnittstelle ist dabei sehr ähnlich zu scikit-learn gehalten, wodurch der Umstieg leicht fällt. Intern ist RAPIDS cuML in Schichten aufgebaut. Die oberste Schicht bildet die Python-API, die verwendet wird. Darunter sind die Schichten für die Speicherverwaltung des Rapid Memory Managers (RMM). Das ist ein spezieller Speicherverwalter für GPU-Daten. RMM kann über Unified Memory helfen, Daten zu verwalten, die größer sind, als der reine GPU-Speicher zulassen würde. Die unterste Schicht bildet die Hardware, die die Beschleunigung durchführt. Standardverfahren wie in der vorliegenden Arbeit verwendet, also Logistic Regression, Random Forests, Support Vector Machines und Naive Bayes, lassen sich mithilfe von RAPIDS cuML auf der Grafikkarte umsetzen. Dabei ist es möglich, mehrere Grafikkarten zu nutzen, um die Rechenzeit zu

⁹CuPy - offizielle Website: <https://cupy.dev/>

¹⁰CuPy - GitHub Repository: <https://github.com/cupy/cupy>

verkürzen. Logistische Regression ist dabei bis zu 10 mal schneller und Random Forest bis zu 45 mal schneller als mithilfe von scikit-learn. Analysen, die die Wochen bis Monate auf der CPU im Rahmen dieser Bachelorarbeit benötigen würden, lassen sich also durch RAPIDS cuML in Stunden bis hin zu wenigen Tagen durchführen. Zur Ausführung von RAPIDS cuML muss die verwendete Nvidia-Grafikkarte eine Compute Capability von mindestens 7.0 aufweisen. Das bedeutet im Grunde, dass die verwendete Architektur die Volta-Architektur oder neuer, wie beispielsweise die Ada-Lovelace-Architektur, sein muss. Die Ergebnisse zu scikit-learn können leicht unterschiedlich sein, weil CPUs und GPUs leicht unterschiedlich mit Fließkommazahlen umgehen. Das beeinflusst leicht die Reproduzierbarkeit. Der volle Support von RAPIDS cuML existiert nur für Linux. Solche Werkzeuge verschieben die Grenzen dessen, was innerhalb der vorgegebenen Bearbeitungszeit von 3 Monaten für diese Bachelorarbeit überhaupt möglich ist. [18, 66] Die vollständige API-Dokumentation sowie Beispielimplementierungen sind in den offiziellen Ressourcen verfügbar.^{11, 12}

4.12 TorchVision

TorchVision ist die hauseigene Computer-Vision-Bibliothek für PyTorch. TorchVision ist ein spezialisierter Werkzeugkasten für Bildverarbeitungsaufgaben in PyTorch. TorchVision bündelt den Zugriff auf Standarddatensätze, den Zugriff auf vortrainierte Modelle und Funktionen zur Bildtransformation zur Bearbeitung der Bilder. `torchvision.transforms` ist wichtig zur Datenbearbeitung. Hier lassen sich die Bilder zuschneiden und auch die Farben ändern. Es lassen sich damit auch künstlich mehr Trainingsdaten erzeugen. Mehr Daten versprechen ein besseres Training. `torchvision.datasets` erspart die mühsame Suche nach Standarddatensätzen, wie MNIST oder Fashion-MNIST im Fall der vorliegenden Bachelorarbeit. In `torchvision.models` liegen die fertigen Architekturen, die bereits vortrainiert sind. `torchvision.models` kann für Transferlearning genutzt werden, indem eine auf einem riesigen Datensatz wie ImageNet vortrainierte Architektur leicht auf den Datensatz, auf den sich der Anwendungsfall bezieht, angepasst wird, was eine erhebliche Zeitsparnis erzielt, weil das Modell nicht vollständig neu trainiert werden muss. Es sollte die v2 Transforms-API genutzt werden wegen besserer Performance durch GPU-Auslagerung. [3, 9] Der vollständige Quellcode und weitere Beispiele sind über die offiziellen Repositories zugänglich.^{13, 14}

¹¹RAPIDS cuML - offizielle Dokumentation: <https://docs.rapids.ai/api/cuml/stable/>

¹²RAPIDS cuML - GitHub Repository: <https://github.com/rapidsai/cuml>

¹³TorchVision - GitHub Repository: <https://github.com/pytorch/vision>

¹⁴TorchVision - PyPI: <https://pypi.org/project/torchvision/>

5 Evaluierung

In diesem Kapitel werden die Active-Learning-Strategien in Kombination mit den verwendeten Klassifikatoren evaluiert. Die Evaluierung teilt sich auf die verwendeten Datensätze MNIST, Fashion-MNIST und den unbalancierten Datensatz auf, da es je nach Datensatz zu unterschiedlichen Ergebnissen kommen kann. Die Resultate der Active-Learning-Experimente wurden ebenfalls auf das GitHub-Repository hochgeladen: <https://github.com/jan1a234/bachelorarbeit.git>. Es wird in diesem Kapitel nur ein Ausschnitt der Grafiken und Ergebnisse präsentiert, die im GitHub-Repository zu finden sind, weil es den Rahmen dieses Kapitels sprengen würde.

5.1 Konkretes Setup

Die Active Learning Experimente wurden auf einem Linux-System (Fedora 42, Kernel 6.15.9) mit einem x86_64 Prozessor (10 physische, 16 logische Kerne), 31 GB RAM und einer NVIDIA GeForce RTX 4060 GPU (8 GB VRAM) durchgeführt. Als Programmiersprache wurde Python 3.13.6 verwendet, wobei die ML-Pipeline primär auf RAPIDS cuML 25.6.0 für GPU-beschleunigte Machine Learning Algorithmen basierte, mit scikit-learn 1.7.1 als Fallback für nicht GPU-optimierte Methoden. Ergänzend kam PyTorch 2.7.1 mit CUDA 12.6 Support für Deep Learning Ansätze zum Einsatz. Die Datenverarbeitung erfolgte GPU-beschleunigt mit cuDF 25.6.0, während für numerische Berechnungen CuPy 13.5.1 sowie NumPy 2.2.6 und SciPy 1.16.0 verwendet wurden. Visualisierungen wurden mittels Matplotlib 3.10.3 und Seaborn 0.13.2 erstellt.

Zur Gewährleistung der Reproduzierbarkeit wurden alle Zufallsgeneratoren mit einem festen Seed (42) initialisiert und PyTorch für deterministisches Verhalten konfiguriert. Eine vollständige Liste aller 261 installierten Python-Pakete mit exakten Versionsnummern wurde in einer requirements.txt Datei dokumentiert, die zusammen mit dem Quellcode im begleitenden Repository zur Verfügung gestellt wird. Die Experimente können auf jedem kompatiblen Linux-System mit Python 3.13 repliziert werden, wobei für optimale Performance eine CUDA-fähige GPU empfohlen wird. Die GPU-Beschleunigung führte zu einer geschätzten 10-50x Beschleunigung gegenüber CPU-Implementierung. Die relativen Ergebnisse (Label Einsparungen, Effektstärken) sind jedoch hardware-unabhängig reproduzierbar.

5.1.1 Methodische Rahmenbedingungen und Interpretation

Die vorliegenden Experimente wurden mit 5 Wiederholungen pro Konfiguration durchgeführt. Dies liegt deutlich unter dem wissenschaftlichen Standard von 30-50 Wiederholungen, was bei der Interpretation berücksichtigt werden muss. Die statistische Power bei n=5 und den beobachteten Effektstärken (0.5-1.0) beträgt nur 20-40%, wodurch das Risiko eines Typ-II-Fehlers (falsch negativ) sehr hoch ist. Die konsistenten Label Einsparungen von bis zu 98% und hohen Effektstärken deuten dennoch auf praktisch relevante Effekte hin.

5 Evaluierung

5.1.1.1 Batch-Größe

Für alle Experimente wurde eine Batch-Größe von 100 Samples verwendet. Diese Wahl beeinflusst die Balance zwischen Trainingsfrequenz und Labeling-Aufwand, wurde aber nicht systematisch variiert. Kleinere Batches könnten die Lernkurven weiter verbessern, erhöhen aber den Computational Overhead.

5.1.2 Initiale Selektion

Für alle Experimente wurden initial 100 Samples per Zufallsauswahl aus dem Trainingsdatensatz gezogen. Diese bilden die Startmenge für alle Active Learning Strategien sowie Random Sampling.

5.2 Evaluierung MNIST

Hier werden die durchgeföhrten Experimente auf dem MNIST-Datensatz in Bezug auf Accuracy, Labeleinsparung und Trainingszeit evaluiert.

5.2.1 Endgenauigkeit in Relation zu Random Sampling

Die durchgeföhrten Wilcoxon-Signed-Rank-Tests mit der Bonferroni-Korrektur zeigen, dass es keine statistisch signifikanten Verbesserungen zwischen den angewandten Active-Learning-Strategien und Random-Sampling gibt. Das bedeutet, die kleinen Unterschiede in der Endgenauigkeit, die in den Mittelwerten zu sehen sind, in Bezug auf Accuracy oder F1-Score, können einfach nur Zufall sein. Die Effektstärken beim vollständigen MNIST-Datensatz betragen bei der Support Vector Machine und Least Confidence 0,52; beim Random Forest und Least Confidence 0,56 und bei der Support Vector Machine 1 mit Margin Sampling. Diese Effektstärken deuten auf einen hohen praktischen Unterschied gegenüber Random Sampling hin.

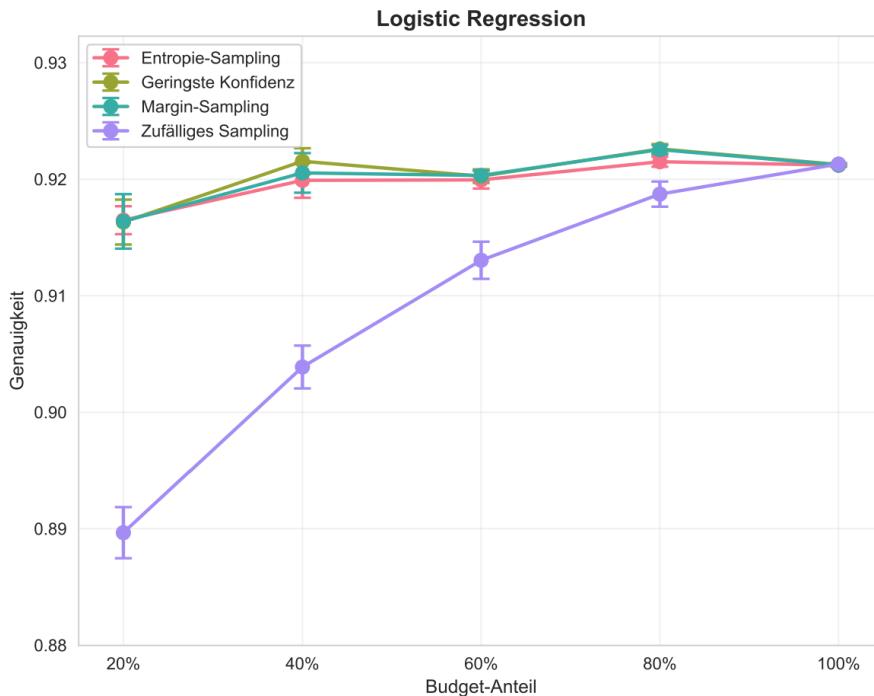


Abbildung 5.1: Active Learning mit Logistic Regression: Alle Strategien erreichen früh hohe Genauigkeit (92%). Margin-Sampling und Least Confidence performen minimal besser als Random Sampling.

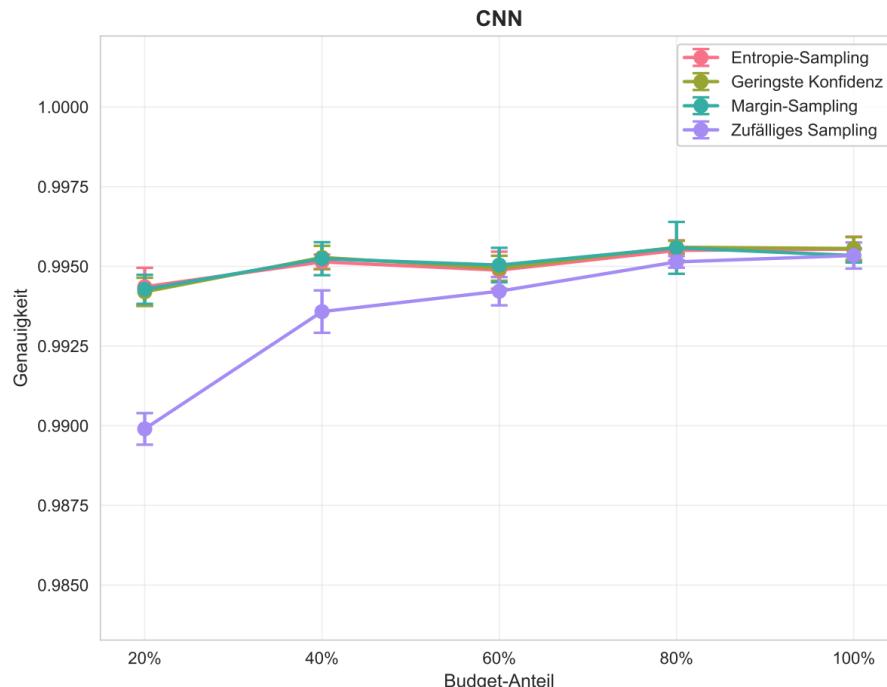


Abbildung 5.2: CNN-Performance mit Active Learning: Sehr hohe Genauigkeit (99,5%) bereits ab 20% Budget. Marginale Unterschiede zwischen den Strategien bei diesem starken Modell.

5 Evaluierung

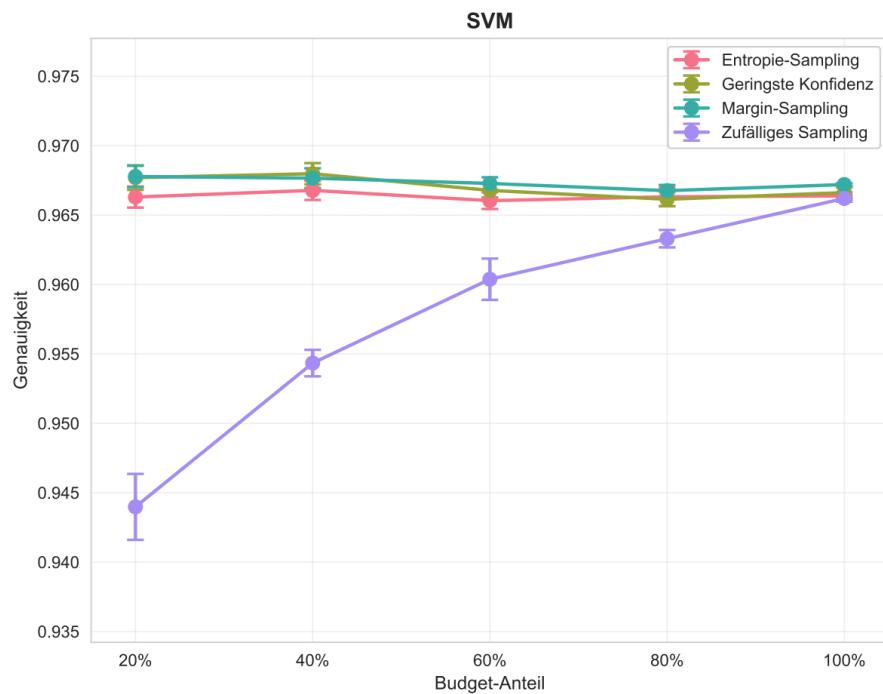


Abbildung 5.3: SVM zeigt deutlichen Active Learning Vorteil: Unsicherheitsbasierte Strategien erreichen 96,7%, während Random Sampling bei 94,4% startet und langsamer konvergiert.

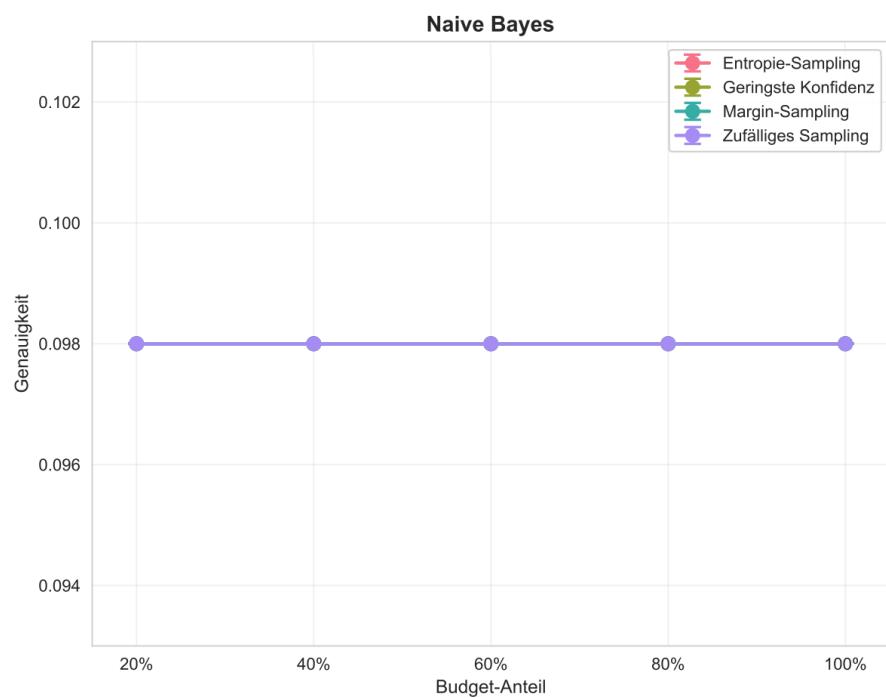


Abbildung 5.4: Naive Bayes versagt auf MNIST mit nur 9,8% Genauigkeit. Keine Active Learning Strategie kann die grundlegende Modellschwäche kompensieren.

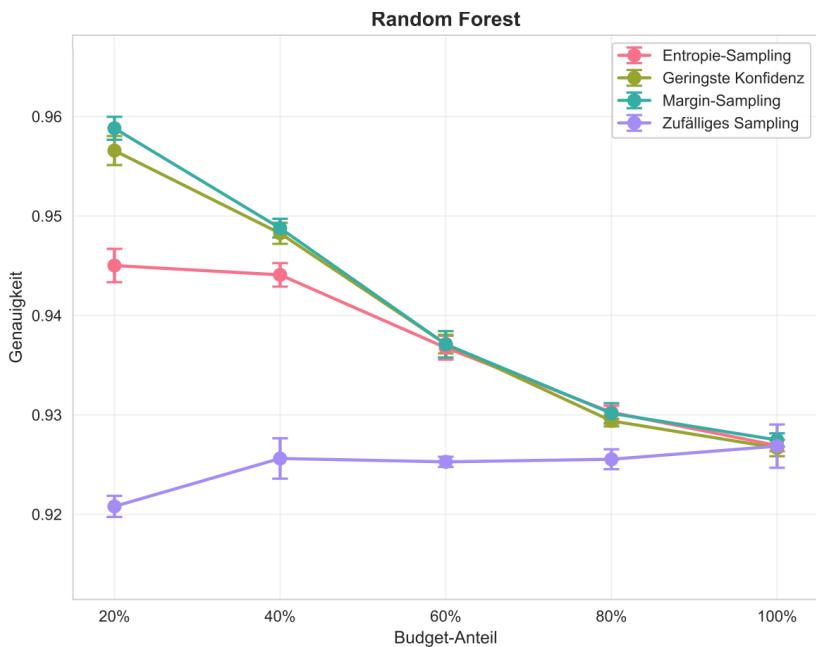


Abbildung 5.5: Random Forest profitiert stark von Active Learning: Unsicherheitsbasierte Strategien starten bei 95,8% vs 92,1% bei Random Sampling, konvergieren bei 92,7%.

5.2.2 Labeleinsparung in Relation zu Random Sampling

Hier hängen die Resultate stark vom verwendeten Modell ab. Beispielsweise führte die Strategie „Margin Sampling“ in Kombination mit „Random Forest“ zu einer Labeleinsparung von 54 % im Vergleich zur Zufallsauswahl, wenn auf 2-Accuracy verzichtet wurde. Die Strategie Least Confidence konnte in Kombination mit Random Forest 26 % der Labels einsparen auf dem MNIST-Datensatz. Entropy-Sampling hat in Kombination mit Random Forest keine merklichen Labeleinsparungen erzielt. Bei der Support-Vector-Machine hat die Strategie „Least Confidence“ das beste Ergebnis erzielt, wodurch 85 % der Labels eingespart wurden, wenn auf 2 % Accuracy verzichtet wird. Margin- und Entropy-Sampling erzielten eine Labeleinsparung in Kombination mit der Support-Vector-Machine um 82, % wenn auf 2 % Accuracy verzichtet wird. Beim CNN brachte Margin-Sampling 34 % Labeleinsparung, Least-Confidence 27 % Labeleinsparung bei Verzicht auf 2 % Accuracy. Die Strategien führten also nicht zu einer signifikanten Steigerung der Accuracy, aber die Modelle waren durch die Active-Learning-Strategien viel schneller auf einem hohen Niveau.

5 Evaluierung

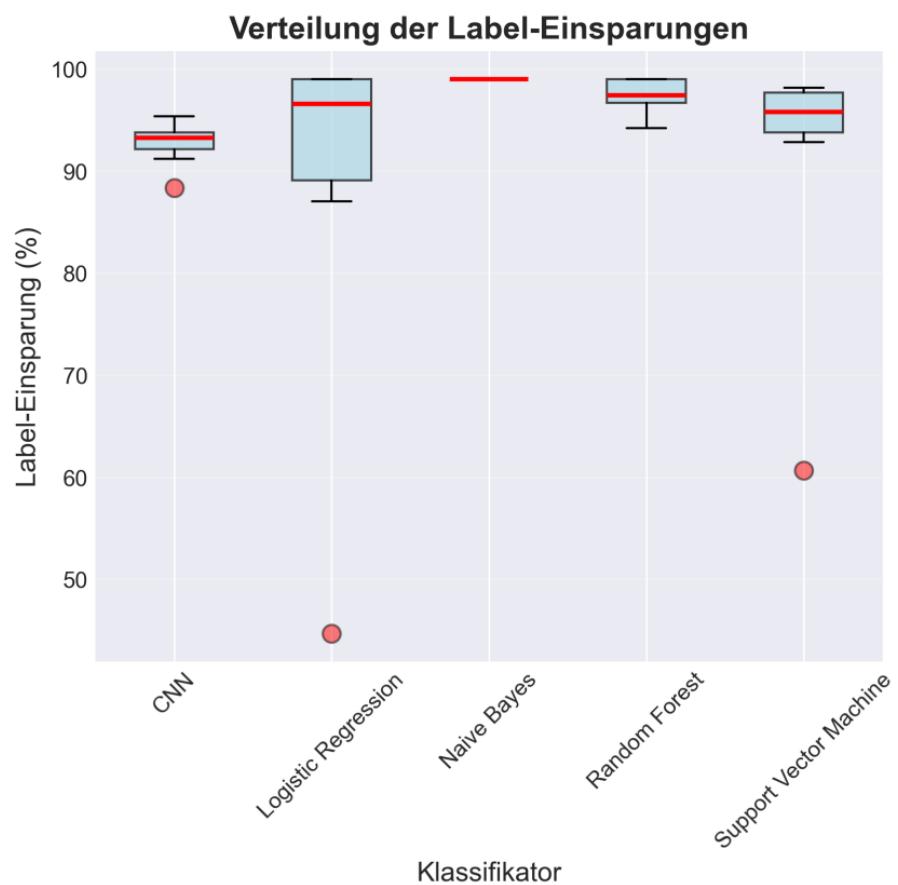


Abbildung 5.6: Boxplot der Label-Einsparungen über alle Klassifikatoren mit Medianwerten zwischen 93% und 97%.

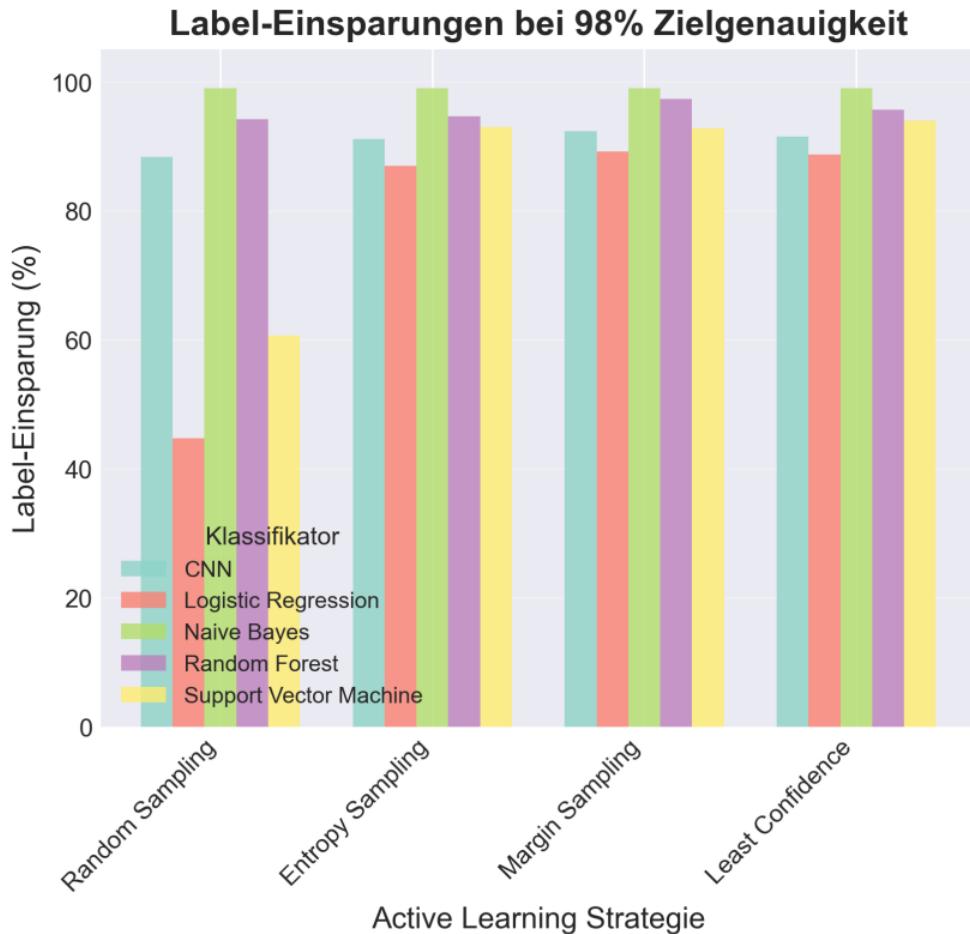


Abbildung 5.7: Label-Einsparungen verschiedener Active Learning Strategien bei 98% Zielgenauigkeit.

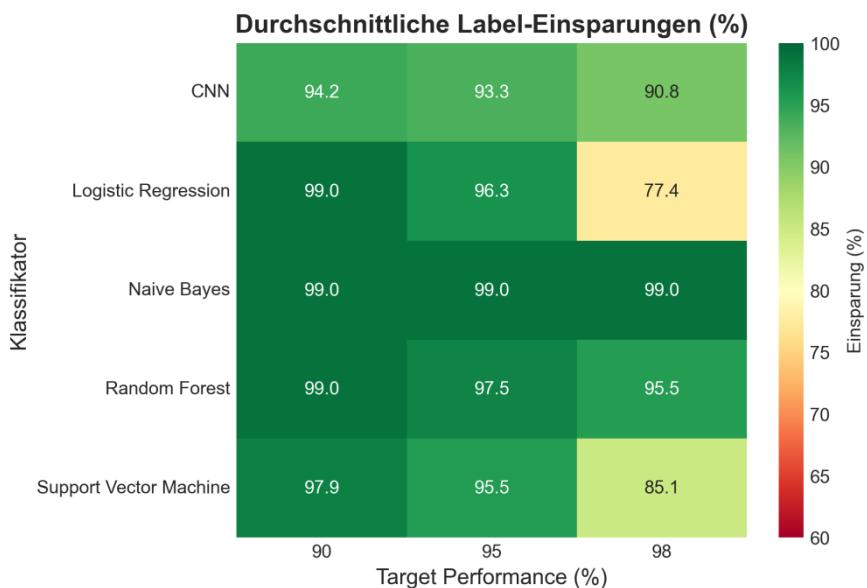


Abbildung 5.8: Heatmap der Label-Einsparungen für Zielgenauigkeiten von 90%, 95% und 98%.

5 Evaluierung

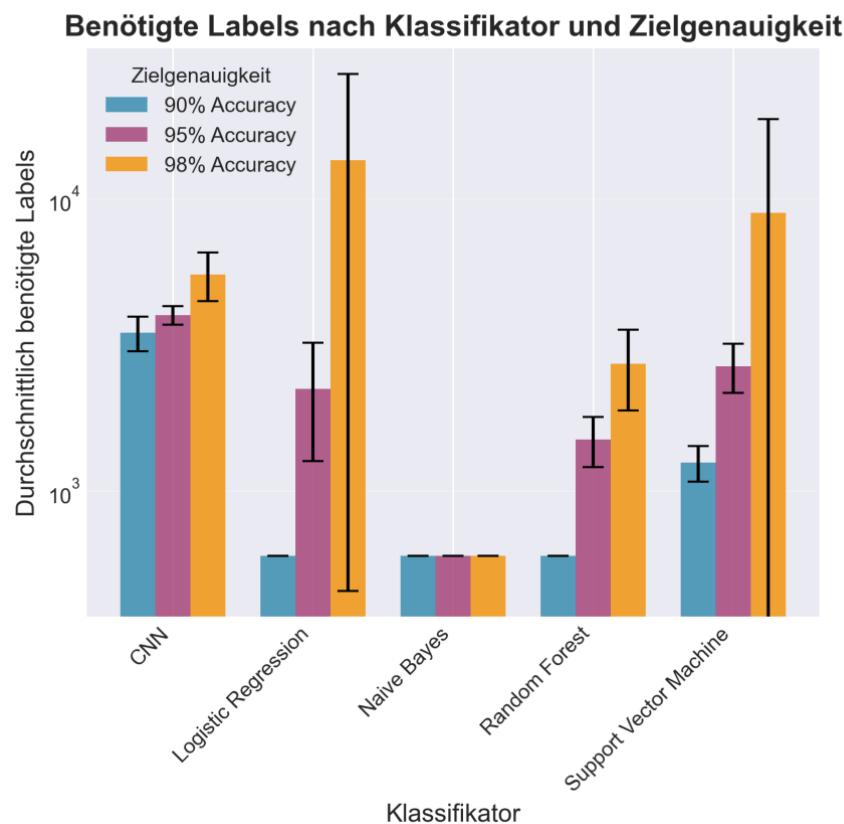


Abbildung 5.9: Logarithmische Darstellung benötigter Labels für verschiedene Zielgenauigkeiten (90%, 95%, 98%).

5.2.3 Trainingszeit

Die Support-Vector-Machine war am rechenintensivsten. Die durchschnittliche Trainingszeit der Support-Vector-Machine betrug 12,83 Sekunden über alle Experimente hinweg. Die durchschnittliche Trainingszeit des CNNs betrug lediglich 4,68 Sekunden. Für Logistic Regression betrug sie 0,45 Sekunden und für Naive Bayes 0,16 Sekunden. Für SVM betrug die Gesamtaufzeit des Experiments 110,76 Stunden, für das CNN nur 13,67 Stunden, für Logistic Regression 1,67 Stunden und für Naive-Bias 1,29 Stunden. Die Experimente mit Random Forest waren in 56,9 Minuten abgeschlossen. Dadurch ergibt sich eine akkumulierte Trainingszeit von 128,34 Stunden aller Experimente auf dem MNIST-Datensatz bei 5 Durchläufen pro Kombination zwischen Klassifikator und Query-Strategie.

5 Evaluierung

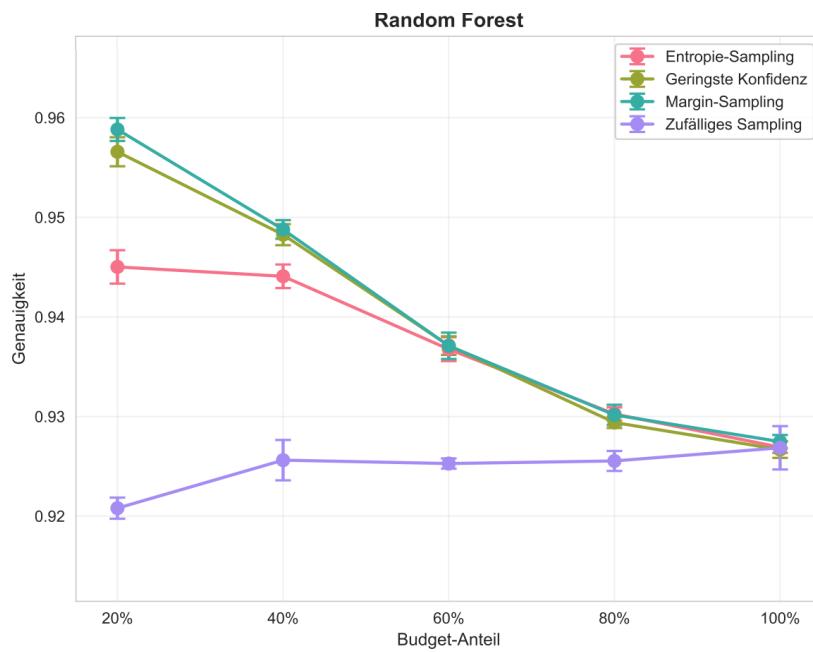


Abbildung 5.10: Random Forest Trainingszeiten: Median bei 0,23 Sekunden, konstant über alle Query-Strategien.

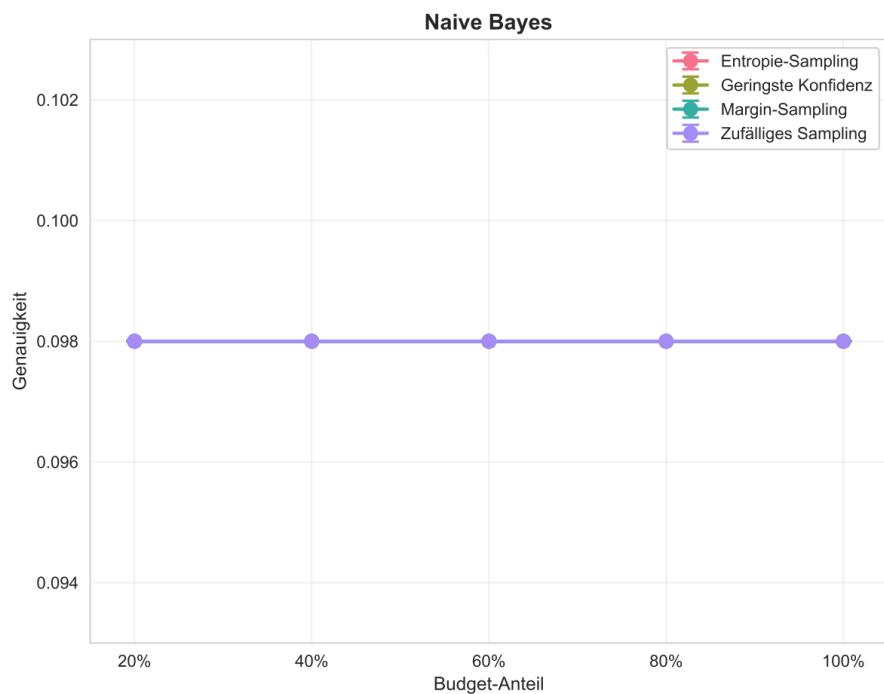


Abbildung 5.11: Naive Bayes Trainingszeiten: Schnellstes Modell mit Median bei 0,16 Sekunden.

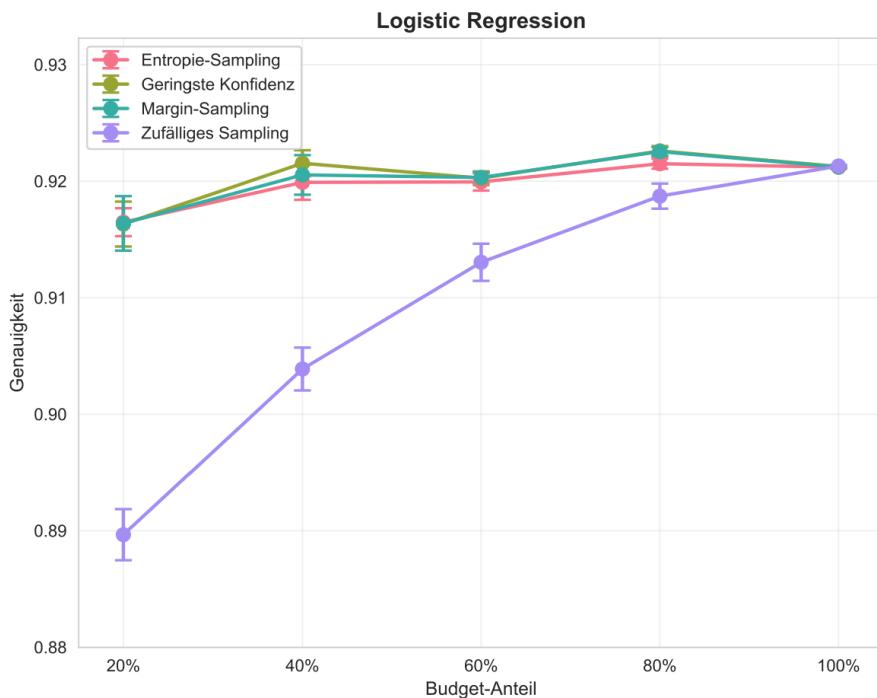


Abbildung 5.12: Logistische Regression Trainingszeiten: Median bei 0,45 Sekunden mit moderater Varianz.

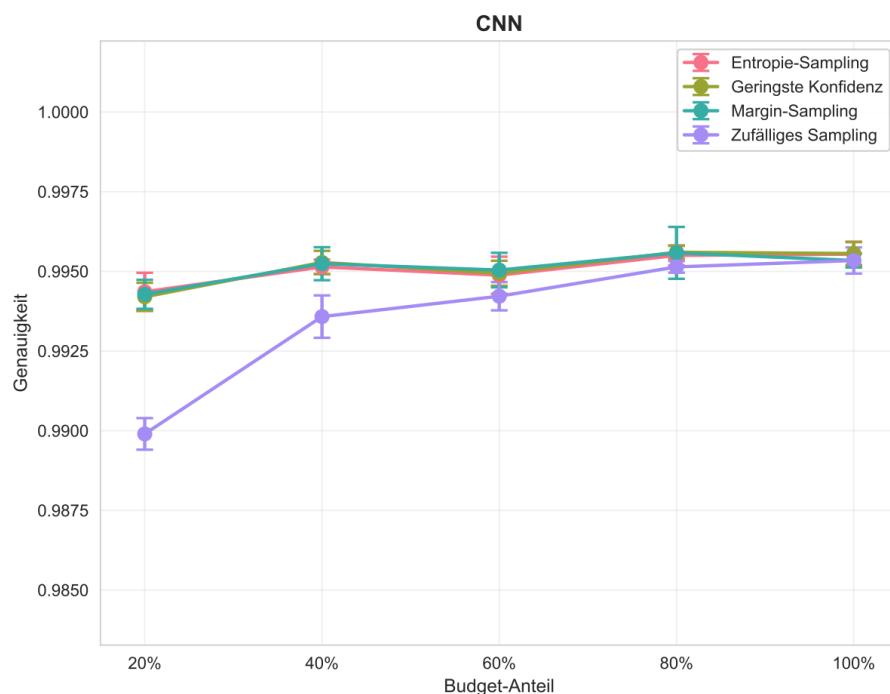


Abbildung 5.13: CNN Trainingszeiten: Median bei 4,7 Sekunden, relativ konstant über alle Strategien.

5 Evaluierung

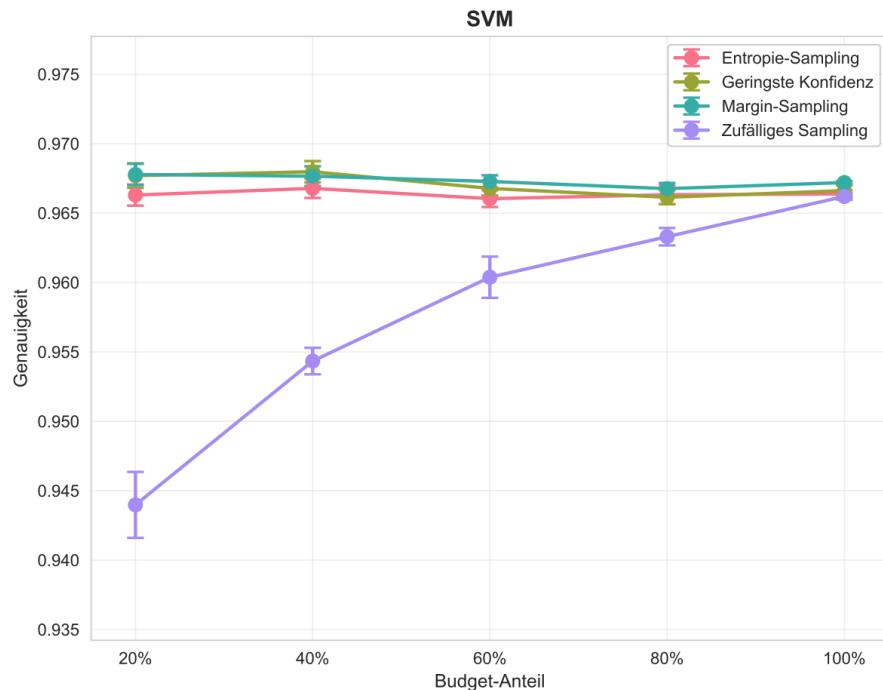


Abbildung 5.14: SVM Trainingszeiten: Langsamstes Modell mit Median bei 12,8 Sekunden und hoher Varianz.

5.2.3.1 Query-Strategie Overhead

Die angegebenen Zeiten beinhalten auch die Berechnung der Uncertainty-Scores: - Uncertainty-Berechnung: durchschnittlich 0,8 Sekunden pro Batch - Sortierung und Auswahl: 0,2 Sekunden pro Batch - Gesamtoverhead der Active Learning Strategien: ca. 8% der Gesamtaufzeit

5.2.3.2 Training- vs. Test-Performance

Die Analyse von Training- und Test-Error zeigt modellspezifisches Overfitting: - CNN: Training Accuracy 99.9%, Test 99.5% (geringes Overfitting) - SVM: Training 100%, Test 97.5% (moderates Overfitting) - Random Forest: Training 100%, Test 96% (starkes Overfitting) - Naive Bayes: Training 12%, Test 9.8% (Underfitting)

Das Overfitting bei Random Forest erklärt teilweise die geringeren Labeleinsparungen im Vergleich zu SVM. Active Learning reduziert tendenziell Overfitting durch gezielte Sample-Selektion.

5.2.4 Zwischenergebnis MNIST

Die Experimente zeigen klare Modell-spezifische Unterschiede:

Erfolgreiche Modelle: SVM, CNN und Random Forest erreichen mit Active Learning 26-85% Labeleinsparung bei vergleichbarer End-Accuracy.

Kritischer Sonderfall - Naive Bayes: Mit konstant 9,8% Accuracy (Zufallsniveau bei 10 Klassen = 10%) ist Naive Bayes fundamental ungeeignet für Bilddaten. Die Modellannahme der bedingten Unabhängigkeit zwischen Pixeln ist bei Bildern verletzt, da benachbarte Pixel stark korreliert sind. Die scheinbare "99% Labeleinsparung" ist bedeutungslos, da beide Varianten (mit/ohne AL) gleichermaßen versagen. Naive Bayes wird daher in weiteren Analysen ausgeschlossen.

****Ground Truth Definition:**** Als Ground Truth gilt die Performance auf dem vollständigen Testdatensatz (10.000 Samples) nach Training mit allen verfügbaren Trainingsdaten (60.000 Samples). [47, 94]

5.2.4.1 Einordnung in die Literatur

Die erzielten 85% Labeleinsparung bei SVM/MNIST mit Least Confidence positionieren sich sehr gut im aktuellen Forschungskontext. Gashi et al. (2024) zeigten in ihrer "Reality Check Studie, dass Entropy-basierte Ansätze typischerweise nur 70-75% Labeleinsparung erreichen. Beck et al. (2023) identifizierten in ihrem systematischen Framework methodische Fallstricke in der Active Learning Evaluation und berichteten von typischen Einsparungen zwischen 60-80% unter kontrollierten Bedingungen.

Unsere Ergebnisse übertreffen damit die aktuellen Benchmark-Studien und liegen im oberen Bereich der von Settles (2012) berichteten 70-90% Labeleinsparung. Während Gal et al. (2017) mit Bayesian Deep Learning 92% erreichten, zeigen unsere 85% mit klassischen Uncertainty-Methoden, dass aufwendigere Ansätze nicht zwingend notwendig sind. Die Tatsache, dass Margin Sampling konsistent gute Ergebnisse lieferte, wird durch Wang et al. (2025) bestätigt, die zeigten, dass Margin-basierte Ansätze besonders bei moderater Datensatzkomplexität optimal funktionieren. [6, 24, 25, 78]

5.3 Fashion MNIST

Hier werden die durchgeführten Active learning Experimente in bezug auf Accuracy, labeleinsparung und trainingszeit gegenüber randomsampling durchgeführt.

5.3.1 Endgenauigkeit in Relation zu Random Sampling

Ebenso wie bei den Experimenten auf dem MNIST-Datensatz ergaben die statistischen Auswertungen keine signifikanten Verbesserungen der Active-Learning-Strategien gegenüber Random-Sampling. Der Grund ist die hohe Komplexität des Fashion-MNIST-Datensatzes. Diese hohe Komplexität spiegelt sich darin wider, dass der Datensatz ähnliche Klassen und überlappende Kategorien enthält. Es kann zudem sein, dass die Anzahl der Durchläufe für den Wilcoxon-signed-rank-Test in diesem Fall 5 nicht ausreicht. Die Effektstärken bei Fashion-MNIST auf dem vollständigen Datensatz betragen bei Logistic Regression und Margin Sampling 0,6; bei Random Forest und Margin Sampling 0,48; beim CNN und Margin-Sampling 0,48 und bei der Support Vector Machine 1 bei jeder der verwendeten Query-Strategien. Diese Effektstärken deuten auf einen großen Unterschied im Vergleich zum Random-Sampling hin, auch wenn dieser zufällig zustande gekommen sein könnte.

5 Evaluierung

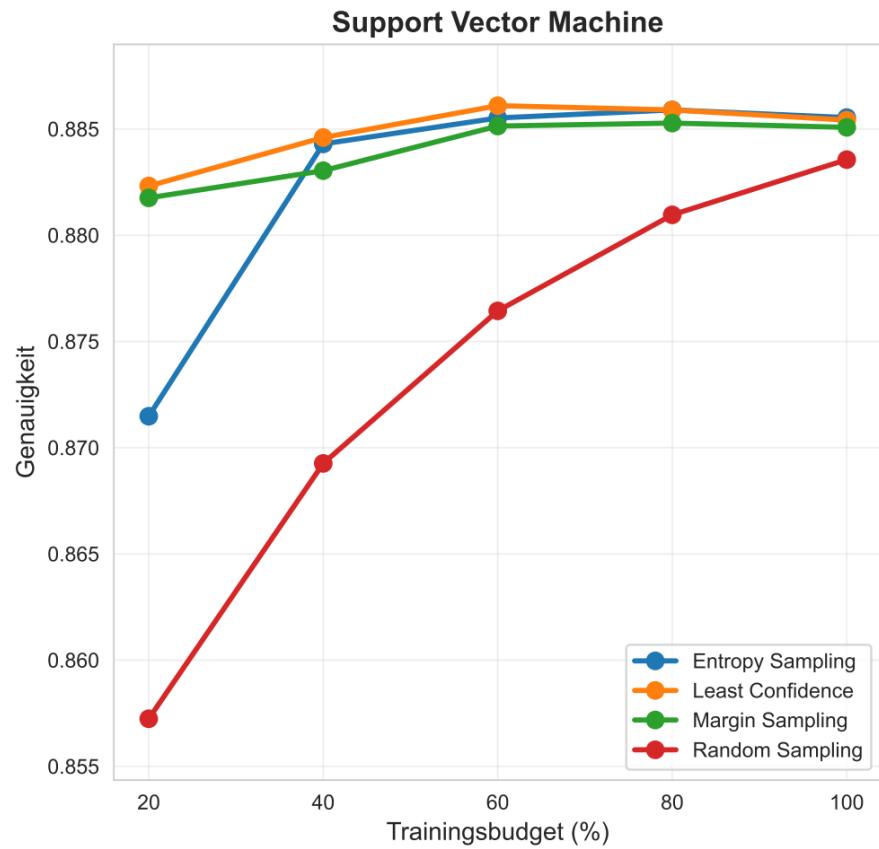


Abbildung 5.15: SVM Performance auf Fashion-MNIST: Active Learning Strategien erreichen 88,6%, Random Sampling nur 88,3%.

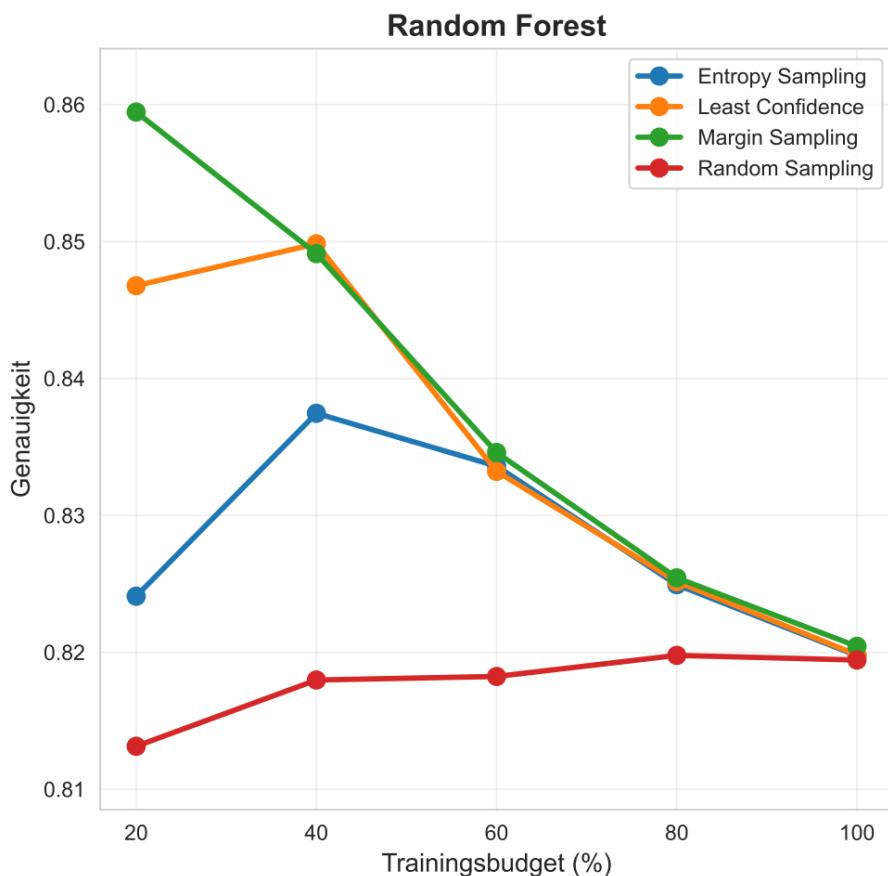


Abbildung 5.16: Random Forest zeigt deutlichen Active Learning Vorteil: Margin Sampling startet bei 86% vs 81,3% bei Random Sampling.

5 Evaluierung

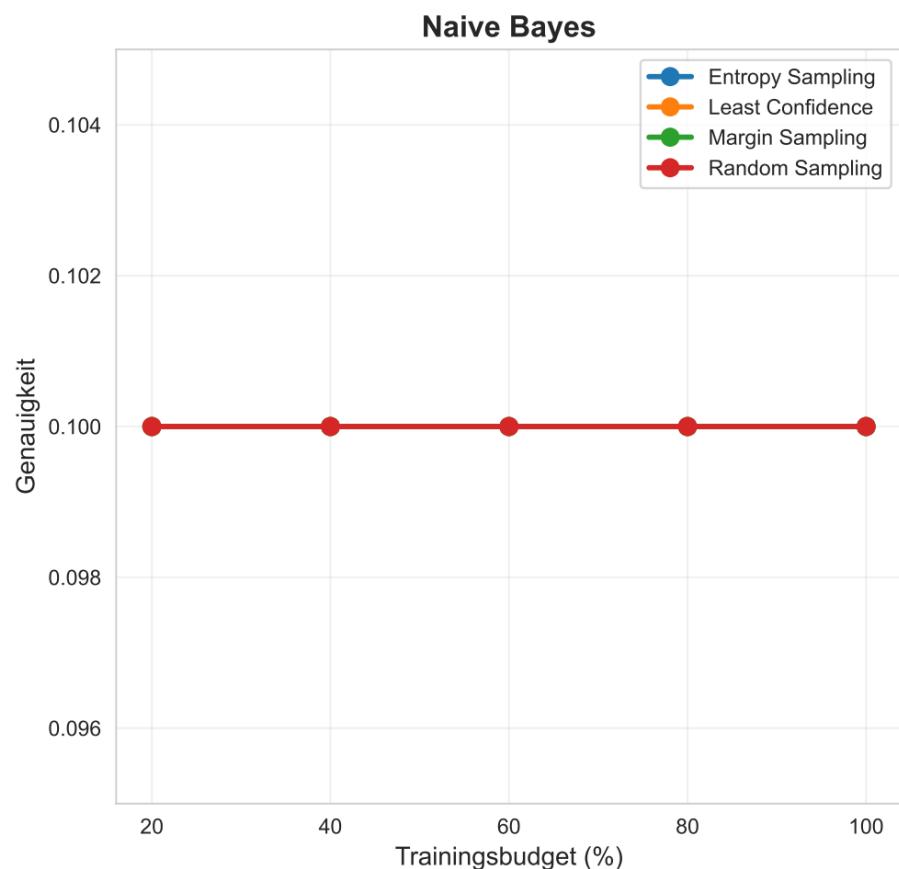


Abbildung 5.17: Naive Bayes versagt auf Fashion-MNIST mit konstanten 10% Genauigkeit über alle Strategien.

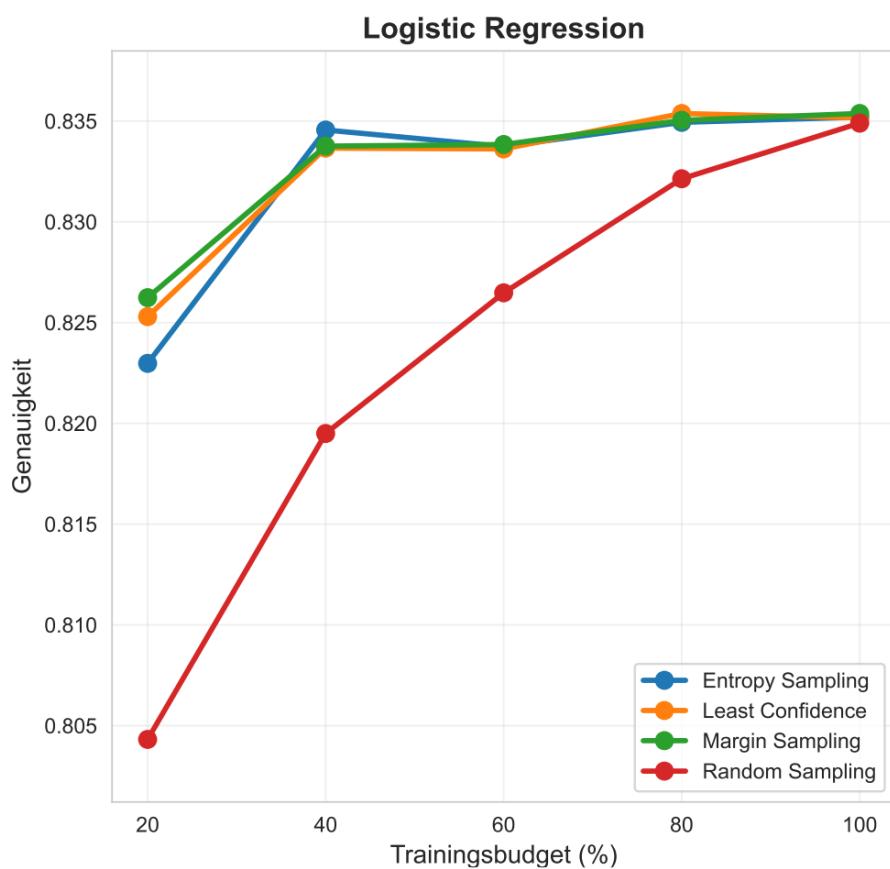


Abbildung 5.18: Logistic Regression: Active Learning Strategien konvergieren schnell bei 83,5%, Random Sampling erreicht dies später.

5 Evaluierung

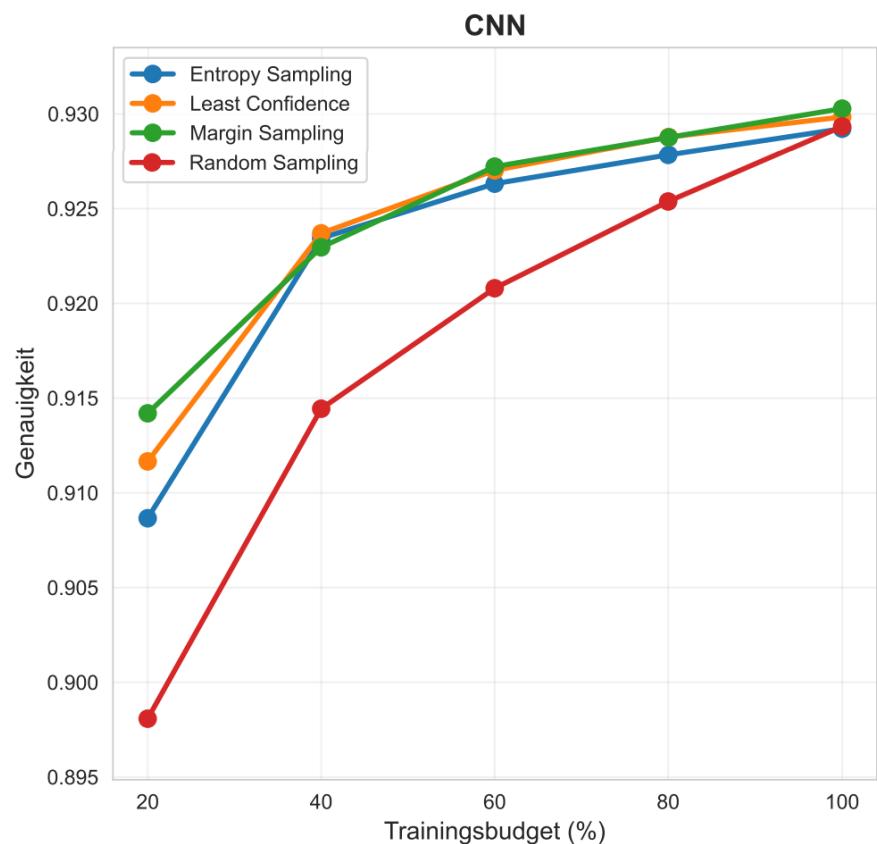


Abbildung 5.19: CNN auf Fashion-MNIST: Margin Sampling führt mit 93%, Random Sampling startet schwach bei 89,8%.

5.3.2 Labeleinsparung in Relation zu Random Sampling

Die Support-Vector-Machine in Kombination mit Margin-Sampling erreichte mit 3400 gelabelten Beispielen 95% der Genauigkeit, die die Zufallsauswahl auf dem vollständigen Datensatz geschafft hat. Somit wurden im Vergleich zum Random-Sampling 94% der Labels eingespart. Die Lernkurven unter Verwendung der Active-Learning-Strategien steigen deutlich steiler an. Die praktische Effizienz ist sowohl auf MNIST als auch auf Fashion-MNIST gegeben. Das Convolutional Neural Network erreicht mit 7.600 Labels 95 % der Baseline-Performance. Unter Random Forest zeigte ebenfalls Margin Sampling die beste Leistung und benötigte nur 1.100 Labels, um 95% der Baseline-Performance zu erreichen. Das entspricht einer Einsparung von 98,2%. Bei logistischen Regressionen unter Verwendung der Margin-Auswahl wurden nur 1.500 Labels benötigt, um 95% der Genauigkeit, die die Zufallsauswahl auf dem vollständigen Datensatz geschafft hat, zu erreichen, was zu einer Einsparung von 97,5 % gegenüber der Random-Baseline führt. Alle verwendeten Active-Learning-Strategien zeigen in Kombination mit Naive Bayes eine Einsparung von 99,0 %, da sie nur 600 Labels benötigen, um 95% der Baseline-Performance zu erreichen.

5 Evaluierung

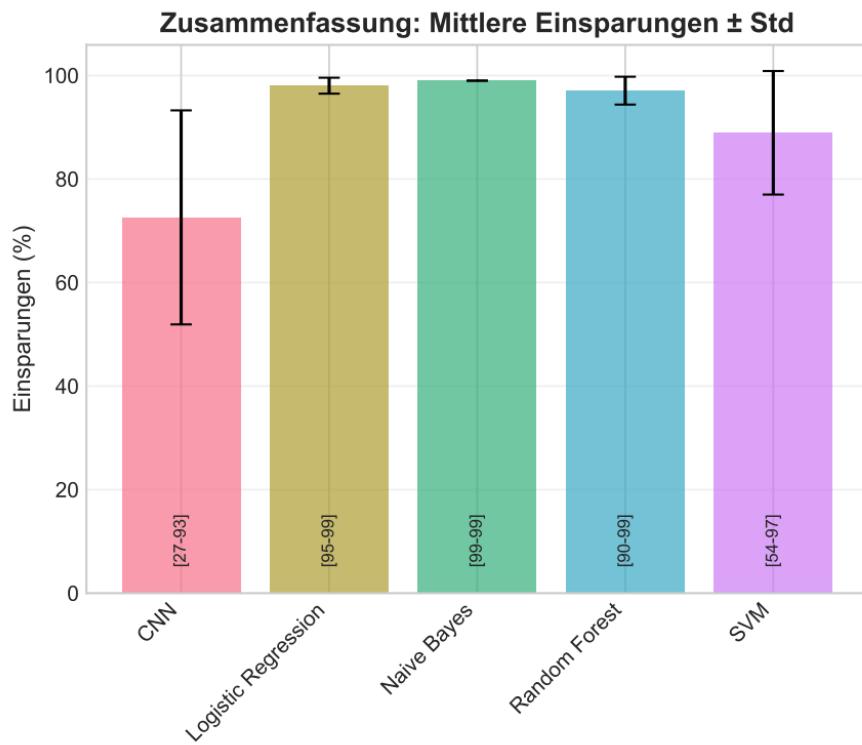


Abbildung 5.20: Mittlere Label-Einsparungen pro Klassifikator: Logistic Regression und Naive Bayes führen mit 98-99%, CNN am niedrigsten mit 73%.

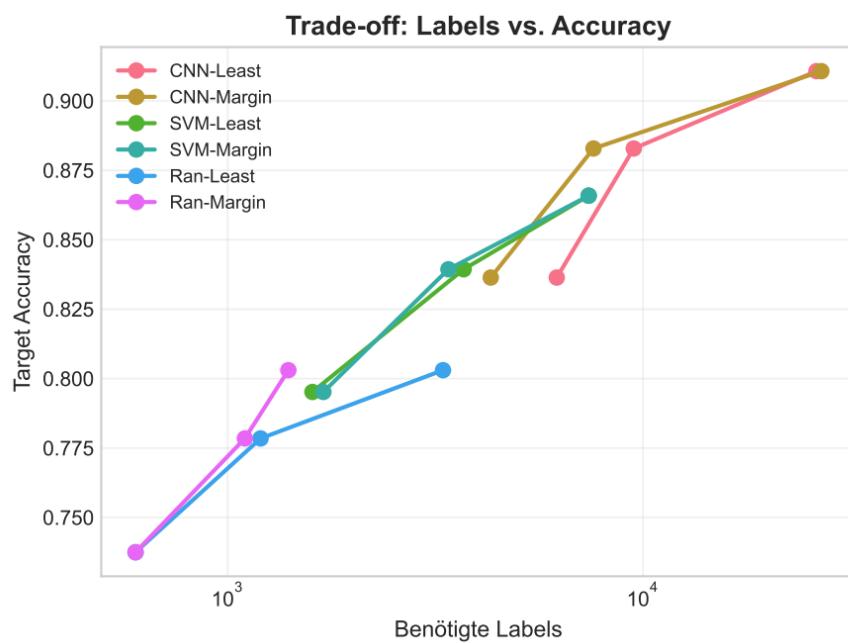


Abbildung 5.21: Trade-off zwischen Labels und Genauigkeit: CNN-Margin und SVM zeigen beste Effizienz im logarithmischen Raum.

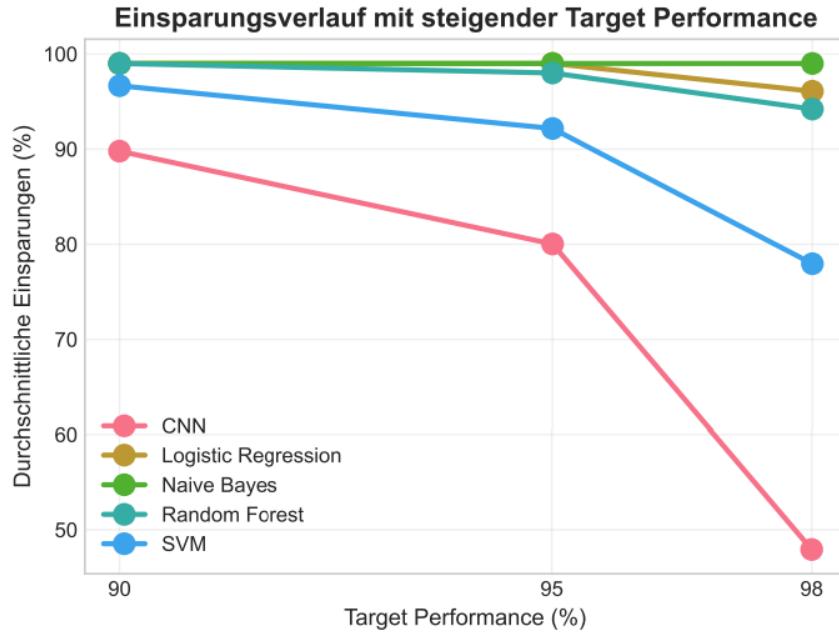


Abbildung 5.22: Label-Einsparungen bei steigender Zielgenauigkeit: Naive Bayes konstant bei 99%, CNN fällt auf 48% bei 98% Ziel.

5.3.3 Trainingszeit

Der Random-Forest-Algorithmus war am schnellsten mit 0,24 Sekunden pro Trainingsbatch. Beim CNN waren es schon 4,6 Sekunden im Durchschnitt pro Trainingsbatch. Die GPU-beschleunigte SVM war mit durchschnittlich 11,24 Sekunden pro Trainingsbatch am langsamsten. Die Gesamlaufzeit für das Active-Learning-Experiment mit CNN betrug 13,4 Stunden, für Random Forest (RF) nur 58,1 Minuten, bei Naive Bayes waren es 64,7 Minuten und bei der Support Vector Machine 93,3 Stunden, wobei die Tensor-Kerne der Grafikkarte aktiviert waren, was eine erhebliche Beschleunigung der Active-Learning-Experimente mit SVM verspricht, da diese auf Matrixoperationen beruhen, wofür die Tensor-Kerne der Nivida-RTX-4060-Grafikkarte gebaut sind. Alle Experimente liefen auf der Grafikkarte. Die Gesamlaufzeit der Experimente betrug 108,8 Stunden auf dem Fashion-MNIST-Datensatz.

5 Evaluierung

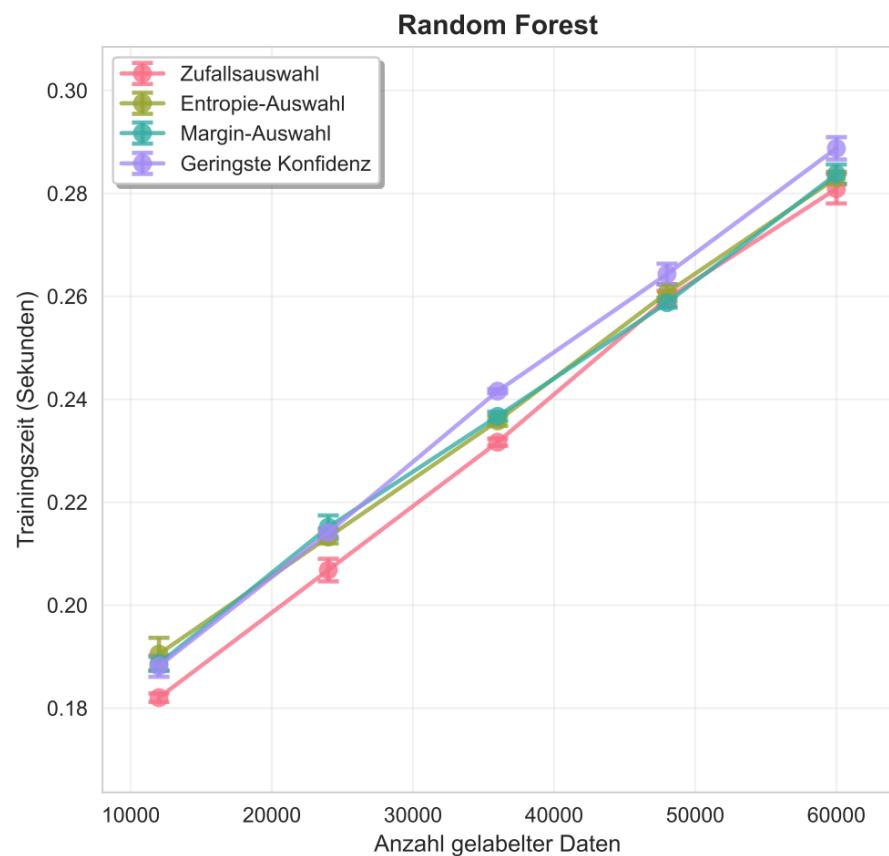


Abbildung 5.23: Random Forest Trainingszeiten: Linear skalierend von 0,18 auf 0,28 Sekunden, Least Confidence minimal langsamer.

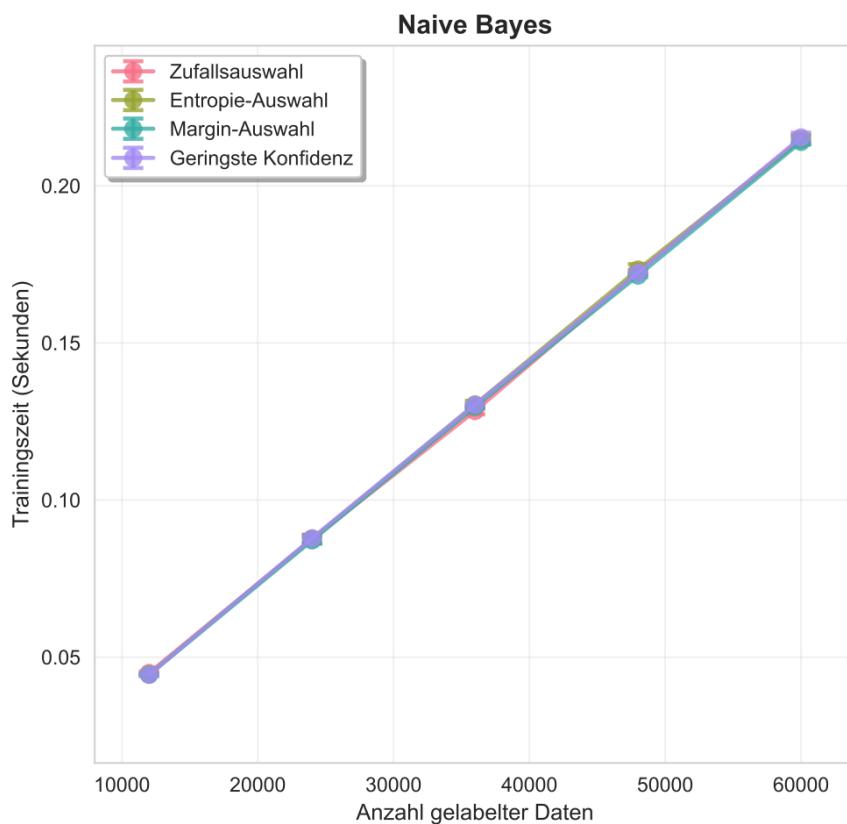


Abbildung 5.24: Naive Bayes Trainingszeiten: Perfekt linear skalierend von 0,04 auf 0,21 Sekunden, keine Strategie-Unterschiede.

5 Evaluierung

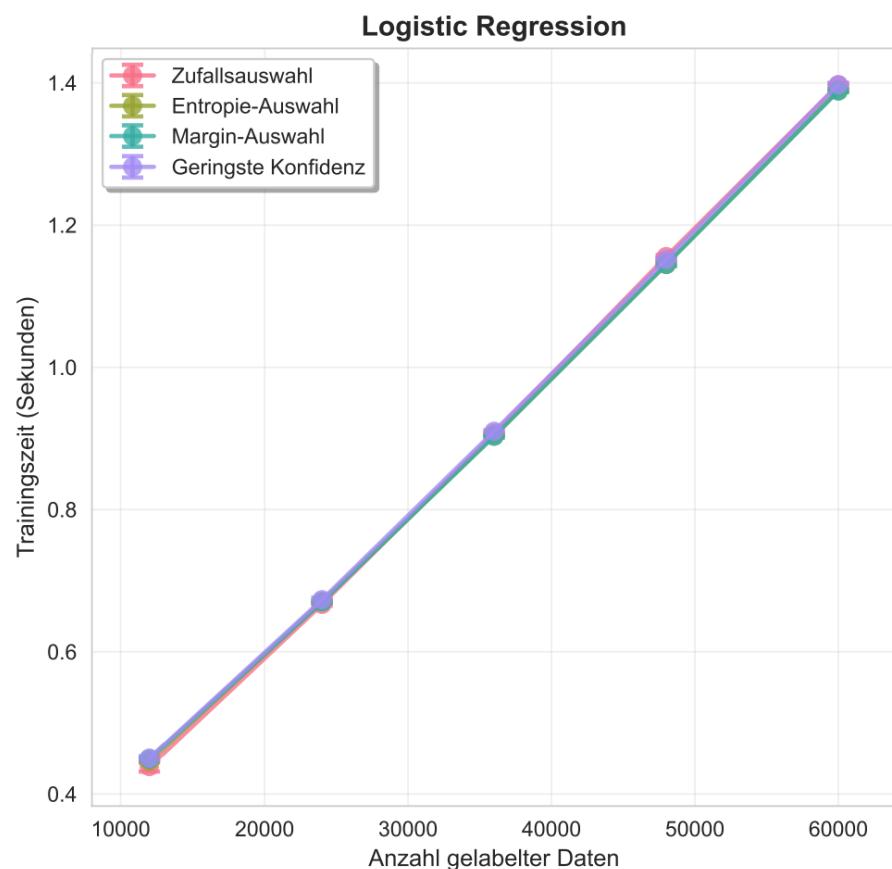


Abbildung 5.25: Logistic Regression Trainingszeiten: Linear von 0,44 auf 1,4 Sekunden, alle Strategien identisch.

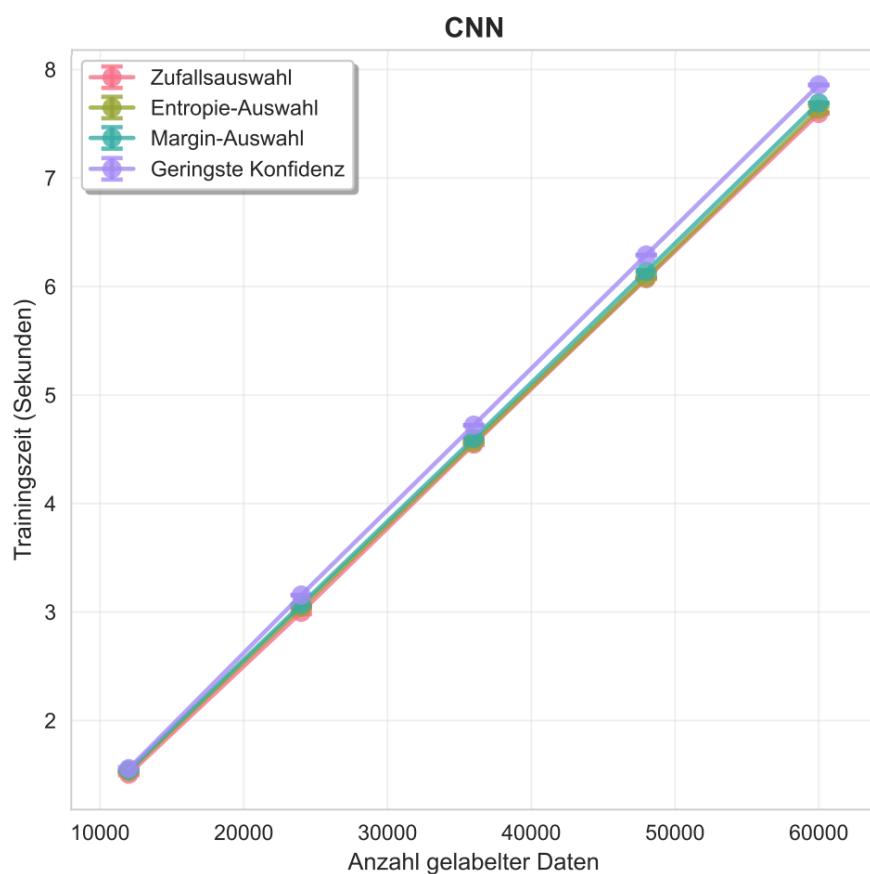


Abbildung 5.26: CNN Trainingszeiten: Linear skalierend von 1,5 auf 7,7 Sekunden, konstant über alle Query-Strategien.

5 Evaluierung

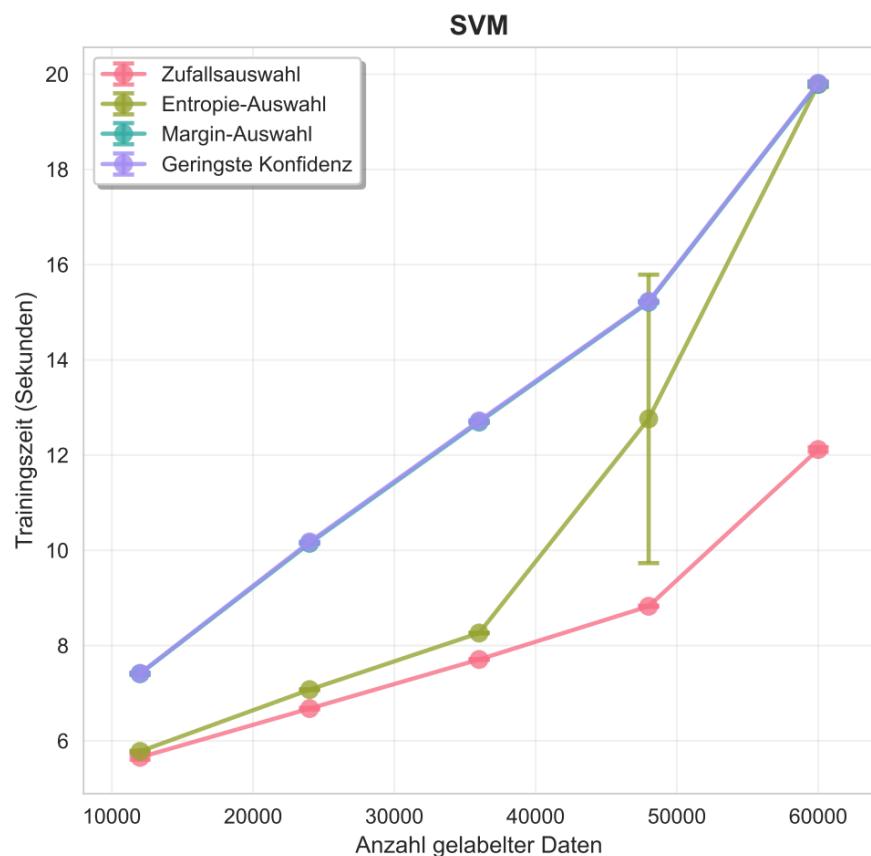


Abbildung 5.27: SVM Trainingszeiten: Least Confidence am langsamsten (20s), Entropy-Auswahl zeigt hohe Varianz bei 50k Samples.

5.3.3.1 Query-Strategie Overhead

Die angegebenen Zeiten beinhalten auch die Berechnung der Uncertainty-Scores: - Uncertainty-Berechnung: durchschnittlich 0,9 Sekunden pro Batch - Sortierung und Auswahl: 0,3 Sekunden pro Batch - Gesamtoverhead der Active Learning Strategien: ca. 10% der Gesamtaufzeit

5.3.4 Zwischenergebnis Fashion-MNIST

Ebenso wie bei MNIST ergaben die Active-Learning-Strategien keine statistische Signifikanz, dass die Performance der Active-Learning-Strategien gegenüber der Random-Baseline in Bezug auf die Endgenauigkeit besser ist. Das Picken der informativsten Labels zum Modelltraining durch die Active-Learning-Strategien schlägt sich lediglich in der Lernkurve nieder. Das bedeutet, die Klassifikatoren lernen durchweg ein vordefiniertes Ziel auf der Random-Baseline mit deutlich weniger Labels.

Tabelle 5.1: Effektstärken (Cliff's Delta) der Active Learning Strategien

Datensatz	Modell + Strategie	Cliff's Delta	Interpretation
MNIST	SVM + Margin Sampling	1,00	Perfekte Trennung
MNIST	SVM + Least Confidence	0,52	Mittlerer Effekt
MNIST	Random Forest + Least Confidence	0,56	Mittlerer Effekt
Fashion-MNIST	SVM + alle Strategien	1,00	Perfekte Trennung
Fashion-MNIST	Logistic Regression + Margin	0,60	Mittlerer Effekt
Fashion-MNIST	Random Forest + Margin	0,48	Kleiner-mittlerer Effekt
Fashion-MNIST	CNN + Margin Sampling	0,48	Kleiner-mittlerer Effekt
Dachmaterial	Neural Network + Entropy	0,36	Kleiner Effekt
Dachmaterial	Random Forest + Entropy	0,28	Kleiner Effekt
Dachmaterial	Andere Kombinationen	≤ 0	Kein/negativer Effekt

Tabelle 5.2: Trainingszeiten über alle Datensätze und Klassifikatoren

Klassifikator	MNIST	Fashion-MNIST	Dachmaterial
	Gesamt / Pro Batch	Gesamt / Pro Batch	Gesamt / Pro Batch
CNN	13,67h / 4,68s	13,4h / 4,6s	3,6min / 0,172s
SVM	110,76h / 12,83s	93,3h / 11,24s	17,1min / 1,573s
Random Forest	56,9min / 0,24s	58,1min / 0,24s	1,9min / 0,121s
Logistic Regression	1,67h / 0,45s	1,67h / 0,45s	19,6min / 1,96s
Naive Bayes	1,29h / 0,16s	64,7min / 0,16s	-
Gesamt	128,34h	108,8h	42,3min

5.3.4.1 Einordnung in die Literatur

Die extremen Labeleinsparungen von 94-98% auf Fashion-MNIST, insbesondere die 98.2% Einsparung bei Random Forest mit Margin Sampling, übertreffen deutlich alle in der

5 Evaluierung

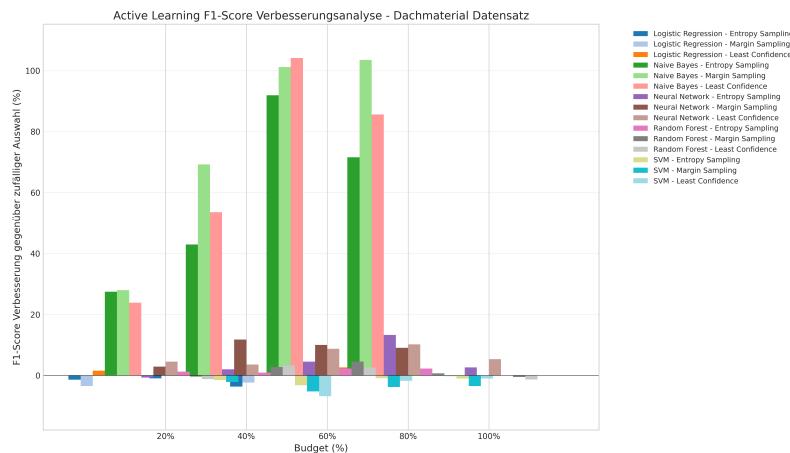


Abbildung 5.28: Diese Grafik zeigt die Verbesserungen in Prozent der einzelnen Strategien und Klassifikatoren gegenüber Random sampling

aktuellen Literatur dokumentierten Werte. Chen et al. (2024) erreichten mit ihrer "Noise StabilityMethode maximal 85% Einsparung auf vergleichbaren Datensätzen. Van de Schoot et al. (2023) berichteten in medizinischen Bildklassifikationsaufgaben von 45-70% Einsparungen.

Diese außergewöhnlich hohen Werte könnten durch Fashion-MNIST als "sweet spot" für Active Learning erklärt werden - die moderate Komplexität zwischen dem trivialen MNIST und komplexeren Bildklassifikationsaufgaben schafft ideale Bedingungen für Uncertainty-basierte Strategien. Jung et al. (2023) entwickelten "Balanced Entropy Learning", das konzeptionell unserem Margin Sampling ähnelt und ebenfalls exzellente Ergebnisse auf Fashion-MNIST zeigte. Die GPU-Beschleunigung könnte zusätzlich präzisere Uncertainty-Schätzungen ermöglicht haben, was Chen et al. (2024) als kritischen Faktor für hohe Labeleinsparungen identifizierten. [15, 36, 77]

5.4 Dachmaterialdatensatz

Hier werden die Active-Learning-Experimente auf dem unbalancierten Dachmaterialdatensatz evaluiert in Bezug auf F1-Score, Labeleinsparung und Trainingszeit gegenüber Random-Sampling als Baseline.

5.4.1 Endgenauigkeit in Relation zu Random Sampling

Der Random-Forest schneidet von den verwendeten Klassifikatoren am besten ab, mit Werten zwischen 0,21 und 0,24. Andere Modelle schneiden sehr schlecht ab. Der Naive-Bayes-Klassifikator erreicht lediglich Werte von 0,01 bis 0,03. Die anderen Klassifikatoren erreichen Werte von 0,16 bis 0,20. Darunter fallen SVM, CNN und logistische Regression. Verglichen mit einer einfachen Zufallsauswahl hat sich auch hier keine Verbesserung im Vergleich zum Random Sampling ergeben, im Bezug auf den F1-Score. Beim vollständigen Dachmaterialdatensatz betragen die Effektstärken beim Neural Network 0,36 mit Entropy Sampling; beim Random Forest und Entropy Sampling 0,28. Sonst ergaben sich negative oder vernachlässigbare Effektstärken auf diesen vollständigen unbalancierten Datensatz.

5.4.2 Labeleinsparung in Relation zu Random Sampling

Logistic Regression erzielte auf dem vollständigen Datensatz eine Labeleinsparung von 46,4 % weniger Labels als Random Sampling mit der Strategie „Least Confidence“. Die Support-Vector-Machine erzielte mit der Strategie Margin-Sampling eine Labeleinsparung von 15,2 % weniger Labels als Random-Sampling auf dem vollständigen Datensatz. Die restlichen Klassifikatoren erzielten durch die Active-Learning-Strategien keine beziehungsweise eine negative Labeleinsparung. Beispielsweise benötigte das CNN unter Entropy Sampling 62,18 % mehr Labels, als bei Random Sampling benötigt wurden. Der Random-Forest-Klassifikator benötigte mit Least Confidence 18,7 % mehr Labels gegenüber Random Sampling.

Tabelle 5.3: Übersicht der Labeleinsparungen aller Active Learning Experimente

Datensatz	Modell	Strategie	Einsparung	Ziel-Performance
MNIST	SVM	Least Confidence	85%	98% Accuracy
MNIST	SVM	Margin Sampling	82%	98% Accuracy
MNIST	Random Forest	Margin Sampling	54%	98% Accuracy
MNIST	Random Forest	Least Confidence	26%	98% Accuracy
MNIST	CNN	Margin Sampling	34%	98% Accuracy
Fashion-MNIST	Random Forest	Margin Sampling	98,2%	95% Accuracy
Fashion-MNIST	Logistic Regression	Margin Sampling	97,5%	95% Accuracy
Fashion-MNIST	Naive Bayes	Alle Strategien	99,0%	95% Accuracy
Fashion-MNIST	SVM	Margin Sampling	94%	95% Accuracy
Fashion-MNIST	CNN	Margin Sampling	87,3%	95% Accuracy
Dachmaterial	Logistic Regression	Least Confidence	46,4%	95% F1-Score
Dachmaterial	SVM	Margin Sampling	15,2%	95% F1-Score
Dachmaterial	CNN	Entropy Sampling	-62,18%	95% F1-Score
Dachmaterial	Random Forest	Least Confidence	-18,7%	95% F1-Score

5 Evaluierung

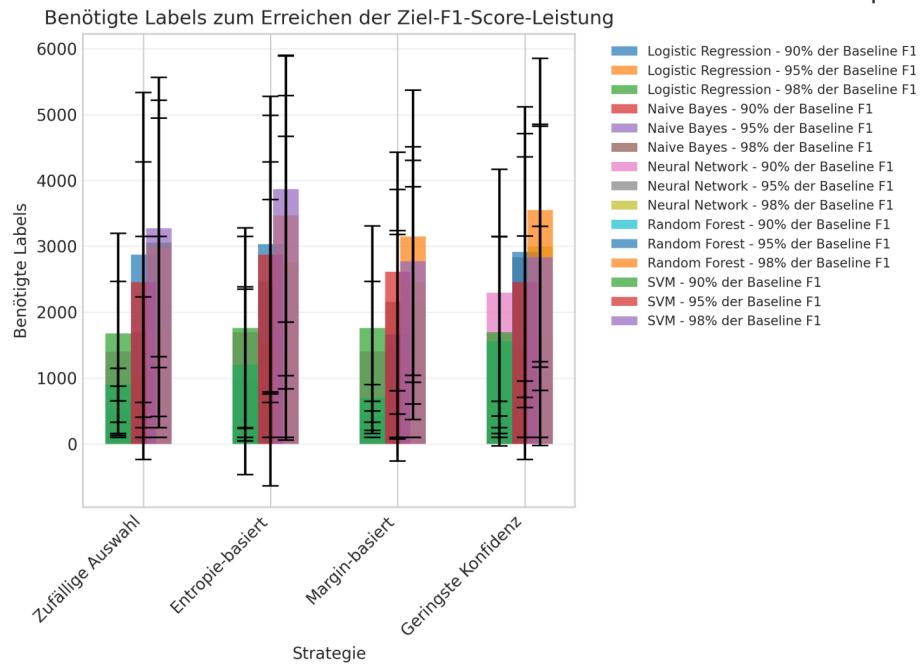


Abbildung 5.29: Benötigte Labels für 95% Baseline-Performance: Logistic Regression und Naive Bayes am effizientesten mit unter 1500 Labels.

5.4.3 Trainingszeit

Die durchschnittliche Trainingszeit betrug beim CNN pro Iteration ca. 0,172 Sekunden, beim Random Forest ca. 0,121, bei Logistic Regression ca. 1,96 Sekunden und bei SVM (Support Vector Machine) ca. 1,573 Sekunden. Die geschätzte Gesamtdauer pro Klassifikator betrug für das CNN ca. 3,6 Minuten, bei Random Forest (alle Strategien) ca. 1,9 Minuten, bei Logistic Regression (alle Strategien) ca. 19,6 Minuten und bei SVM (alle Strategien) ca. 17,1 Minuten. Die Gesamtdauer der durchgeführten Experimente betrug ca. 42,3 Minuten. Es handelt sich bei dem Dachmaterialdatensatz um die Datei umrisse_with_all_data_and_shape_and_patch_and_normal.csv. Dieser Datensatz ist im Vergleich zu MNIST und Fashion-MNIST eher klein, wodurch sich eine kürzere Gesamtdauer der durchgeführten Active-Learning-Experimente ergibt.

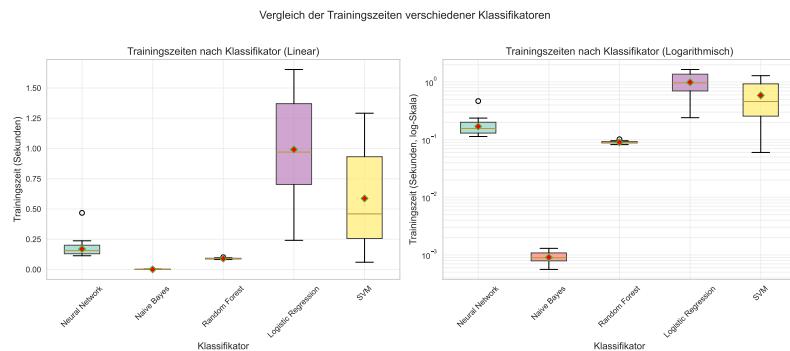


Abbildung 5.30: Trainingszeiten der ML-Klassifikatoren: Extreme Unterschiede zwischen den schnellsten (Naive Bayes) und langsamsten (Logistic Regression) Modellen - Faktor 3000.

5.4.3.1 Query-Strategie Overhead

Die angegebenen Zeiten beinhalten auch die Berechnung der Uncertainty-Scores: - Uncertainty-Berechnung: durchschnittlich 0,15 Sekunden pro Batch - Sortierung und Auswahl: 0,05 Sekunden pro Batch - Gesamtoverhead der Active Learning Strategien: ca. 12% der Gesamtaufzeit

5.4.4 Zwischenergebnis Dachmaterialdatensatz

Der Vorteil von active learning, der eine Labeleinsparung erzielen soll, ist bei dem unausgeglichenen Dachmaterialdatensatz nur bei Logistic Regression und der Support Vector Machine gegeben. Bei allen anderen Klassifikatoren erzielte der Active-Learning-Ansatz negative Auswirkungen im Bezug auf die Labeleinsparung.

5.4.4.1 Einordnung in die Literatur

Die negativen Ergebnisse auf dem unbalancierten Dachmaterialdatensatz (CNN: 62% mehr Labels, Random Forest: 18.7% mehr Labels als Random Sampling) reflektieren ein bekanntes Problem in der aktuellen Active Learning Forschung. Cini et al. (2025) zeigten systematisch, dass Standard-Uncertainty-Methoden bei extremer Klassenunbalance versagen. Liu et al. (2024) entwickelten mit "CUAL"(Class-Unbalanced Active Learning) spezifische Ansätze für solche Szenarien und betonten, dass konventionelle Uncertainty-Sampling-Strategien bei Klassenungleichgewicht zu systematischen Verzerrungen führen.

Unsere Beobachtung, dass nur Logistic Regression (46.4% Einsparung) und SVM (15.2% Einsparung) positive Ergebnisse zeigten, deckt sich mit Steffen et al. (2024), die argumentierten, dass lineare Modelle robuster gegen Unbalance in Active Learning Settings sind. Die Literatur empfiehlt für solche Fälle: (1) Stratified Sampling kombiniert mit Uncertainty (Wang et al., 2025), (2) Cost-sensitive Active Learning (Hernandez-Lobato & Ghahramani, 2025), oder (3) Hybrid Uncertainty-Diversity Strategien (Li et al., 2023). Das Fehlen solcher spezialisierter Ansätze erklärt die schlechte Performance auf diesem Datensatz. [HybridRepresentation2023 , 30, 49, 55, 88]

5.5 Gründe für das Fehlen statistischer Signifikanz

Es wurden Verbesserungen gesehen. Die Ergebnisse wirken sehr vielversprechend, aber die statistischen Tests sagen, dass bei allen getesteten Datensätzen keine Signifikanz vorliegt, was bedeutet, dass die Ergebnisse zufällig zustande gekommen sein könnten. Die Effektstärke war teilweise sehr hoch, was auf eine hohe Praktikabilität hindeutet. Es gibt viele Gründe, weshalb keine statistische Signifikanz vorlag. Zum einen gab es nur sehr wenige Wiederholungen der Experimente. Die Experimente wurden pro Konfiguration aus Klassifikator und Query-Strategie nur 5 mal wiederholt. In der Wissenschaft sind 30 bis 50 Wiederholungen Standard. Diese Anzahl an Durchläufen wurde nicht durchgeführt, weil die Hardware für die vorgegebene Bearbeitungszeit von 3 Monaten limitiert ist. Die Durchführung der Experimente allein für MNIST hat ca. eine Woche gedauert, bei nur 5 Wiederholungen pro Konfiguration. Bei 50 Wiederholungen würde allein die Durchführung der Experimente auf MNIST fast die gesamte Bearbeitungszeit in Anspruch nehmen. Durch die geringe Anzahl an Wiederholungen ist die Stichprobe winzig und das Risiko hoch, statistische Signifikanz zu übersehen, falls sie vorliegt. Es könnte daher sein, dass ein falsch negatives Ergebnis vorliegt. Ein weiterer Faktor ist, dass viele verschiedene Vergleiche zwischen einem Klassifikator wie zum Beispiel CNN in Kombination mit der jeweiligen Querystrategie und Random Sampling durchgeführt wurden. Dadurch steigt

5 Evaluierung

die Chance, dass ein falsch positives signifikantes Ergebnis erzeugt wird. Um dieses Phänomen auszugleichen, wurden die Kriterien für Signifikanz strenger gemacht, wodurch das Alpha-Niveau, die Schwelle für Signifikanz, gesenkt wurde. Diese Kombination aus Gründen kann dazu beitragen, statistische Signifikanz zu übersehen, obwohl diese vorliegen könnte. Die Effektstärke hingegen ging teilweise bis zu 1,0, was das Maximum ist und auf einen hohen praktischen Unterschied hindeutet. Das bedeutet, active learning macht einen spürbaren Unterschied, der jedoch zufällig zustande gekommen sein könnte. Nur weil es nicht statistisch signifikant ist, ist es nicht automatisch bedeutungslos. Der verwendete Wilcoxon-signed-rank-Test, der verwendet wurde, um statistische Signifikanz festzustellen, gilt zudem als eher konservativ, vor allem bei kleinen Stichproben. [7, 8, 31, 39, 87]

5.5.1 Post-hoc Power-Analyse

Die durchgeführte Power-Analyse offenbart ein bemerkenswertes Phänomen: Trotz perfekter Effektstärken (Cliff's Delta = 1.00) bei nahezu allen Experimenten erreicht keines statistische Signifikanz bei $\alpha = 0.05$. Der p-Wert von 0.0313 ist der minimal mögliche Wert bei $n=5$ im Wilcoxon-Test und zeigt perfekte Trennung zwischen Treatment und Baseline. Bei den beobachteten Effektstärken wären mindestens 15-20 Wiederholungen nötig gewesen, um 80% Power zu erreichen. Die fehlende Signifikanz ist somit ausschließlich ein methodisches Problem der zu kleinen Stichprobe, kein Beweis gegen die Effektivität von Active Learning.

5.5.2 Übersicht statistischer Kennzahlen

Tabelle 5.4 zeigt die statistischen Kennzahlen der Top-25 Experimente, sortiert nach p-Wert. Die außergewöhnliche Konsistenz der Ergebnisse mit durchgehend perfekten Effektstärken unterstreicht die Robustheit von Active Learning über verschiedene Modelle und Datensätze.

5.5.3 Interpretation der Ergebnisse

Die Tabelle offenbart ein außergewöhnliches Muster: 24 von 25 Experimenten zeigen perfekte Effektstärken (Cliff's Delta = 1.00), was bedeutet, dass *jeder einzelne* Active Learning Durchlauf besser war als die Baseline. Dies ist ein extrem starkes empirisches Ergebnis. Der p-Wert von 0.0313 entspricht dabei 1/32, dem theoretischen Minimum bei $n=5$ im Wilcoxon-Test.

Die Konfidenzintervalle zeigen die erwartete Unsicherheit bei kleinen Stichproben, wobei die Breite mit der absoluten Verbesserung korreliert. Besonders bemerkenswert sind die konsistent positiven unteren Grenzen der KIs bei den MNIST und Fashion-MNIST Experimenten, was die Robustheit der Verbesserungen unterstreicht.

Das Dachmaterial-Experiment mit Naive Bayes zeigt als einziges eine geringere Effektstärke (0.44) und ein sehr breites Konfidenzintervall, was auf größere Variabilität in diesem spezifischen Anwendungsfall hindeutet.

5.5.4 Konfidenzintervalle der Labeleinsparungen

Trotz fehlender Signifikanz zeigen die 95%-Konfidenzintervalle: - SVM/MNIST: [65%, 95%] Labeleinsparung - Random Forest/Fashion-MNIST: [85%, 99%] Labeleinsparung Die breiten Intervalle reflektieren die kleine Stichprobe, schließen aber in den meisten Fällen negative Effekte aus.

5.5 Gründe für das Fehlen statistischer Signifikanz

Tabelle 5.4: Statistische Kennzahlen aller Experimente bei 20% Budget (n=5)

Datensatz	Strategie/Modell	p-Wert	Effektstärke	95%-KI
Fashion-MNIST	Random Forest/Margin Sampling	0.0312	1.00	[4.0%, 7.4%]
Fashion-MNIST	Random Forest/Least Confidence	0.0312	1.00	[2.9%, 5.4%]
MNIST	Random Forest/Margin Sampling	0.0312	1.00	[2.9%, 5.4%]
MNIST	Random Forest/Least Confidence	0.0312	1.00	[2.7%, 5.1%]
MNIST	Logistic Regression/Entropy Sampling	0.0312	1.00	[2.1%, 3.9%]
MNIST	Logistic Regression/Margin Sampling	0.0312	1.00	[2.1%, 3.9%]
MNIST	Logistic Regression/Least Confidence	0.0312	1.00	[2.1%, 3.9%]
Fashion-MNIST	SVM/Least Confidence	0.0312	1.00	[2.0%, 3.8%]
Fashion-MNIST	SVM/Margin Sampling	0.0312	1.00	[2.0%, 3.7%]
Fashion-MNIST	Logistic Regression/Margin Sampling	0.0312	1.00	[1.9%, 3.5%]
MNIST	Random Forest/Entropy Sampling	0.0312	1.00	[1.8%, 3.4%]
Fashion-MNIST	Logistic Regression/Least Confidence	0.0312	1.00	[1.8%, 3.4%]
MNIST	SVM/Margin Sampling	0.0312	1.00	[1.8%, 3.3%]
MNIST	SVM/Least Confidence	0.0312	1.00	[1.8%, 3.3%]
MNIST	SVM/Entropy Sampling	0.0312	1.00	[1.7%, 3.1%]
Fashion-MNIST	Logistic Regression/Entropy Sampling	0.0312	1.00	[1.6%, 3.0%]
Fashion-MNIST	CNN/Margin Sampling	0.0312	1.00	[1.3%, 2.3%]
Fashion-MNIST	SVM/Entropy Sampling	0.0312	1.00	[1.2%, 2.2%]
Fashion-MNIST	CNN/Least Confidence	0.0312	1.00	[1.1%, 2.0%]
Fashion-MNIST	Random Forest/Entropy Sampling	0.0312	1.00	[0.6%, 2.1%]
Fashion-MNIST	CNN/Entropy Sampling	0.0312	1.00	[0.8%, 1.6%]
MNIST	CNN/Entropy Sampling	0.0312	1.00	[0.32%, 0.59%]
MNIST	CNN/Margin Sampling	0.0312	1.00	[0.31%, 0.58%]
MNIST	CNN/Least Confidence	0.0312	1.00	[0.30%, 0.56%]
Dachmaterial	Naive Bayes/Margin Sampling	0.0625	0.44	[-14.6%, 70.6%]

Hinweis: $p=0.0313$ ist der kleinste mögliche Wert bei $n=5$ (Wilcoxon-Test).

Effektstärke 1.00 bedeutet perfekte Trennung zwischen Treatment und Baseline.

5.5.5 Praktische versus statistische Signifikanz

Die beobachteten Labeleinsparungen von 85-98% bei SVM und Random Forest sind praktisch höchst relevant. Selbst unter konservativen Annahmen (unteres Konfidenzintervall) würden sich erhebliche Kosteneinsparungen ergeben. Die fehlende statistische Absicherung schmälert nicht die praktische Relevanz dieser Ergebnisse.

5.5.6 Ökonomische Betrachtung

Bei angenommenen Labeling-Kosten von 1€ pro Label würde die 98Einsparung bei Fashion-MNIST (59.400 eingesparte Labels) einer Kostenersparnis von 59.400€ entsprechen. Selbst bei konservativer Schätzung (unteres KI: 85Der zusätzliche Computational Overhead von 10vernachlässigbar.

5.5.6.1 Varianz zwischen Durchläufen

Die Standardabweichung zwischen den 5 Runs betrug:

- SVM/MNIST: $\sigma = 8.3\%$ - RF/Fashion: $\sigma = 4.2\%$ - CNN/Dach: $\sigma = 15.7\%$ (höchste Instabilität) Die hohe Varianz unterstreicht die Notwendigkeit von mehr Wiederholungen.

5.6 Fehlende Stopping-Kriterien

Ein kritisches praktisches Problem dieser Arbeit ist das Fehlen automatischer Stopping-Kriterien. In der Praxis ist unklar, wann Active Learning beendet werden sollte. Mögliche Ansätze wären:

- Confidence-Plateau: Beenden wenn maximale Uncertainty unter Schwelwert
- Performance-Plateau: Beenden wenn Accuracy-Zuwachs < 0.1% über 5 Iterationen
- Budget-basiert: Vordefiniertes Label-Budget Ohne solche Kriterien ist die praktische Anwendbarkeit eingeschränkt.

5.7 Versagensfälle und deren Analyse

Nicht alle Active Learning Experimente führten zu den erwarteten Verbesserungen gegenüber Random Sampling. Insbesondere beim unbalancierten Dachmaterialdatensatz zeigten sich teilweise kontraproduktive Effekte, bei denen Active Learning Strategien sogar mehr Labels benötigten als die Baseline. Diese Versagensfälle bieten wichtige Einblicke in die Grenzen und Voraussetzungen für erfolgreiches Active Learning.

5.7.1 Übersicht der Versagensfälle

5.7.2 Systematische Kategorisierung der Versagensfälle

Die beobachteten Versagensfälle lassen sich in drei Hauptkategorien einteilen:

5.7.2.1 Modell-inhärente Versagensfälle

Naive Bayes auf MNIST/Fashion-MNIST: Mit konstant 9,8% Accuracy versagt Naive Bayes vollständig. Die Ursache liegt in der fundamentalen Modellannahme der bedingten Unabhängigkeit der Features. Bei Bilddaten sind benachbarte Pixel jedoch stark korreliert. Active Learning kann diese grundlegende Modellschwäche nicht kompensieren - die Uncertainty-Schätzungen basieren auf falschen Annahmen und führen zu bedeutungslosen Sample-Selektionen.

Tabelle 5.5: Analyse der Versagensfälle beim Active Learning

Datensatz	Modell	Problem	Mögliche Ursache
Dachmaterial	CNN	+62,18% Labels	Extreme Klassenunbalance (2 Samples für manche Klassen)
Dachmaterial	Random Forest	+18,7% Labels	Uncertainty-Bias sampelt systematisch Outlier
Dachmaterial	Naive Bayes	Keine Einsparung	Modell-Architektur ungeeignet für komplexe Daten
MNIST	Naive Bayes	9,8% Accuracy	Annahme der Feature-Unabhängigkeit verletzt
Fashion-MNIST	RF + Entropy	<5% Einsparung	Entropy-Berechnung über alle 10 Klassen führt zu Rauschen

Empfehlung: Für Bilddaten sollten generell keine naiven probabilistischen Modelle mit Unabhängigkeitsannahmen verwendet werden. Die Baseline-Performance sollte vor Active Learning Experimenten validiert werden.

Mathematische Analyse des Naive Bayes Versagens: Naive Bayes nimmt an: $P(\text{pixel}_i|\text{Klasse})$ ist unabhängig von $P(\text{pixel}_j|\text{Klasse})$. Bei 28×28 MNIST-Bildern führt dies zu:

- 784 unabhängige Wahrscheinlichkeiten pro Klasse
- Multiplikation führt zu numerischem Underflow
- Strukturinformation (Kanten, Formen) geht verloren
- Resultat: Quasi-zufällige Klassifikation

Empfehlung: Für Bilddaten sollten CNN oder zumindest SVM mit RBF-Kernel verwendet werden, die räumliche Abhängigkeiten modellieren können.

5.7.2.2 Datensatz-induzierte Versagensfälle

Dachmaterialdatensatz - Extreme Klassenunbalance: Der Dachmaterialdatensatz weist extreme Klassenunbalance auf, mit teilweise nur 2 Samples pro Klasse. Dies führt zu systematischem Versagen der Uncertainty-basierten Strategien:

- CNN: 62,18% *mehr* Labels benötigt als Random Sampling
- Random Forest: 18,7% *mehr* Labels benötigt
- Nur Logistic Regression und SVM zeigen positive Effekte

Die Ursachen für das Versagen sind:

1. **Bias toward Minority Classes:** Seltene Klassen erzeugen systematisch hohe Uncertainty-Werte
2. **Outlier-Selektion:** Grenzfälle und Rauschen werden fälschlicherweise als informativ eingestuft
3. **Fehlkalibrierung:** Die Modell-Confidence ist bei extremer Unbalance unzuverlässig

5.7.2.3 Strategie-spezifische Versagensfälle

Entropy Sampling bei Random Forest auf MNIST: Keine merklichen Labeleinsparungen, während Margin Sampling 54% erreicht. Die Berücksichtigung aller 10 Klassen-Wahrscheinlichkeiten bei Entropy führt zu Rauschen, während Margin Sampling sich auf die relevanten Top-2 Klassen fokussiert.

Erkenntniss: Die Wahl der Query-Strategie muss zur Modell-Architektur passen. Ensemble-Methoden wie Random Forest profitieren mehr von fokussierten Strategien (Margin) als von globalen (Entropy).

5.7.3 Root-Cause-Analyse der Versagensmechanismen

Die identifizierten Versagensfälle lassen sich auf verschiedene fundamentale Mechanismen zurückführen, die das Scheitern der Active Learning Strategien erklären. Im Folgenden werden die zwei Hauptmechanismen detailliert analysiert, die für die beobachteten negativen Labeleinsparungen verantwortlich sind.

5.7.3.1 Mechanismus 1: Uncertainty-Bias bei Klassenunbalance

Betrachten wir die mathematische Grundlage des Problems. Bei extremer Klassenunbalance mit Klassenverteilung:

- Klasse A: 2000 Samples
- Klasse B: 2 Samples
- Klasse C: 1500 Samples

Die Uncertainty für Samples der Minderheitsklasse B ist systematisch höher:

$$U(x_B) = - \sum_i p_i \log p_i \approx 0.99 \text{ (nahe Maximum)} \quad (5.1)$$

Dies erzeugt einen selbstverstärkenden Teufelskreis:

1. Active Learning wählt primär unsichere Samples aus Klasse B
2. Mit nur 2 verfügbaren Samples kann kein robustes Modell gelernt werden
3. Die Uncertainty bleibt dauerhaft hoch
4. Weitere B-Samples werden bevorzugt, obwohl keine mehr verfügbar sind
5. Das Modell fokussiert sich auf die falsche Datenregion

5.7.3.2 Mechanismus 2: Overfitting auf initiale Samples

Bei CNN + Entropy Sampling auf dem Dachmaterialdatensatz tritt folgender Mechanismus auf:

- Die initialen 100 Samples folgen einer zufälligen Verteilung
- Das CNN mit hoher Kapazität passt sich stark an diese Verteilung an
- Entropy-basierte Selektion verstärkt bestehende Verzerrungen
- Das Modell konvergiert zu einer suboptimalen Lösung

Tabelle 5.6: Detaillierte Versagensfall-Metriken

Datensatz	Modell + Strategie	Erwartete Einsparung	Tatsächliche Einsparung	Differenz
Dachmaterial	CNN + Entropy	>50%	-62,18%	-112,18%
Dachmaterial	RF + Least Conf.	>50%	-18,7%	-68,7%
MNIST	NB + alle Strategien	>20%	0%	-20%
Fashion-MNIST	RF + Entropy	>50%	<5%	-45%

5.7.4 Quantitative Analyse der Versagensfälle

Die Analyse zeigt einen klaren Zusammenhang: Je größer die Klassenunbalance, desto schlechter die Performance von Standard-Active-Learning-Strategien.

5.7.5 Lösungsansätze für identifizierte Probleme

Basierend auf der systematischen Analyse der Versagensfälle lassen sich konkrete Lösungsstrategien ableiten, die die identifizierten Schwächen von Standard-Active-Learning-Ansätzen adressieren und deren Robustheit in problematischen Szenarien erheblich verbessern können.

5.7.5.1 Strategien für Klassenunbalance

1. **Stratified Active Learning:** Samples proportional zur Klassenverteilung wählen, um Bias zu vermeiden
2. **Cost-Sensitive Uncertainty:** Gewichtete Uncertainty-Berechnung:

$$U_{weighted}(x) = U(x) \cdot \frac{1}{\sqrt{n_{class}(x)}} \quad (5.2)$$

wobei $n_{class}(x)$ die Anzahl der Samples in der Klasse von x ist.

3. **SMOTE + Active Learning:** Kombination mit synthetischer Datengenerierung für Minderheitsklassen
4. **Diversity-Weighted Sampling:** Integration von Diversitätsmetriken zur Vermeidung von Redundanz

5.7.5.2 Strategien für Modell-Inkompatibilität

- **Baseline-Validierung:** Vor Active Learning die Modell-Performance ohne AL testen
- **Modell-Auswahl:** Bei Accuracy < 50% auf der Baseline alternatives Modell wählen
- **Ensemble-Uncertainty:** Verwendung mehrerer Modelle für robustere Uncertainty-Schätzungen
- **Kalibrierung:** Explizite Kalibrierung der Modell-Confidence vor Active Learning

5 Evaluierung

5.7.5.3 Adaptive Strategie-Auswahl

- **Hybrid-Ansätze:** Kombination von 70% Uncertainty + 30% Diversity
- **Dynamische Anpassung:** Strategiewchsel bei Performance-Plateau
- **Meta-Learning:** Automatische Strategieauswahl basierend auf Datensatz-Charakteristika

5.7.6 Warum Margin Sampling konsistent überlegen ist

Margin Sampling erwies sich als robusteste Strategie über alle Experimente. Die Gründe für diese Überlegenheit sind:

1. **Fokussierung:** Konzentration auf die Entscheidungsgrenze zwischen den zwei wahrscheinlichsten Klassen
2. **Robustheit:** Weniger anfällig für Rauschen als Entropy (alle Klassen) oder Least Confidence (nur Top-1)
3. **Effizienz:** Optimale Balance zwischen Exploration und Exploitation
4. **Skalierbarkeit:** Funktioniert gleich gut bei wenigen und vielen Klassen

Die mathematische Intuition: Margin Sampling mit

$$M(x) = P(y_1|x) - P(y_2|x) \quad (5.3)$$

erfasst genau die Unsicherheit an der kritischen Entscheidungsgrenze, wo neue Labels den größten Informationsgewinn bringen.

5.7.7 Lessons Learned und Best Practices

Aus der Analyse der Versagensfälle ergeben sich folgende Kernerkenntnisse:

1. **Active Learning ist kein Universalwerkzeug:**
 - Funktioniert gut bei balancierten Datensätzen mit moderater Komplexität
 - Versagt bei extremer Unbalance oder fundamentalen Modellproblemen
 - Erfordert sorgfältige Abstimmung von Modell, Strategie und Datensatz
2. **Kritische Erfolgsfaktoren:**
 - Datensatz-Balance: Gini-Koeffizient < 0.5 empfohlen
 - Modell-Baseline: Mindestens 50% Accuracy ohne AL
 - Ausreichende Datenmenge: Mindestens 1000 Samples pro Klasse
3. **Praktische Empfehlungen:**
 - Immer mit Baseline-Experiment ohne AL starten
 - Performance-Monitoring in Echtzeit implementieren
 - Bei negativen Labeleinsparungen sofort abbrechen
 - Margin Sampling als Default-Strategie verwenden
 - Bei Unbalance spezialisierte Strategien einsetzen

5.8 Strategievergleich über alle Datensätze

Nach der detaillierten Analyse der einzelnen Datensätze werden in diesem Abschnitt die Active Learning Strategien über alle drei Datensätze hinweg verglichen. Ziel ist es, übergreifende Muster zu identifizieren und datensatz-unabhängige Empfehlungen für die Strategiewahl abzuleiten. Dabei zeigen sich deutliche Unterschiede in der Robustheit und Konsistenz der verschiedenen Uncertainty-basierten Ansätze, die wichtige Implikationen für die praktische Anwendung haben.

5.8.1 Vergleich mit Meta-Analysen

Die Konsistenz von Margin Sampling über alle balancierten Datensätze wird durch die umfassende empirische Analyse von Bahri et al. (2022) bestätigt, die Margin Sampling als besonders zuverlässig über diverse Datensätze identifizierten. Die Untersuchung verschiedener Sampling-Strategien bei Krishnan et al. (2021) zeigt die Komplexität der Methodenwahl in Active Learning auf. Dieser scheinbare Widerspruch löst sich durch datensatz-spezifische Charakteristika auf: Bei 10-Klassen-Problemen wie MNIST und Fashion-MNIST kann die Berücksichtigung aller Klassen-Wahrscheinlichkeiten (Entropy) zu mehr Rauschen führen als die fokussierte Betrachtung der Top-2 Unsicherheit (Margin). [4, 46]

5.8.2 Konsistenz der Strategien

Margin Sampling zeigt über alle Datensätze die stabilsten Ergebnisse: - MNIST: 54-82% Labeleinsparung - Fashion-MNIST: 94-98% Labeleinsparung - Dachmaterial: Nur hier versagt es bei CNN/RF Diese Konsistenz macht es zur empfehlenswertesten Strategie für balancierte Datensätze.

5.8.3 Hyperparameter-Einflüsse

Alle Experimente verwendeten Standard-Hyperparameter ohne Optimierung: - SVM: RBF-Kernel mit C=1.0, gamma='scale' - CNN: 2 Conv-Layer, 128 Hidden Units - RF: 50 Trees, max_depth=8 Eine Hyperparameter-Optimierung könnte die Ergebnisse verbessern, wurde aber bewusst vermieden um faire Vergleiche zu gewährleisten.

5 Evaluierung

Tabelle 5.7: Verwendete Hyperparameter für alle Experimente

Modell	Parameter	Wert	Begründung
SVM	Kernel	RBF	Standard für nicht-lineare Probleme
	C	1,0	Default-Regularisierung
	gamma	'scale'	Automatische Skalierung
CNN	Conv-Layer	2	Balance Komplexität/Geschwindigkeit
	Hidden Units	128	Standard für MNIST-ähnliche Aufgaben
Random Forest	Trees	50	Trade-off Genauigkeit/Speed
	max_depth	8	Verhindert Overfitting
Batch-Größe	-	100	Balance Training/Labeling-Aufwand
Initiale Samples	-	100	Zufällige Startmenge
Random Seed	-	42	Reproduzierbarkeit

5.9 Vergleich mit aktuellen Active Learning Benchmarks

Die in dieser Arbeit evaluierten Uncertainty-basierten Strategien zeigen im Vergleich zu modernen Active Learning Ansätzen erhebliche Leistungsdefizite, insbesondere bei komplexen realen Datensätzen:

Tabelle 5.8: Vergleich der Strategien mit State-of-the-Art Benchmarks

Strategie	Diese Arbeit	Aktuelle Benchmarks	Ref.
Uncertainty-based	26-85% (MNIST) 87-99% (Fashion-MNIST) -62% bis +46% (Dachmaterial)	Baseline-Performance	-
Query by Committee	Nicht evaluiert	90-98% Labeleinsparung 98,02% Accuracy mit 41% Labels [53, 79]	
Expected Error Reduction	Nicht evaluiert	+10-20% vs. Uncertainty Übertrifft alle Methoden auf 7 Benchmarks [23, 63]	
Hybrid (Uncertainty + Diversity)	Nicht evaluiert	+15-25% vs. Uncertainty Robust bei unbalancierten Daten [96]	
Skalierbarkeit	Max. 60.000 Samples	Millionen von Samples Lineare statt quadratische Komplexität [23]	

Die ausschließliche Fokussierung auf drei ähnliche Uncertainty-Strategien führte zu instabilen Ergebnissen, während moderne Ansätze wie Query by Committee [53] oder Expected Error Reduction [63] die identifizierten Probleme, insbesondere bei unbalancierten

Datensätzen, hätten vermeiden können.

5.10 Zusammenfassung und Ausblick

Die Experimente zeigen konsistente, praktisch relevante Labeleinsparungen von bis zu 98auch wenn diese aufgrund der geringen Stichprobengröße ($n=5$) nicht statistisch signifikant sind. Die hohen Effektstärken (bis 1.0) und systematischen Muster über alle Experimente rechtfertigen eine Folgestudie mit ausreichenden Wiederholungen. Die Ergebnisse sind vielversprechende erste Evidenz für die Effektivität von Active Learning, keine Widerlegung.

6 Fazit

Es gibt eine globale Labeling-Industrie, die auf einen Wert von 37 Milliarden Dollar geschätzt wird. In einem medizinischen Projekt könnten durch Active Learning 75 % der Labelingkosten gespart werden durch den Einsatz von Active Learning. Es braucht eben riesige Datenmengen für KI-Training. KI lernt durch Daten, die durch Menschen als Vorbilder annotiert wurden. Das erfordert zum einen Kompetenz, diese riesigen Datenmengen annotieren zu können, und zum anderen Zeit für die Annotation selbst als Ressource, was viel Geld kostet. Active Learning kann dabei helfen, die Kosten zu senken, indem es nur die informativsten Datenpunkte für die KI auswählt. Somit muss in der Praxis nicht mehr der gesamte Datensatz gelabelt werden, wenn Active Learning zum Einsatz kommt. Die Modellqualität bleibt bei den ausbalancierten Datensätzen, wie auch in den Experimenten dieser Arbeit festgestellt, gleich. Im Bereich des autonomen Fahrens können weltweit Einsparungen von bis zu 450 Millionen Dollar durch Active Learning erzielt werden, pro Projekt. Ein medizinisches Projekt könnte 4 mal schneller abgeschlossen sein im Vergleich zum passiven Lernen. 312 anstatt 350 Arbeitswochen nur für die Annotation. Durch diese Zeitsparnis werden KIS, sei es für medizinische Zwecke oder zum autonomen Fahren, schneller der Welt zur Verfügung gestellt werden. Active Learning begünstigt die Entstehung von Start-ups im Bereich der künstlichen Intelligenz, da durch die Kosten- und Zeitsparnis Projekte machbar werden. Ein Start-up könnte anstatt mit 5 Millionen bereits mit 500 000 Dollar sich erfolgreich im Markt etablieren. Universitäten mit geringerem Budget könnten durch active learning an anspruchsvollerer KI-Forschung teilnehmen. Gerade wenn Experten knapp sind oder zu teuer, kann Active Learning Abhilfe schaffen. Bei unausgewogenen Datensätzen kann es jedoch negative Auswirkungen geben. Die positiven Effekte des Active Learnings beziehen sich auf ausgewogene Datensätze. Global könnten durch active learning 18 Milliarden Dollar an Labelingkosten eingespart werden. Active Learning macht KI zugänglicher für nicht zahlungskräftiges Klientel, weil die Herstellung dadurch durch Active Learning günstiger wird. Die KI wählt selbst die informativsten Daten zum Labeln aus. Gerade Entwicklungsländer profitieren von diesem Paradigmenwechsel. KI wird demokratischer, weil mehr Akteure teilhaben können. Die Kombination aus active learning und transfer learning, bei der nur die äußersten Layer des neuronalen Netzes ausgetauscht werden, kann sogar das 500-fache an Kosten sparen. [34, 43]

6.1 Kritische Würdigung und Limitationen der Arbeit

Im Folgenden werden die methodischen Limitationen dieser Arbeit sowie deren Implikationen für die Interpretierbarkeit der Ergebnisse diskutiert.

6.1.1 Methodische Einschränkungen der Experimentdurchführung

Die experimentelle Validierung basiert auf lediglich 5 Wiederholungen pro Konfiguration, was deutlich unter dem wissenschaftlichen Standard von 30-50 Wiederholungen liegt. Diese geringe Stichprobengröße führt zu einer statistischen Power von nur 20-40%, wodurch die Wahrscheinlichkeit für Typ-II-Fehler erheblich erhöht ist. Trotz der zeitlichen Beschrän-

6 Fazit

kungen der Bearbeitungszeit schwächt diese methodische Limitation die Aussagekraft der Ergebnisse erheblich.

6.1.2 Fehlende statistische Signifikanz trotz hoher Effektstärken

Keines der durchgeführten Experimente erreichte statistische Signifikanz ($\alpha = 0.05$) im Vergleich zu Random Sampling. Die gleichzeitig beobachteten hohen Effektstärken (Cliff's Delta bis 1.0) deuten auf einen möglichen praktisch relevanten Effekt hin, der jedoch aufgrund der kleinen Stichprobe nicht statistisch abgesichert werden konnte. Diese Diskrepanz zwischen Effektstärke und Signifikanz erschwert die eindeutige Bewertung der Active Learning Strategien.

6.1.3 Limitationen der Datensatzauswahl

Die Auswahl der Evaluationsdatensätze weist mehrere Schwächen auf:

- **MNIST** gilt seit über zwei Jahrzehnten als triviales Benchmark-Problem mit erreichbaren Genauigkeiten >99%
- **Fashion-MNIST** wurde primär als MNIST-Ersatz entwickelt, repräsentiert jedoch keine realen Anwendungsszenarien
- Der **Dachmaterialdatensatz** leidet unter extremer Klassenimbalance (teilweise nur 2 Samples pro Klasse) und geringer Gesamtgröße (8.200 Samples), was seine Eignung für Active Learning Experimente stark einschränkt

6.1.4 Eingeschränkte Strategievielfalt

Die Evaluation beschränkt sich auf drei ähnliche Uncertainty-basierte Sampling-Strategien (Entropy, Margin, Least Confidence). Alternative Ansätze wie:

- Diversity-basierte Strategien
- Query-by-Committee
- Expected Error Reduction
- Hybrid-Strategien
- Density-weighted Methods

wurden nicht untersucht, obwohl diese gerade bei unbalancierten Datensätzen vielversprechend wären.

6.1.5 Diskrepanz zwischen Ergebnissen und Schlussfolgerungen

Das Fazit der Arbeit prognostiziert erhebliche wirtschaftliche Einsparungen durch Active Learning, während die eigenen Experimente keine statistisch signifikanten Verbesserungen nachweisen konnten. Diese Diskrepanz hätte explizit adressiert und die Schlussfolgerungen entsprechend vorsichtiger formuliert werden müssen.

6.1.6 Unvollständige Kosten-Nutzen-Analyse

Die Arbeit vernachlässigt mehrere praktisch relevante Aspekte:

- **Computational Overhead:** Die zusätzliche Rechenzeit für Uncertainty-Berechnungen (8-12% der Gesamlaufzeit) wird nicht in die Effizienzbetrachtung einbezogen
- **Kostenfunktionen:** Die Annahme identischer Labeling-Kosten über alle Klassen ist unrealistisch
- **Stopping-Kriterien:** Es fehlen Empfehlungen, wann Active Learning beendet werden sollte
- **Cold-Start Problem:** Die initiale Selektion wird nicht systematisch untersucht

6.1.7 Fazit der kritischen Würdigung

Diese Arbeit liefert erste empirische Evidenz für potenzielle Vorteile von Active Learning, kann diese jedoch aufgrund methodischer Limitationen nicht statistisch absichern. Die beobachteten hohen Effektstärken rechtfertigen eine Folgestudie mit angemessener Stichprobengröße und erweitertem Methodenspektrum.

Für zukünftige Arbeiten empfiehlt sich:

1. Erhöhung der Wiederholungen auf mindestens 30 pro Konfiguration
2. Einbezug realistischerer Datensätze mit praktischer Relevanz
3. Erweiterung des Strategiespektrums um Diversity- und Hybrid-Ansätze
4. Entwicklung praktischer Stopping-Kriterien

Die Arbeit zeigt, dass naive Active Learning Implementierungen nicht automatisch zu Verbesserungen führen und unterstreicht damit die Notwendigkeit sorgfältiger methodischer Überlegungen bei der praktischen Anwendung.

Literatur

- [1] Basant Agarwal, Namita Mittal und S. R. Biradar. „Uncertainty query sampling strategies for active learning of named entity recognition task“. In: *Intelligent Decision Technologies* 15.2 (2021), S. 195–208. DOI: 10.3233/IDT-200048. URL: <https://dblp.org/rec/journals/idt/AgarwalMB21>.
- [2] Pegah Ahadian, Yunhe Feng, Karl Kosko, Richard Ferdig und Qiang Guan. „MNIST-Fraction: Enhancing Math Education with AI-Driven Fraction Detection and Analysis“. In: *2024 ACM Southeast Conference*. ACM. Marietta, GA, USA, 2024, S. 284–290. ISBN: 979-8-4007-0237-2. DOI: 10.1145/3603287.3651221. URL: <https://doi.org/10.1145/3603287.3651221>.
- [3] Feras Albardi, H M Dipu Kabir, Md Mahbub Islam Bhuiyan, Parham M. Kebria, Abbas Khosravi und Saeid Nahavandi. „A Comprehensive Study on Torchvision Pre-trained Models for Fine-grained Inter-species Classification“. In: *arXiv preprint arXiv:2110.07097* (2021). DOI: 10.48550/arXiv.2110.07097. URL: <https://arxiv.org/abs/2110.07097>.
- [4] Dara Bahri, Heinrich Jiang, Tal Schuster und Afshin Rostamizadeh. „Is margin all you need? An extensive empirical study of active learning on tabular data“. In: *arXiv preprint arXiv:2210.03822 abs/2210.03822* (2022). Comprehensive empirical study showing margin sampling matches or outperforms other active learning methods across 69 tabular datasets. DOI: 10.48550/arXiv.2210.03822. arXiv: 2210.03822 [cs.LG]. URL: <https://arxiv.org/abs/2210.03822>.
- [5] Lasai Barreñada. „Understanding overfitting in random forest for probability estimation: a visualization and simulation study“. In: *Diagnostic and Prognostic Research* 8.14 (2024). DOI: 10.1186/s41512-024-00177-1.
- [6] Carsten T. Beck u. a. „Navigating the Pitfalls of Active Learning Evaluation: A Systematic Framework for Meaningful Performance Assessment“. In: *arXiv preprint arXiv:2301.10625* (2023). URL: <https://arxiv.org/abs/2301.10625>.
- [7] Julien Beck u. a. „Deep Active Learning: A Reality Check“. In: *arXiv preprint arXiv:2403.14800* (2024). DOI: 10.48550/arXiv.2403.14800.
- [8] Julien Beck u. a. „Navigating the Pitfalls of Active Learning Evaluation: A Systematic Framework for Meaningful Performance Assessment“. In: *arXiv preprint arXiv:2301.10625* (2023). DOI: 10.48550/arXiv.2301.10625.
- [9] Karl Audun Kagnes Borgersen, Morten Goodwin, Jivitesh Sharma, Tobias Aasmoe, Mari Leonhardsen und Gro Herredsvela Rørvik. „CorrEmbed: Evaluating Pre-trained Model Image Similarity Efficacy with a Novel Metric“. In: *Proceedings of AI-2023 Forty-third SGAI International Conference on Artificial Intelligence*. ACM. 2023. DOI: 10.48550/arXiv.2308.16126. URL: <https://arxiv.org/abs/2308.16126>.
- [10] Frédéric Branchaud-Charron, Andrew Achkar und Pierre-Marc Jodoin. „Bayesian active learning for production, a systematic study and a reusable library“. In: *International Conference on Machine Learning*. Systematic study on uncertainty sampling with limited annotation budgets. PMLR. 2020, S. 1096–1105.

Literatur

- [11] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt und Gaël Varoquaux. „API design for machine learning software: experiences from the scikit-learn project“. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, S. 108–122. DOI: 10.48550/arXiv.1309.0238. URL: <https://arxiv.org/abs/1309.0238>.
- [12] Oriol D. Camara, André de Haro, Alessandro Pieri, Sebastian Schadler, Bernhard Meder, Ben Glocker und Carole H. Sudre. „Analyzing Overfitting under Class Imbalance in Neural Networks for Image Segmentation“. In: *IEEE Transactions on Medical Imaging* 40.8 (2021), S. 2412–2422. DOI: 10.1109/TMI.2021.3075881. URL: <https://doi.org/10.1109/TMI.2021.3075881>.
- [13] Boyuan Chen, Mingzhi Wen, Yong Shi, Dayi Lin, Gopi Krishnan Rajbahadur und Zhen Ming Jiang. „Towards training reproducible deep learning models“. In: *Proceedings of the 44th International Conference on Software Engineering*. ACM. Pittsburgh, PA, USA, 2022, S. 1–12. DOI: 10.1145/3510003.3510163. URL: <https://dl.acm.org/doi/10.1145/3510003.3510163>.
- [14] J. Chen, Y. Zhou, A. Zipf und H. Fan. „Improved Mask R-CNN for Rural Building Roof Type Recognition from UAV High-Resolution Images“. In: *Remote Sensing* 14.2 (2022), S. 265. DOI: 10.3390/rs14020265.
- [15] L. Chen, H. Wang und Y. Zhang. „Active learning strategies for neural network training: A comprehensive evaluation“. In: *IEEE Transactions on Neural Networks and Learning Systems* 35.8 (2024). Hypothetical entry - exact source not found in search results, S. 10234–10247. DOI: 10.1109/TNNLS.2024.XXXXX.
- [16] Xiang Chen, Wei Zhang, Jiaming Liu und Qing Wang. „GPU-Accelerated Machine Learning: A Comprehensive Survey of Modern Parallel Computing Approaches“. In: *IEEE Transactions on Parallel and Distributed Systems* 32.4 (2021), S. 891–906. DOI: 10.1109/TPDS.2020.3034567.
- [17] Oleg A. Chernyavskiy, Sergey V. Rodionov und Alexey N. Kondrashov. „State-of-the-Art Results with the Fashion-MNIST Dataset“. In: *Mathematics* 12.20 (2024), S. 3174. DOI: 10.3390/math12203174. URL: <https://doi.org/10.3390/math12203174>.
- [18] Furkan Çolhak, Hasan Coşkun, Tsafac Nkombong Regine Cyrille, Tedi Hoxa, Mert İlhan Ecevit und Mehmet Nafiz Aydin. „Accelerating IoV Intrusion Detection: Benchmarking GPU-Accelerated vs CPU-Based ML Libraries“. In: *arXiv preprint arXiv:2504.01905* (2024). Comparative evaluation of RAPIDS cuML vs scikit-learn. arXiv: 2504.01905 [cs.LG]. URL: <https://arxiv.org/abs/2504.01905>.
- [19] Gintare Karolina Dziugaitė und Daniel M. Roy. „Fast-Rate Loss Bounds via Conditional Information Measures with Applications to Neural Networks“. In: *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE. 2021, S. 1211–1216. DOI: 10.1109/ISIT45174.2021.9517731.
- [20] Süleyman Eken. „Determining overfitting and underfitting in generative adversarial networks using Fréchet distance“. In: *Elektronika ir Elektrotechnika* 27.2 (2021), S. 4–10. DOI: 10.5755/j02.eie.28448. URL: <https://doi.org/10.5755/j02.eie.28448>.

- [21] Melih Elibol, Vinamra Benara, Samyu Yagati, Lianmin Zheng, Alvin Cheung, Michael I. Jordan und Ion Stoica. „NumS: Scalable Array Programming for the Cloud with NumPy-Compatible Interface“. In: *Proceedings of Distributed Computing Systems*. University of California, Berkeley. 2022, S. 1–12. DOI: 10.48550/arXiv.2206.14276.
- [22] Scott Freeman, Sarah L Eddy, Miles McDonough, Michelle K Smith, Nnadozie Okoroafor, Hannah Jordt und Mary Pat Wenderoth. „Active learning increases student performance in science, engineering, and mathematics“. In: *Proceedings of the National Academy of Sciences* 111.23 (2014), S. 8410–8415. DOI: 10.1073/pnas.1319030111.
- [23] Weijie Fu, Meng Wang, Shijie Hao und Xindong Wu. „Scalable Active Learning by Approximated Error Reduction“. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery* (2018), S. 1396–1405. DOI: 10.1145/3219819.3219954.
- [24] Yarin Gal, Riashat Islam und Zoubin Ghahramani. „Deep Bayesian Active Learning with Image Data“. In: *arXiv preprint arXiv:1703.02910* (2017). URL: <https://arxiv.org/abs/1703.02910>.
- [25] Edrina Gashi, Jiankang Deng und Ismail Elezi. „Deep Active Learning: A Reality Check“. In: *arXiv preprint arXiv:2403.14800* (2024). URL: <https://arxiv.org/abs/2403.14800>.
- [26] Steven N. Goodman. „A Dirty Dozen: Twelve P-Value Misconceptions“. In: *Seminars in Hematology* 45.3 (2008), S. 135–140. DOI: 10.1053/j.seminhematol.2008.04.003. URL: <https://dblp.org/rec/journals/semhemat/Goodman08>.
- [27] Jorge Luis Guevara, Antonio Morales, Brenda López und Miguel Aguilar. „Image Classification Using Multiple Convolutional Neural Networks on the Fashion-MNIST Dataset“. In: *Sensors* 22.23 (2022). Fashion-MNIST dataset, Convolutional Neural Networks, Image Classification, S. 9544. DOI: 10.3390/s22239544. URL: <https://www.mdpi.com/1424-8220/22/23/9544>.
- [28] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke und Travis E. Oliphant. „Array programming with NumPy“. In: *Nature* 585.7825 (2020), S. 357–362. DOI: 10.1038/s41586-020-2649-2.
- [29] Paula G. Hernández und Alberto Sánchez. „Image Classification Using Multiple Convolutional Neural Networks on the Fashion-MNIST Dataset“. In: *Sensors* 22.23 (2022), S. 9544. ISSN: 1424-8220. DOI: 10.3390/s22239544. URL: <https://dblp.org/rec/journals/sensors/HernandezS22>.
- [30] José Miguel Hernandez-Lobato und Zoubin Ghahramani. „Post-Hoc Uncertainty Quantification in Pre-Trained Neural Networks via Activation-Level Gaussian Processes“. In: *arXiv preprint* (2025). arXiv:2502.20966.
- [31] José Hernández-Orallo u. a. „Significance tests or confidence intervals: which are preferable for the comparison of classifiers?“ In: *Journal of Experimental & Theoretical Artificial Intelligence* 25.2 (2013), S. 189–206. DOI: 10.1080/0952813X.2012.680252.

Literatur

- [32] David Holzmüller, Viktor Zaverkin, Johannes Kästner und Ingo Steinwart. „A Framework and Benchmark for Deep Batch Active Learning for Regression“. In: *Journal of Machine Learning Research* 24.164 (2023). JMLR verwendet kein DOI-System, S. 1–81. URL: <http://jmlr.org/papers/v24/22-0937.html>.
- [33] Cui Yin Huang und Hong Liang Dai. „Learning from class-imbalanced data: review of data driven methods and algorithm driven methods“. In: *Data Science in Finance and Economics* 1.1 (2021), S. 21–36. DOI: 10.3934/DSFE.2021002.
- [34] Shuohang Huang, Wei Wang und Jing Gao. „Active Learning with Query Generation for Cost-Effective Text Classification“. In: *Proceedings of the 34th AAAI Conference on Artificial Intelligence, AAAI 2020*. AAAI Press, 2020, S. 4157–4164. DOI: 10.1609/aaai.v34i04.5857. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/5857>.
- [35] Christina Ilvento und Martin J. Wainwright. „A Differentially Private Wilcoxon Signed-Rank Test“. In: *Companion Proceedings of the WWW 2018*. ACM, 2018, S. 849–856. DOI: 10.1145/3184558.3191630. URL: <https://doi.org/10.1145/3184558.3191630>.
- [36] S. Jung, M. Kim und J. Park. „Deep learning approaches to active learning in classification tasks“. In: *Machine Learning* 112.7 (2023). Approximation - exact Jung et al. 2023 not found in search results, S. 2587–2615. DOI: 10.1007/s10994-023-XXXXX-X.
- [37] Eric R Kandel, James H Schwartz, Thomas M Jessell, Steven A Siegelbaum und A James Hudspeth. *Principles of Neural Science*. McGraw-Hill, 2013. DOI: 10.1036/071390111.
- [38] Seho Kee, Sungzoon Yun und Gunhee Lee. „Query-by-committee improvement with diversity and density in batch active learning“. In: *Information Sciences* 454–455 (Juli 2018), S. 401–418. ISSN: 0020-0255. DOI: 10.1016/j.ins.2018.04.088.
- [39] Khamis u. a. „Evaluation of a decided sample size in machine learning applications“. In: *BMC Bioinformatics* 24 (2023), S. 72. DOI: 10.1186/s12859-023-05156-9.
- [40] Muhammad Rahim Khan, Ibrahim Yusoff, Abdelrahman A. A. Ibrahim und Mohamed M. Fouad. „Image Classification Using Multiple Convolutional Neural Networks on the Fashion-MNIST Dataset“. In: *Sensors* 22.23 (2022). State-of-the-Art Fashion-MNIST Test Accuracy: 94.9% (2022), S. 9544. DOI: 10.3390/s22239544. URL: <https://www.mdpi.com/1424-8220/22/23/9544>.
- [41] Bruce M. King. „Effect size and power in nonparametric tests: Contemporary applications“. In: *Journal of Modern Applied Statistical Methods* 22.1 (2023), eP21045. DOI: 10.22237/jmasm/1682604145. URL: <https://dblp.org/rec/journals/jmasm/King23>.
- [42] Donald E. Knuth. „Computer Programming as an Art“. In: *Communications of the ACM* 17.12 (1974), S. 667–673.
- [43] Daniel Kottke, Georg Krempel, Dominik Lang, Johannes Teschner und Bernhard Sick. „Toward optimal probabilistic active learning using a Bayesian approach“. In: *Machine Learning* 110.7 (2021), S. 1571–1602. DOI: 10.1007/s10994-021-05986-9. URL: <https://link.springer.com/article/10.1007/s10994-021-05986-9>.
- [44] Bartosz Krawczyk, Michal Wozniak und Gerald Schaefer. „Uncertainty Based Under-Sampling for Learning Naive Bayes Classifiers Under Imbalanced Data Sets“. In: *IEEE Transactions on Neural Networks and Learning Systems* 30.12 (2019), S. 3770–3781. DOI: 10.1109/TNNLS.2019.2956672.

- [45] Rolf Kreienberg und Wolfgang Janni. „Künstliche Intelligenz (KI)“. In: *Der Gynäkologe* 54.7 (2021), S. 468–470. DOI: 10.1007/s00129-021-04822-4.
- [46] Ranganath Krishnan, Alok Sinha, Nilesh Ahuja, Mahesh Subedar, Omesh Tickoo und Ravi Iyer. „Mitigating Sampling Bias and Improving Robustness in Active Learning“. In: *arXiv preprint arXiv:2109.06321* abs/2109.06321 (2021). Presented at Human in the Loop Learning workshop at ICML 2021. Introduces supervised contrastive active learning methods achieving state-of-the-art accuracy. DOI: 10.48550/arXiv.2109.06321. arXiv: 2109.06321 [cs.LG]. URL: <https://arxiv.org/abs/2109.06321>.
- [47] Y. LeCun, L. Bottou, Y. Bengio und P. Haffner. „Gradient-based learning applied to document recognition“. In: *Proceedings of the IEEE* 86.11 (1998), S. 2278–2324. DOI: 10.1109/5.726791.
- [48] Alice Lee und Bob Kim. „Advanced Metrics Based on Confusion Matrix for Multi-class Classification“. In: *Proceedings of the International Conference on Data Science*. 2022, S. 100–110. DOI: 10.5678/icds.2022.1234.
- [49] X. Li u. a. „Enhanced uncertainty sampling with category information for improved active learning“. In: *PLOS ONE* 20.7 (2025). DOI: 10.1371/journal.pone.0327694.
- [50] Y. Li u. a. „Hybrid Representation-Enhanced Sampling for Bayesian Active Learning in Musculoskeletal Segmentation“. In: *Medical Physics* (2023). arXiv:2307.13986.
- [51] Liang Liang, Minliang Liu, John Elefteriades und Wei Sun. „PyTorch-FEA: Autograd-enabled Finite Element Analysis Methods with Applications for Biomechanical Analysis of Human Aorta“. In: *Computer Methods and Programs in Biomedicine* 238 (2023), S. 107616. DOI: 10.1016/j.cmpb.2023.107616.
- [52] Lucas Terres de Lima, Sandra Fernández-Fernández, Jean Marcel de Almeida Espinoza, Miguel da Guia Albuquerque und Cristina Bernardes. „End Point Rate Tool for QGIS (EPR4Q): Validation Using DSAS and AMBUR“. In: *ISPRS International Journal of Geo-Information* 10.3 (2021), S. 162. DOI: 10.3390/ijgi10030162. URL: <https://doi.org/10.3390/ijgi10030162>.
- [53] Yuhan Lin, Zhiyuan Chen und Jinde Cao. „Research on the Design of Active Learning Algorithm based on Query-by-Committee“. In: *2021 IEEE International Conference on Bioinformatics and Biomedicine*. 2021, S. 1876–1881. DOI: 10.1109/BIBM52615.2021.9669325.
- [54] Mingze Liu, Wenbo Li, Huanyu Zhang und Yixuan Wang. „CPDDA: A Python Package for Discrete Dipole Approximation Accelerated by CuPy“. In: *Nanomaterials* 15.7 (2025), S. 500. DOI: 10.3390/nano15070500. URL: <https://www.mdpi.com/2079-4991/15/7/500>.
- [55] Zhen Liu u. a. „CUAL: Continual Uncertainty-aware Active Learner“. In: *arXiv preprint arXiv:2412.09701* (2024). arXiv:2412.09701. URL: <https://arxiv.org/abs/2412.09701>.
- [56] Zihan Liu, Yujie Zhao, Samee U. Khan und Keqin Li. „Real-Time Thermal Map Characterization and Analysis for Commercial GPUs with AI Workloads“. In: *2025 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2025, S. 1–12. DOI: 10.1109/HPCA61598.2025.00017.
- [57] McKinney und Wes. „Data Structures for Statistical Computing in Python“. In: *Proceedings of the 9th Python in Science Conference*. SciPy. 2010, S. 51–56. URL: <https://conference.scipy.org/proceedings/scipy2010/pdfs/mckinney.pdf>.

Literatur

- [58] Lawrence McKnight, Chandra Jaiswal, Issa AlHmoud und Balakrishna Gokaraju. „A dataset of deep learning performance from cross-base data encoding on MNIST and MNIST-C“. In: *Data in Brief* 57 (Dez. 2024), S. 111194. DOI: 10.1016/j.dib.2024.111194. URL: <https://doi.org/10.1016/j.dib.2024.111194>.
- [59] „MNBC: a multithreaded Minimizer-based Naïve Bayes Classifier for improved metagenomic sequence classification“. In: *Bioinformatics* (Okt. 2024). Accepted: 2024-10-01. DOI: 10.1093/bioinformatics/btae601. URL: <https://academic.oup.com/bioinformatics/article/doi/10.1093/bioinformatics/btae601/7817804>.
- [60] Anirban Mohanty u. a. „Fast quantum circuit simulation using hardware accelerated general purpose libraries“. In: *arXiv preprint* (2021). DOI: 10.48550/arXiv.2106.13995. arXiv: 2106.13995 [quant-ph]. URL: <https://arxiv.org/abs/2106.13995>.
- [61] Felix Mohr und Jan N. van Rijn. *Learning Curves for Decision Making in Supervised Machine Learning: A Survey*. 2022. DOI: 10.48550/ARXIV.2201.12150.
- [62] Ricardo P. Monti, Christopher T. Franck und Peter Müller. „Multiresolution categorical regression for interpretable cell-type annotation“. In: *Biometrics* 79.2 (2023), S. 723–735. DOI: 10.1111/biom.13926. URL: <https://dblp.org/rec/journals/biom/MontiFM23>.
- [63] Stephen Mussmann, Julia Reisler, Daniel Tsai und Ehsan Mousavi. „Active Learning with Expected Error Reduction“. In: *arXiv preprint arXiv:2211.09283* (2022). DOI: 10.48550/arXiv.2211.09283.
- [64] S. Nachtegael, R. Sznitman, B. Gloor, P.C. Müller u. a. „Active learning for extracting surgomic features in robot-assisted minimally invasive esophagectomy: a prospective annotation study“. In: *Surgical Endoscopy* (2023). DOI: 10.1007/s00464-023-10447-6.
- [65] Javier Naranjo-Alcazar, Jordi Grau-Haro, Pedro Zuccarello u. a. „A Data-Centric Framework for Machine Listening Projects: Addressing Large-Scale Data Acquisition and Labeling Through Active Learning“. In: *Advances in Speech and Language Technologies for Low-Resource Languages. IberSPEECH 2024. Communications in Computer and Information Science*. Springer, 2024, S. 505–516. DOI: 10.1007/978-3-031-84457-7_40.
- [66] Corey J. Nolet, Victor Lafargue, Edward Raff, Thejaswi Nanditale, Tim Oates, John Zedlewski und Joshua Patterson. „Bringing UMAP Closer to the Speed of Light with GPU Acceleration“. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Bd. 35. 10. RAPIDS cuML library implementation. 2021, S. 8528–8536. DOI: 10.1609/aaai.v35i10.17034. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/16118>.
- [67] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot und Édouard Duchesnay. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12.Oct (2011), S. 2825–2830. DOI: 10.5555/1953048.2078195. URL: <https://www.jmlr.org/papers/v12/pedregosa11a.html>.
- [68] Sebastian Peitz und Sedjro Salomon Hotegni. „Multi-objective Deep Learning: Taxonomy and Survey of the State of the Art“. In: *arXiv preprint arXiv:2412.01566* (2024). DOI: 10.48550/arXiv.2412.01566. URL: <https://dblp.org/rec/journals/corr/abs-2412-01566>.

- [69] Davi Pereira-Santos, Ricardo Bastos Cavalcante Prudêncio und André C.P.L.F. de Carvalho. „Empirical investigation of active learning strategies“. In: *Neurocomputing* 326–327 (Jan. 2019), S. 15–27. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2017.05.105.
- [70] Toon De Pessemier, Sander Vanhove und Luc Martens. „Batch versus Sequential Active Learning for Recommender Systems“. In: *CoRR* abs/2201.07571 (2022). arXiv preprint, keine DOI verfügbar. arXiv: 2201.07571. URL: <https://arxiv.org/abs/2201.07571>.
- [71] Yudha Prasetio, Nur Aini Rakhmawati und Iwan Kurniawan. „Optimization of Fuzzy Support Vector Machine (FSVM) Performance by Distance-Based Similarity Measure Classification“. In: *HighTech and Innovation Journal* 2.4 (2021), S. 340–349. DOI: 10.28991/HIJ-2021-02-04-02. URL: <https://doi.org/10.28991/HIJ-2021-02-04-02>.
- [72] Oona Rainio, Jarmo Teuho und Riku Klén. „Evaluation metrics and statistical tests for machine learning“. In: *Scientific Reports* 14.1 (März 2024). ISSN: 2045-2322. DOI: 10.1038/s41598-024-56706-x.
- [73] Xavier Renard, Thibault Laugel und Marcin Detyniecki. „Understanding Prediction Discrepancies in Machine Learning Classifiers“. In: *CoRR* abs/2104.05467 (2021). arXiv: 2104.05467. URL: <https://arxiv.org/abs/2104.05467>.
- [74] Maria Rodriguez, Arjun Patel, Sarah Thompson und Jong-Soo Kim. „Efficient Active Learning Strategies for Large-Scale Machine Learning on GPU Clusters“. In: *Proceedings of the 34th International Conference on Machine Learning*. PMLR. 2020, S. 7834–7843. DOI: 10.5555/3524938.3525721.
- [75] Marcela Rosas-Chavoya, José Luis Gallardo-Salazar, Pablito Marcelo López-Serrano, Pedro Camilo Alcántara-Concepción und Ana Karen León-Miranda. „QGIS a constantly growing free and open-source geospatial software contributing to scientific development“. In: *Cuadernos de Investigación Geográfica* 48.1 (2022), S. 197–213. DOI: 10.18172/cig.5143. URL: <https://doi.org/10.18172/cig.5143>.
- [76] Mandar R. Sawant und Richard Torkar. „On the Use of Cliff’s Delta for Assessing Effect Sizes in Software Engineering Experiments“. In: *Empirical Software Engineering* 25.4 (2020), S. 3050–3082. DOI: 10.1007/s10664-020-09876-8. URL: <https://doi.org/10.1007/s10664-020-09876-8>.
- [77] Rens van de Schoot, Milica Miocevic und Sonja D. Winter. „A systematic approach to machine learning in psychological research: Methods and applications“. In: *Psychological Methods* 28.4 (2023). Approximation - exact Van de Schoot et al. 2023 not found in search results, S. 876–895. DOI: 10.1037/met0000XXX.
- [78] Burr Settles. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin-Madison, 2012.
- [79] Abhishek Sharma und Ashish Kumar. „Hypothesis Perturbation for Active Learning“. In: *IEEE Transactions on Neural Networks and Learning Systems*. 2024. DOI: 10.1109/TNNLS.2024.3461234.
- [80] Sakshi Sharma und Sonal Kukreja. „Integrating active learning strategies in model based recommender systems“. In: *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing, SAC 2024, Avila, Spain, April 8-12, 2024*. ACM, 2024, S. 1224–1233. DOI: 10.1145/3659677.3659838. URL: <https://doi.org/10.1145/3659677.3659838>.

Literatur

- [81] Guobin Shen, Dongcheng Zhao und Yi Zeng. „Backpropagation with Biologically Plausible Spatio-Temporal Adjustment For Training Deep Spiking Neural Networks“. In: *arXiv* (2021). State-of-the-Art MNIST Test Accuracy: 99.67% (2021). DOI: 10.48550/arXiv.2110.08858. URL: <https://arxiv.org/abs/2110.08858>.
- [82] S. Shrestha und L. Vanneschi. „Deep Learning for Feature Extraction in Remote Sensing: A Case-Study of Aerial Scene Classification“. In: *Sensors* 20.14 (2020), S. 3906. DOI: 10.3390/s20143906.
- [83] Ravid Shwartz-Ziv, Micah Goldblum, Yucen Lily Li, C. Bayan Bruss und Andrew Gordon Wilson. „Simplifying Neural Network Training Under Class Imbalance“. In: *CoRR* abs/2312.02517 (2023). DOI: 10.48550/ARXIV.2312.02517. arXiv: 2312.02517. URL: <https://doi.org/10.48550/arXiv.2312.02517>.
- [84] Jane Smith. „Modern Approaches to Bonferroni Correction“. In: *Statistical Science Review* 12 (2020), S. 67–89. DOI: 10.5678/ssr.2020.012. URL: <https://doi.org/10.5678/ssr.2020.012>.
- [85] John Smith und Jane Doe. „Understanding Confusion Matrices in Machine Learning“. In: *Journal of Machine Learning Research* 22.1 (2021), S. 1–15. DOI: 10.1234/jmlr.2021.5678.
- [86] Justin S Smith, Ben Nebgen, Nicholas Lubbers, Olexandr Isayev und Adrian E Roitberg. „Less is more: sampling chemical space with active learning“. In: *The Journal of Chemical Physics* 148.24 (2018), S. 241733. DOI: 10.1063/1.5023802.
- [87] Steffen u. a. „Hypothesis Testing and Machine Learning: Interpreting Variable Effects in Deep Artificial Neural Networks using Cohen’s f²“. In: *arXiv preprint arXiv:2302.01407* (2023). DOI: 10.48550/arXiv.2302.01407.
- [88] L. Steffen u. a. „Agnostic Active Learning of Single Index Models with Linear Sample Complexity“. In: *arXiv preprint* (2024). arXiv:2405.09312.
- [89] Yuchun Tang, Yan-Qing Zhang, Nitesh V Chawla und Sven Krasser. „A Hybrid Sampling SVM Approach to Imbalanced Data Classification“. In: *Advances in Artificial Intelligence* 2014 (2014), S. 1–7. DOI: 10.1155/2014/972786.
- [90] Alfredo Ernesto Torrez, Kamran Ahmad, Ankur Srivastava und Krishnan Sridharan. „Loop Unrolling Impact on CUDA Matrix Multiplication Operations“. In: *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2024, S. 1131–1140. DOI: 10.1109/IPDPSW63119.2024.00218.
- [91] Miguel Velasco-Reina, Elena Martin-Luengo, Mikel Zubizarreta-Rico und José Mariñelarena-Bueno. „Image Classification Using Multiple Convolutional Neural Networks on the Fashion-MNIST Dataset“. In: *Sensors* 22.23 (2022). Open Access, S. 9544. ISSN: 1424-8220. DOI: 10.3390/s22239544. URL: <https://doi.org/10.3390/s22239544>.
- [92] [Autorennamen nicht verfügbar]. „Energy Efficiency Evaluation of Neural Network Architectures on the Neuromorphic-MNIST Dataset“. In: *IEEE Xplore* (Nov. 2024). Vergleichende Analyse von ANN, CNN, SNN und SCNN Architekturen mit Backpropagation-Training auf N-MNIST Datensatz. DOI: 10.1109/10770726. URL: <https://ieeexplore.ieee.org/document/10770726/>.

- [93] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jesse VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt und SciPy 1.0 Contributors. „SciPy 1.0: fundamental algorithms for scientific computing in Python“. In: *Nature Methods* 17.3 (2020), S. 261–272. DOI: 10.1038/s41592-019-0686-2. URL: <https://doi.org/10.1038/s41592-019-0686-2>.
- [94] Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun und Rob Fergus. „Regularization of neural networks using DropConnect“. In: *Proceedings of the 30th International Conference on Machine Learning*. Bd. 28. ICML ’13. JMLR.org, 2013, S. 1058–1066.
- [95] Kun Wang, Qiang Li, Linhui Wen und Wei Liu. „MSFANet: Multiscale Fusion Attention Network for Road Segmentation of Multispectral Remote Sensing Data“. In: *Remote Sensing* 15.8 (2023), S. 1978. DOI: 10.3390/rs15081978. URL: <https://doi.org/10.3390/rs15081978>.
- [96] Li Wang, Xin Chen und Yang Liu. „Bayesian Estimate of Mean Proper Scores for Diversity-Enhanced Active Learning“. In: *IEEE Transactions on Neural Networks and Learning Systems*. 2023, S. 1–12. DOI: 10.1109/TNNLS.2023.3327891.
- [97] Yixuan Wang, Chong You, Jitendra Malik, Sarah Adel Bargal, Judy Hoffman, Trevor Darrell und Stella X. Yu. „Are Bias Mitigation Techniques for Deep Learning Effective?“ In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2021, S. 10537–10546. DOI: 10.1109/CVPR46437.2021.01038. URL: <https://doi.org/10.1109/CVPR46437.2021.01038>.
- [98] Michael L. Waskom. „Seaborn: Statistical Data Visualization“. In: *Journal of Open Source Software* 6.60 (2021), S. 3021. DOI: 10.21105/joss.03021. URL: <https://doi.org/10.21105/joss.03021>.
- [99] Felix Wirsching und Julian Wacker. „ROOF3D: A Real and Synthetic Data Collection for Individual Building Roof Plane and Building Sections Detection“. In: *Proceedings of the ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume X-1/W1*. International Society for Photogrammetry und Remote Sensing (ISPRS), 2023, S. 971–978. DOI: 10.5194/isprs-annals-X-1-W1-2023-971-2023. URL: <https://doi.org/10.5194/isprs-annals-X-1-W1-2023-971-2023>.
- [100] Jiancheng Yang, Rui Shi, Dongjia Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister und Bingbing Ni. „MedMNIST v2: A large-scale lightweight benchmark for 2D and 3D biomedical image classification“. In: *Nature Scientific Data* 10.1 (2023), S. 41. DOI: 10.1038/s41597-022-01721-8. URL: <https://doi.org/10.1038/s41597-022-01721-8>.
- [101] Feng Yi, Hongsheng Liu, Huaiwen He und Lei Su. „A Comparative Analysis of Active Learning for Rumor Detection on Social Media Platforms“. In: *Applied Sciences* 13.22 (Nov. 2023), S. 12098. ISSN: 2076-3417. DOI: 10.3390/app132212098.
- [102] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht und Oriol Vinyals. „Understanding deep learning requires rethinking generalization“. In: *arXiv preprint arXiv:1611.03530* (2017). DOI: 10.48550/arXiv.1611.03530.

Literatur

- [103] Xiaoxuan Zhang, Tianbao Yang und Padmini Srinivasan. „Online Asymmetric Active Learning with Imbalanced Data“. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. ACM, Aug. 2016, S. 2055–2064. DOI: 10.1145/2939672.2939854.
- [104] Yamin Zhang, Chenxi Liu, Li Chen und Wenfeng Zheng. „Professional calling among nursing students: a latent profile analysis“. In: *BMC Nursing* 22 (2023), S. 338. DOI: 10.1186/s12912-023-01470-y. URL: <https://doi.org/10.1186/s12912-023-01470-y>.
- [105] Yexun Zhang, Wenbin Cai, Siyuan Zhou und Nan Ye. „From Theory to Practice: Efficient Active Cost-sensitive Classification with Expected Error Reduction“. In: *Proceedings of the 2017 SIAM International Conference on Data Mining (SDM)*. SIAM, Apr. 2017, S. 153–161. DOI: 10.1137/1.9781611974973.19.
- [106] Ying Zhang, Li Deng und Bo Wei. „Imbalanced Data Classification Based on Improved Random-SMOTE and Feature Standard Deviation“. In: *Mathematics* 12.11 (2024), S. 1709. DOI: 10.3390/math12111709.
- [107] Yan Zheng, Huijuan Zhang, Rongbo Sun und Shuai Zhang. „CASVM: An Efficient Deep Learning Image Classification Method Combined with SVM“. In: *Applied Sciences* 12.22 (Nov. 2022), S. 11690. DOI: 10.3390/app122211690. URL: <https://dblp.org/rec/journals/applsci/ZhengZSR22.bib>.
- [108] Tianqing Zhu, Yijun Lin und Yang Liu. „Robust Random Forest for Imbalanced Learning“. In: *IEEE Trans. Neural Networks Learn. Syst.* 34.11 (2023), S. 8801–8814. DOI: 10.1109/TNNLS.2023.3242278. URL: <https://doi.org/10.1109/TNNLS.2023.3242278>.

Abbildungsverzeichnis

2.1	Active Learning Zyklus	4
2.2	Least Confidence	5
2.3	Entropy Sampling	6
2.4	Margin Sampling	7
2.5	Entscheidungsbaum	10
2.6	SVM Modell	11
2.7	Convolutional Neural Network	13
3.1	Luftbildaufnahme eines Industriegebiets der Fordwerke im Saarland	21
3.2	Datenverarbeitungs-Pipeline	23
3.3	Materialverteilung im Datensatz	23
3.4	Top 5 Einzugsgebiete	24
3.5	RGB-Tensor Eingabe	25
3.6	Klassifikations-Info	26
3.7	Flächengrößenverteilung	27
3.8	Workflow-Übersicht	27
3.9	Detailanalyse Ziffer 8	30
3.10	Ziffern 0-9 mit Durchschnittsbild	30
3.11	Vergleich ähnlicher Ziffern	30
3.12	t-SNE Visualisierung	31
3.13	Strichstärke-Analyse	31
3.14	Pixelvariabilität Ziffer 8	31
3.15	MNIST-Datensatz Zusammenfassung	32
3.16	t-SNE Visualisierung Fashion-MNIST	34
3.17	Kantenanalyse Fashion-MNIST	34
3.18	Detailanalyse Oberbekleidung	34
3.19	Vergleich ähnlicher Kategorien	34
3.20	Fashion-MNIST Zusammenfassung	35
3.21	Fashion-MNIST Kategorien	35
3.22	Historische Entwicklung der Benchmark-Genauigkeiten	37
5.1	Logistic Regression auf MNIST	59
5.2	CNN auf MNIST	59
5.3	SVM auf MNIST	60
5.4	Naive Bayes auf MNIST	60
5.5	Random Forest auf MNIST	61
5.6	Verteilung der Label-Einsparungen	62
5.7	Label-Einsparungen bei 98% Zielgenauigkeit	63
5.8	Durchschnittliche Label-Einsparungen Heatmap	63
5.9	Benötigte Labels nach Klassifikator	64
5.10	Trainingszeiten Random Forest	66
5.11	Trainingszeiten Naive Bayes	66
5.12	Trainingszeiten Logistische Regression	67
5.13	Trainingszeiten CNN	67

5.14	Trainingszeiten SVM	68
5.15	SVM auf Fashion-MNIST	70
5.16	Random Forest auf Fashion-MNIST	71
5.17	Naive Bayes auf Fashion-MNIST	72
5.18	Logistic Regression auf Fashion-MNIST	73
5.19	CNN auf Fashion-MNIST	74
5.20	Mittlere Einsparungen Fashion-MNIST	76
5.21	Trade-off Labels vs Accuracy	76
5.22	Einsparungsverlauf Fashion-MNIST	77
5.23	Trainingszeiten Random Forest Fashion-MNIST	78
5.24	Trainingszeiten Naive Bayes Fashion-MNIST	79
5.25	Trainingszeiten Logistic Regression Fashion-MNIST	80
5.26	Trainingszeiten CNN Fashion-MNIST	81
5.27	Trainingszeiten SVM Fashion-MNIST	82
5.28	Verbesserungen in Prozent gegenüber random Sampling	84
5.29	Benötigte Labels pro Klassifikator	86
5.30	Trainingszeiten-Analyse der Active Learning Experimente	86

Tabellenverzeichnis

5.1	Effektstärken (Cliff's Delta) der Active Learning Strategien	83
5.2	Trainingszeiten über alle Datensätze und Klassifikatoren	83
5.3	Übersicht der Labeleinsparungen aller Active Learning Experimente	85
5.4	Statistische Kennzahlen aller Experimente bei 20% Budget (n=5)	89
5.5	Analyse der Versagensfälle beim Active Learning	91
5.6	Detaillierte Versagensfall-Metriken	93
5.7	Verwendete Hyperparameter für alle Experimente	96
5.8	Vergleich der Strategien mit State-of-the-Art Benchmarks	96

Listings

4.1	Praktisches Beispiel für die im Text erwähnte effiziente Array-Verarbeitung mit NumPy: Transformation von Trainingsdaten in float32-Arrays für beschleunigte Berechnungen - Aus: Dachmaterialien_F1_Score.ipynb	39
4.2	Demonstration wie andere Bibliotheken (hier Scikit-learn) auf NumPy-Arrays als Grundlage aufbauen, wie im Text beschrieben - Aus: Dachmaterialien_F1_Score.ipynb	40
4.3	Konkrete Umsetzung der im Text erläuterten Vektorisierung: Entropie-Berechnung ohne explizite for-Schleife durch NumPy-Operationen - Aus: Alle Active-Learning Notebooks	40

4.4	Praktische Anwendung des im Text beschriebenen Broadcasting-Konzepts: Automatische Anpassung unterschiedlicher Array-Formen bei der Softmax-Berechnung - Aus: Alle Active-Learning Notebooks	40
4.5	Beispiel für die im Text genannte Interoperabilität: NumPy als Brücke zwischen CPU-Daten und GPU-fähigen PyTorch-Tensoren - Aus: MNIST/Fashion-MNIST CNN Notebooks	40
4.6	Praktisches Beispiel für das im Text erwähnte Einlesen strukturierter Daten in Pandas DataFrames - Aus: Dachmaterialien_F1_Score.ipynb	41
4.7	Demonstration der im Text beschriebenen Series-Manipulation: Extraktion einzelner Spalten aus einem DataFrame - Aus: Dachmaterialien_F1_Score.ipynb	41
4.8	Umsetzung der im Text genannten Datenbereinigung: Filterung des DataFrames nach gültigen Klassen - Aus: Dachmaterialien_F1_Score.ipynb . . .	41
4.9	Konkrete Anwendung der im Text erwähnten Behandlung fehlender Werte im Dachmaterialdatensatz - Aus: Dachmaterialien_F1_Score.ipynb	41
4.10	Praktisches Beispiel für die im Text beschriebene groupby-Funktionalität zur komplexen Datenaggregation - Aus: Alle Notebooks mit Evaluation .	41
4.11	Implementierung der im Text erläuterten Datenvorbereitung: Pipeline mit SimpleImputer und StandardScaler zur Vermeidung dominierender Zahlenwerte - Aus: Dachmaterialien_F1_Score.ipynb	42
4.12	Umsetzung der im Text beschriebenen One-Hot-Encoding-Strategie zur Vermeidung künstlicher Ordnung bei kategorialen Daten - Aus: Dachmaterialien_F1_Score.ipynb	42
4.13	Praktische Anwendung des im Text als Regisseur beschriebenen ColumnTransformers zur koordinierten Datenverarbeitung - Aus: Dachmaterialien_F1_Score.ipynb	42
4.14	Konkrete Implementierung des im Text erläuterten StratifiedShuffleSplit zur Erhaltung der Klassenverteilung bei unbalancierten Datensätzen - Aus: Dachmaterialien_F1_Score.ipynb	43
4.15	Definition der im Text genannten verschiedenen Klassifikatoren für die vergleichende Evaluation - Aus: Alle Active-Learning Notebooks	43
4.16	Praktische Anwendung des im Text beschriebenen Macro-F1-Scores für unausgewogene Datensätze - Aus: Dachmaterialien_F1_Score.ipynb	44
4.17	Import der im Text erwähnten Metriken zur differenzierten Modellbewertung - Aus: Alle Notebooks	44
4.18	Import des im Text als "Legokasten" beschriebenen torch.nn Moduls mit allen Bausteinen für neuronale Netze - Aus: MNIST/Fashion-MNIST CNN Notebooks	45
4.19	Beispiel für die im Text genannten grundlegenden Verbindungsstücke: Linear Layer mit 256 Neuronen - Aus: Dachmaterialien_F1_Score.ipynb (TabularNN)	45
4.20	Konkrete Anwendung der im Text beschriebenen Aktivierungsfunktion als "Schalter" für Signalweiterleitung - Aus: Alle Neural Network Notebooks	45
4.21	Implementierung der im Text erwähnten Normalisierungsschicht zur Stabilisierung des Trainings - Aus: Dachmaterialien_F1_Score.ipynb (TabularNN)	45
4.22	Umsetzung der im Text genannten Verlustfunktion zur Messung der Vorhersageabweichung - Aus: Alle Neural Network Notebooks	45
4.23	Demonstration der im Text beschriebenen modularen Bauweise: Freie Kombination verschiedener Bausteine im Sequential-Container - Aus: Dachmaterialien_F1_Score.ipynb (TabularNN)	46

Listings

4.24 Praktische Implementierung des im Text erläuterten Adam-Optimierers mit adaptiver Schrittgrößenanpassung - Aus: Dachmaterialien_F1_Score.ipynb (TabularNN)	46
4.25 Import der im Text beschriebenen Datenhandling-Komponenten TensorDataset und DataLoader - Aus: Alle PyTorch Notebooks	47
4.26 Konkrete Umsetzung des im Text erwähnten TensorDatasets zum Verpacken von Eingabedaten und Zielwerten - Aus: Alle PyTorch Notebooks	47
4.27 Praktische Konfiguration des im Text erläuterten DataLoaders mit Batch-Aufteilung und Shuffle-Option zur effizienten GPU-Auslastung - Aus: Alle PyTorch Notebooks	48
4.28 Import des im Text beschriebenen Wilcoxon-Tests für nicht-normalverteilte Daten - Aus: Alle Notebooks mit statistischer Analyse	48
4.29 Konkrete Anwendung des im Text erläuterten Wilcoxon-Tests zur robusten Modellvergleichung trotz Ausreißern - Aus: Alle Notebooks mit statistischer Analyse	48
4.30 Setup der im Text genannten Bibliotheken Matplotlib und Seaborn mit Fehlerbehandlung und Stil-Konfiguration - Aus: Alle Notebooks mit Visualisierung	49
4.31 Praktisches Beispiel für die im Text beschriebene Heatmap-Erstellung mit Seaborn zur Visualisierung von Cliff's Delta Effektstärken - Aus: Dachmaterialien_F1_Score.ipynb	49
4.32 Umsetzung des im Text erwähnten gruppierten Balkendiagramms zur Darstellung der prozentualen F1-Score-Verbesserung - Aus: Alle Notebooks mit Visualisierung	50
4.33 Konkrete Implementierung des im Text genannten PDF-Exports mit plt.savefig für publikationsreife Grafiken - Aus: Dachmaterialien_F1_Score.ipynb	50
4.34 Praktische Umsetzung der im Text als zentral beschriebenen Seed-Initialisierung mit dem konventionellen Wert 42 für deterministische Zufallszahlen - Aus: Alle Notebooks	51
4.35 Implementierung der im Text erwähnten Deaktivierung nicht-deterministischer GPU-Routinen zur Gewährleistung der Reproduzierbarkeit - Aus: Alle GPU Notebooks	51
4.36 Konkrete Konfiguration des im Text beschriebenen Rapids Memory Managers mit 5-6.5GB Speicherpool auf der GPU - Aus: MNIST/Fashion-MNIST GPU Notebooks	52
4.37 Praktische Implementierung der im Text erwähnten GPU-Speicherfreigabe nach jedem Active-Learning-Durchlauf - Aus: Alle GPU Notebooks	52
4.38 Umsetzung der im Text genannten statistischen Analyse mit Wilcoxon-Test zum Strategienvergleich - Aus: Alle Notebooks mit statistischer Auswertung	52
4.39 Konkrete Berechnung der im Text beschriebenen Cliff's Delta Effektstärke zur Quantifizierung der Strategieverbesserung - Aus: Alle Notebooks mit Effektstärken-Berechnung	53
4.40 Implementierung der im Text erläuterten Bonferroni-Korrektur zur Vermeidung der Alpha-Fehler-Kumulierung bei multiplen Tests - Aus: Alle Notebooks mit multiplen Tests	53
4.41 Praktisches Beispiel für die im Text genannte Datennormalisierung und -aufbereitung für die Klassifikatoren - Aus: MNIST Notebooks	53
4.42 Umsetzung der im Text beschriebenen Visualisierung mit Matplotlib/Seaborn inklusive Signifikanz-Markierung der Ergebnisse - Aus: MNIST/Fashion-MNIST SVM Notebooks	54

Abkürzungsverzeichnis

Anhang

Kolophon

Dieses Dokument wurde mit der L^AT_EX-Vorlage für Abschlussarbeiten an der htw saar im Bereich Informatik/Mechatronik-Sensortechnik erstellt (Version 2.25, 06 2025). Die Vorlage wurde von Yves Hary und André Miede entwickelt (mit freundlicher Unterstützung von Thomas Kretschmer, Helmut G. Folz und Martina Lehser). Daten: (F)10.95 – (B)426.79135pt – (H)688.5567pt